Sequence to Sequence Neural Network Model

An attempt is made to implement Sequence to Sequence (encoder -decoder) model to predict title for the paragraph passed.

In [1]:
```
#Loading Libraries
from __future__ import print_function

import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
from collections import Counter

from keras.models import Model
from keras.layers import Embedding, Dense, Input
from keras.layers.recurrent import LSTM
from keras.preprocessing.sequence import pad_sequences
from keras.callbacks import ModelCheckpoint

import tensorflow as tf
from keras import backend as K
import os
```

```
C:\Users\nitin\AppData\Local\Continuum\anaconda3\lib\site-packages\h5py\__init_
_.py:36: FutureWarning: Conversion of the second argument of issubdtype from `f
loat` to `np.floating` is deprecated. In future, it will be treated as `np.floa
t64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [2]:
```
#Setting up directories

np.random.seed(42)
data_dir_path = './data'
report_dir_path = './reports'
model_dir_path = './models'
very_large_data_dir_path = './very_large_data'
```

In [3]:
```
#Setting up configuration so that GPU is utilised

config = tf.ConfigProto(allow_soft_placement=True,
                        device_count={'CPU': 1, 'GPU': 1})
session = tf.Session(config=config)
K.set_session(session)
```

In [4]:
```
# Reading the Fake or real news dataset
art1 = pd.read_csv("data/fake_or_real_news.csv")
art1.head()
```

Out[4]:

	Unnamed: 0	title	text	label
0	8476	You Can Smell Hillary's Fear	Daniel Greenfield, a Shillman Journalism Fello...	FAKE
1	10294	Watch The Exact Moment Paul Ryan Committed Pol...	Google Pinterest Digg Linkedin Reddit Stumbleu...	FAKE
2	3608	Kerry to go to Paris in gesture of sympathy	U.S. Secretary of State John F. Kerry said Mon...	REAL
3	10142	Bernie supporters on Twitter erupt in anger ag...	— Kaydee King (@KaydeeKing) November 9, 2016 T...	FAKE
4	875	The Battle of New York: Why This Primary Matters	It's primary day in New York and front-runners...	REAL

In [5]:
```
#Check
art1['title'][1]
```

Out[5]:
```
'Watch The Exact Moment Paul Ryan Committed Political Suicide At A Trump Rally (VIDEO)'
```

Let us keep out 64 articles as hold out data, which we would later use to make validation of the model

In [6]:
```
#spliting the data
traindf, hold_out = train_test_split(art1, test_size=.01)


#print out stats about shape of data
print(f'Train: {traindf.shape[0]:,} rows {traindf.shape[1]:,} columns')
print(f'Test: {hold_out.shape[0]:,} rows {hold_out.shape[1]:,} columns')

# preview data
traindf.head(3)
```

```
Train: 6,271 rows 4 columns
Test: 64 rows 4 columns
```

Out[6]:

	Unnamed: 0	title	text	label
1345	2378	Here's why creating single-payer health care i...	The Hillary Clinton campaign is taking some ha...	REAL
6301	9621	Here Are Six 'Miracle' Drugs Big Pharma Now Re...	Here Are Six 'Miracle' Drugs Big Pharma Now Re...	FAKE
2715	5810	Refusal to Acknowledge Uniqueness of Holocaust...	Diversity Macht Frei October 27, 2016 \nThe Je...	FAKE

In [7]:
```
X = traindf['text']
Y = traindf.title
```

We limit the vocabulary size and length of the passage to 8000 and 500 words. Similary for the target (title) we will be considering only 2000 words for vocabulary and length of 50 words.

Glove word embedding of 100 dimensions is considered due to system and memory constraints.

We set the hidden layers to 100, and batch size of 16 for stochastic gradient descent.

In [8]:
```
MAX_INPUT_SEQ_LENGTH = 500
MAX_TARGET_SEQ_LENGTH = 50
MAX_INPUT_VOCAB_SIZE = 8000
MAX_TARGET_VOCAB_SIZE = 2000

#Dimension for the Glove Word Embeddings
GLOVE_EMBEDDING_SIZE = 100

HIDDEN_UNITS = 100  #Hidden Layers
batch_size = 16 #batch size for the stochastic gradient descent
VERBOSE = 1
LOAD_EXISTING_WEIGHTS = True

model_name = 'seq2seq_100L_100D_16'
```

Let us create the configuration file for the model which is fitted on the text. Both for target and input, top most frequently occuring words are assigned with an index and then dictionaries are created for the same.

In [9]:
```
def fit_text(X, Y, input_seq_max_length=None, target_seq_max_length=None):
    if input_seq_max_length is None:
        input_seq_max_length = MAX_INPUT_SEQ_LENGTH
    if target_seq_max_length is None:
        target_seq_max_length = MAX_TARGET_SEQ_LENGTH
    input_counter = Counter()
    target_counter = Counter()
    max_input_seq_length = 0
    max_target_seq_length = 0

    for line in X:
        text = [word.lower() for word in line.split(' ')]
        seq_length = len(text)
        if seq_length > input_seq_max_length:
            text = text[0:input_seq_max_length]
            seq_length = len(text)
        for word in text:
            input_counter[word] += 1
        max_input_seq_length = max(max_input_seq_length, seq_length)

    for line in Y:
        line2 = 'START ' + line.lower() + ' END'
        text = [word for word in line2.split(' ')]
        seq_length = len(text)
        if seq_length > target_seq_max_length:
            text = text[0:target_seq_max_length]
            seq_length = len(text)
        for word in text:
            target_counter[word] += 1
            max_target_seq_length = max(max_target_seq_length, seq_length)

    input_word2idx = dict()
    for idx, word in enumerate(input_counter.most_common(MAX_INPUT_VOCAB_SIZE)):
        input_word2idx[word[0]] = idx + 2
    input_word2idx['PAD'] = 0
    input_word2idx['UNK'] = 1
    input_idx2word = dict([(idx, word) for word, idx in input_word2idx.items()])

    target_word2idx = dict()
    for idx, word in enumerate(target_counter.most_common(MAX_TARGET_VOCAB_SIZE)):
        target_word2idx[word[0]] = idx + 1
    target_word2idx['UNK'] = 0

    target_idx2word = dict([(idx, word) for word, idx in target_word2idx.items()])

    num_input_tokens = len(input_word2idx)
    num_target_tokens = len(target_word2idx)

    config = dict()
    config['input_word2idx'] = input_word2idx
    config['input_idx2word'] = input_idx2word
    config['target_word2idx'] = target_word2idx
    config['target_idx2word'] = target_idx2word
    config['num_input_tokens'] = num_input_tokens
    config['num_target_tokens'] = num_target_tokens
    config['max_input_seq_length'] = max_input_seq_length
```

```
    config['max_target_seq_length'] = max_target_seq_length

    return config
```

In [10]:
```
config = fit_text(X, Y)
```

In [11]:
```
#Function to loading the Glove word embedding data from the path specified
def load_glove(data_dir_path=None):
    if data_dir_path is None:
        data_dir_path = 'very_large_data'
    download_glove(data_dir_path)
    word2em = {}
    glove_model_path = data_dir_path + "/glove.6B." + str(GLOVE_EMBEDDING_SIZE) +
    file = open(glove_model_path, mode='rt', encoding='utf8')
    for line in file:
        words = line.strip().split()
        word = words[0]
        embeds = np.array(words[1:], dtype=np.float32)
        word2em[word] = embeds
    file.close()
    return word2em

def glove_zero_emb():
    return np.zeros(shape=GLOVE_EMBEDDING_SIZE)
```

In [12]:
```
#Assigning values to from config file to variables

max_input_seq_length = config['max_input_seq_length']
num_target_tokens = config['num_target_tokens']
max_target_seq_length = config['max_target_seq_length']
target_word2idx = config['target_word2idx']
target_idx2word = config['target_idx2word']
```

A random vector of 50 dimensions is created is added into the config file, these random is utilised to represent the word that is not present in vocabulary.

In [13]:
```
#Checking for presence of unknown_emb else creating a new random vector and addin
word2em = dict()
if 'unknown_emb' in config:
    unknown_emb = config['unknown_emb']
else:
    unknown_emb = np.random.rand(1, GLOVE_EMBEDDING_SIZE)
    config['unknown_emb'] = unknown_emb
```

Data needs to be transformed into embedding layer form. Each word being replaced with its word vector from the glove embeddings. Any word not present in vocabulary is replaced with a common unknown vector. If the total length of input is less than maximum length of input then it is padded with zeros in the front.

For the target, title is added with 'START' and 'END' at beggining and end of sentence respectively for letting decoder model the start and terminate the prediction of sequence.

In [14]:
```
#Transformation code - text to encoding for the content (input)
def transform_input_text(texts):
    temp = []
    for line in texts:
        x = np.zeros(shape=(max_input_seq_length, GLOVE_EMBEDDING_SIZE))
        for idx, word in enumerate(line.lower().split(' ')):
            if idx >= max_input_seq_length:
                break
            emb = unknown_emb
            if word in word2em:
                emb = word2em[word]
            x[idx, :] = emb
        temp.append(x)
    temp = pad_sequences(temp, maxlen=max_input_seq_length)

    print(temp.shape)
    return temp
```

In [15]:
```
#Transformation code - text to encoding for the title (output)
def transform_target_encoding(texts):
    temp = []
    for line in texts:
        x = []
        line2 = 'START ' + line.lower() + ' END'
        for word in line2.split(' '):
            x.append(word)
            if len(x) >= max_target_seq_length:
                break
        temp.append(x)

    temp = np.array(temp)
    print(temp.shape)
    return temp
```

In [16]:
```
#code to generate the batch samples
def generate_batch(x_samples, y_samples, batch_size):
    num_batches = len(x_samples) // batch_size
    while True:
        for batchIdx in range(0, num_batches):
            start = batchIdx * batch_size
            end = (batchIdx + 1) * batch_size
            encoder_input_data_batch = pad_sequences(x_samples[start:end], max_in
            decoder_target_data_batch = np.zeros(shape=(batch_size, max_target_se
            decoder_input_data_batch = np.zeros(shape=(batch_size, max_target_seq
            for lineIdx, target_words in enumerate(y_samples[start:end]):
                for idx, w in enumerate(target_words):
                    w2idx = 0  # default [UNK]
                    if w in target_word2idx:
                        w2idx = target_word2idx[w]
                    if w2idx != 0:
                        decoder_input_data_batch[lineIdx, idx, w2idx] = 1
                        if idx > 0:
                            decoder_target_data_batch[lineIdx, idx - 1, w2idx] =
            yield [encoder_input_data_batch, decoder_input_data_batch], decoder_t
```

In [17]:
```
#Splitting the training and validation data

Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, Y, test_size=0.2, random_state

print('training size: ', len(Xtrain))
print('testing size: ', len(Xtest))
```

```
training size:  5016
testing size:  1255
```

Setting up the model

In [18]:
```
##### Define Model Architecture ######

########################
#### Encoder Model ####
encoder_inputs = Input(shape=(None, GLOVE_EMBEDDING_SIZE), name='encoder_inputs')
encoder_lstm = LSTM(units=HIDDEN_UNITS, return_state=True, name='encoder_lstm')
encoder_outputs, encoder_state_h, encoder_state_c = encoder_lstm(encoder_inputs)
encoder_states = [encoder_state_h, encoder_state_c]

########################
#### Decoder Model ####
decoder_inputs = Input(shape=(None, num_target_tokens), name='decoder_inputs')
decoder_lstm = LSTM(units=HIDDEN_UNITS, return_state=True, return_sequences=True,
decoder_outputs, decoder_state_h, decoder_state_c = decoder_lstm(decoder_inputs,
                                                                 initial_state=en

# Dense layer for prediction
decoder_dense = Dense(units=num_target_tokens, activation='softmax', name='decode
decoder_outputs = decoder_dense(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['acc

########################
#### Seq2Seq Model ####
encoder_model = Model(encoder_inputs, encoder_states)

decoder_state_inputs = [Input(shape=(HIDDEN_UNITS,)), Input(shape=(HIDDEN_UNITS,)
decoder_outputs, state_h, state_c = decoder_lstm(decoder_inputs, initial_state=de
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = Model([decoder_inputs] + decoder_state_inputs, [decoder_outputs]
```

In [19]:
```
model.summary()
```

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

In [20]:
```
#Saving the config and model architecure

config_file_path = model_dir_path + '/' + model_name + '-config.npy'
architecture_file_path = model_dir_path + '/' + model_name + '-architecture.json'

np.save(config_file_path, config)
open(architecture_file_path, 'w').write(model.to_json())
```

```
C:\Users\nitin\AppData\Local\Continuum\anaconda3\lib\site-packages\keras\engine
\network.py:877: UserWarning: Layer decoder_lstm was passed non-serializable ke
yword arguments: {'initial_state': [<tf.Tensor 'encoder_lstm/while/Exit_2:0' sh
ape=(?, 100) dtype=float32>, <tf.Tensor 'encoder_lstm/while/Exit_3:0' shape=(?,
100) dtype=float32>]}. They will not be included in the serialized model (and t
hus will be missing at deserialization time).
  '. They will not be included '
```

Out[20]: 3098

In [21]:
```
#Transforming the data
Ytrain = transform_target_encoding(Ytrain)
Ytest = transform_target_encoding(Ytest)

Xtrain = transform_input_text(Xtrain)
Xtest = transform_input_text(Xtest)
```

```
(5016,)
(1255,)
(5016, 500, 100)
(1255, 500, 100)
```

In [22]:
```
#Setting the number of training batches based on batch size
train_num_batches = len(Xtrain) // batch_size
test_num_batches = len(Xtest) // batch_size
```

In [23]:
```
#Generating the batches data
train_gen = generate_batch(Xtrain, Ytrain, batch_size)
test_gen = generate_batch(Xtest, Ytest, batch_size)
```

In [24]:
```
#Assigning weights save path and checkpoint for model to save weights after every
weight_file_path = model_dir_path + '/' + model_name + 'weights.{epoch:02d}-{val_
checkpoint = ModelCheckpoint(weight_file_path)
```

In [25]:
```
#Traning the Model
history = model.fit_generator(generator=train_gen, steps_per_epoch=train_num_batc
                                                epochs= 30,
                                                verbose=VERBOSE, validation_data=test_
                                                callbacks=[checkpoint])
```

```
Epoch 1/30
313/313 [==============================] - 182s 580ms/step - loss: 1.0615 - ac
c: 0.0219 - val_loss: 1.0397 - val_acc: 0.0231

C:\Users\nitin\AppData\Local\Continuum\anaconda3\lib\site-packages\keras\engine
\network.py:877: UserWarning: Layer decoder_lstm was passed non-serializable ke
yword arguments: {'initial_state': [<tf.Tensor 'encoder_lstm/while/Exit_2:0' sh
ape=(?, 100) dtype=float32>, <tf.Tensor 'encoder_lstm/while/Exit_3:0' shape=(?,
100) dtype=float32>]}. They will not be included in the serialized model (and t
hus will be missing at deserialization time).
  '. They will not be included '

Epoch 2/30
313/313 [==============================] - 176s 564ms/step - loss: 1.0324 - ac
c: 0.0227 - val_loss: 1.0323 - val_acc: 0.0229
Epoch 3/30
313/313 [==============================] - 179s 572ms/step - loss: 1.0197 - ac
c: 0.0230 - val_loss: 1.0282 - val_acc: 0.0236
Epoch 4/30
313/313 [==============================] - 177s 566ms/step - loss: 1.0091 - ac
c: 0.0237 - val_loss: 1.0237 - val_acc: 0.0244
Epoch 5/30
313/313 [==============================] - 177s 567ms/step - loss: 0.9947 - ac
c: 0.0246 - val_loss: 1.0210 - val_acc: 0.0252
Epoch 6/30
313/313 [==============================] - 178s 568ms/step - loss: 0.9880 - ac
c: 0.0255 - val_loss: 1.0132 - val_acc: 0.0264
Epoch 7/30
313/313 [==============================] - 180s 575ms/step - loss: 0.9676 - ac
c: 0.0271 - val_loss: 1.0036 - val_acc: 0.0270
Epoch 8/30
313/313 [==============================] - 185s 590ms/step - loss: 0.9516 - ac
c: 0.0277 - val_loss: 0.9992 - val_acc: 0.0279
Epoch 9/30
313/313 [==============================] - 180s 574ms/step - loss: 0.9400 - ac
c: 0.0285 - val_loss: 0.9934 - val_acc: 0.0284
Epoch 10/30
313/313 [==============================] - 179s 571ms/step - loss: 0.9310 - ac
c: 0.0293 - val_loss: 0.9973 - val_acc: 0.0286
Epoch 11/30
313/313 [==============================] - 178s 569ms/step - loss: 0.9211 - ac
c: 0.0300 - val_loss: 0.9892 - val_acc: 0.0293
Epoch 12/30
313/313 [==============================] - 177s 567ms/step - loss: 0.9074 - ac
c: 0.0309 - val_loss: 0.9855 - val_acc: 0.0298
Epoch 13/30
313/313 [==============================] - 177s 566ms/step - loss: 0.8961 - ac
c: 0.0317 - val_loss: 0.9851 - val_acc: 0.0300
Epoch 14/30
313/313 [==============================] - 178s 567ms/step - loss: 0.8865 - ac
c: 0.0326 - val_loss: 0.9913 - val_acc: 0.0297
```

```
Epoch 15/30
313/313 [==============================] - 178s 570ms/step - loss: 0.8785 - ac
c: 0.0332 - val_loss: 0.9847 - val_acc: 0.0306
Epoch 16/30
313/313 [==============================] - 181s 580ms/step - loss: 0.8674 - ac
c: 0.0340 - val_loss: 0.9833 - val_acc: 0.0305
Epoch 17/30
313/313 [==============================] - 177s 565ms/step - loss: 0.8577 - ac
c: 0.0349 - val_loss: 0.9856 - val_acc: 0.0307
Epoch 18/30
313/313 [==============================] - 177s 565ms/step - loss: 0.8496 - ac
c: 0.0357 - val_loss: 0.9874 - val_acc: 0.0309
Epoch 19/30
313/313 [==============================] - 177s 566ms/step - loss: 0.8419 - ac
c: 0.0365 - val_loss: 0.9867 - val_acc: 0.0306
Epoch 20/30
313/313 [==============================] - 176s 562ms/step - loss: 0.8336 - ac
c: 0.0374 - val_loss: 0.9873 - val_acc: 0.0303
Epoch 21/30
313/313 [==============================] - 177s 566ms/step - loss: 0.8260 - ac
c: 0.0382 - val_loss: 0.9900 - val_acc: 0.0302
Epoch 22/30
313/313 [==============================] - 177s 565ms/step - loss: 0.8172 - ac
c: 0.0389 - val_loss: 0.9922 - val_acc: 0.0298
Epoch 23/30
313/313 [==============================] - 177s 565ms/step - loss: 0.8097 - ac
c: 0.0398 - val_loss: 0.9966 - val_acc: 0.0291
Epoch 24/30
313/313 [==============================] - 177s 566ms/step - loss: 0.8036 - ac
c: 0.0404 - val_loss: 0.9963 - val_acc: 0.0299
Epoch 25/30
313/313 [==============================] - 177s 565ms/step - loss: 0.7961 - ac
c: 0.0411 - val_loss: 1.0005 - val_acc: 0.0282
Epoch 26/30
313/313 [==============================] - 177s 566ms/step - loss: 0.7902 - ac
c: 0.0418 - val_loss: 0.9986 - val_acc: 0.0292
Epoch 27/30
313/313 [==============================] - 177s 564ms/step - loss: 0.7824 - ac
c: 0.0428 - val_loss: 1.0031 - val_acc: 0.0284
Epoch 28/30
313/313 [==============================] - 177s 564ms/step - loss: 0.7765 - ac
c: 0.0434 - val_loss: 1.0055 - val_acc: 0.0293
Epoch 29/30
313/313 [==============================] - 177s 565ms/step - loss: 0.7696 - ac
c: 0.0439 - val_loss: 1.0082 - val_acc: 0.0281
Epoch 30/30
313/313 [==============================] - 176s 563ms/step - loss: 0.7640 - ac
c: 0.0445 - val_loss: 1.0089 - val_acc: 0.0287
```

In [42]:
```
model.load_weights("./models/seq2seq_100L_100D_16weights.20-1.05.hdf5")
```

In [43]:
```
#Traning the Model
history = model.fit_generator(generator=train_gen, steps_per_epoch=train_num_batc
                                                epochs= 20,
                                                verbose=VERBOSE, validation_data=test_
                                                callbacks=[checkpoint])
```

```
Epoch 1/20
313/313 [==============================] - 179s 572ms/step - loss: 0.7584 - ac
c: 0.0454 - val_loss: 1.0081 - val_acc: 0.0299
Epoch 2/20

C:\Users\nitin\AppData\Local\Continuum\anaconda3\lib\site-packages\keras\engine
\network.py:877: UserWarning: Layer decoder_lstm was passed non-serializable ke
yword arguments: {'initial_state': [<tf.Tensor 'encoder_lstm/while/Exit_2:0' sh
ape=(?, 100) dtype=float32>, <tf.Tensor 'encoder_lstm/while/Exit_3:0' shape=(?,
100) dtype=float32>]}. They will not be included in the serialized model (and t
hus will be missing at deserialization time).
  '. They will not be included '

313/313 [==============================] - 178s 570ms/step - loss: 0.7508 - ac
c: 0.0464 - val_loss: 1.0096 - val_acc: 0.0285
Epoch 3/20
313/313 [==============================] - 176s 562ms/step - loss: 0.7468 - ac
c: 0.0470 - val_loss: 1.0117 - val_acc: 0.0295
Epoch 4/20
313/313 [==============================] - 175s 558ms/step - loss: 0.7404 - ac
c: 0.0474 - val_loss: 1.0126 - val_acc: 0.0295
Epoch 5/20
313/313 [==============================] - 175s 560ms/step - loss: 0.7342 - ac
c: 0.0486 - val_loss: 1.0196 - val_acc: 0.0288
Epoch 6/20
313/313 [==============================] - 175s 559ms/step - loss: 0.7293 - ac
c: 0.0492 - val_loss: 1.0186 - val_acc: 0.0295
Epoch 7/20
313/313 [==============================] - 174s 557ms/step - loss: 0.7253 - ac
c: 0.0497 - val_loss: 1.0202 - val_acc: 0.0287
Epoch 8/20
313/313 [==============================] - 175s 560ms/step - loss: 0.7188 - ac
c: 0.0508 - val_loss: 1.0301 - val_acc: 0.0287
Epoch 9/20
313/313 [==============================] - 175s 559ms/step - loss: 0.7186 - ac
c: 0.0510 - val_loss: 1.0266 - val_acc: 0.0285
Epoch 10/20
313/313 [==============================] - 177s 566ms/step - loss: 0.7097 - ac
c: 0.0521 - val_loss: 1.0288 - val_acc: 0.0284
Epoch 11/20
313/313 [==============================] - 175s 560ms/step - loss: 0.7028 - ac
c: 0.0533 - val_loss: 1.0325 - val_acc: 0.0293
Epoch 12/20
313/313 [==============================] - 175s 559ms/step - loss: 0.6982 - ac
c: 0.0537 - val_loss: 1.0344 - val_acc: 0.0278
Epoch 13/20
313/313 [==============================] - 175s 559ms/step - loss: 0.6945 - ac
c: 0.0542 - val_loss: 1.0354 - val_acc: 0.0282
Epoch 14/20
313/313 [==============================] - 175s 560ms/step - loss: 0.6919 - ac
c: 0.0549 - val_loss: 1.0389 - val_acc: 0.0293
```

```
Epoch 15/20
313/313 [==============================] - 176s 562ms/step - loss: 0.6867 - ac
c: 0.0554 - val_loss: 1.0416 - val_acc: 0.0292
Epoch 16/20
313/313 [==============================] - 175s 559ms/step - loss: 0.6819 - ac
c: 0.0562 - val_loss: 1.0456 - val_acc: 0.0286
Epoch 17/20
313/313 [==============================] - 176s 563ms/step - loss: 0.6771 - ac
c: 0.0569 - val_loss: 1.0507 - val_acc: 0.0276
Epoch 18/20
313/313 [==============================] - 177s 567ms/step - loss: 0.6738 - ac
c: 0.0575 - val_loss: 1.0539 - val_acc: 0.0275
Epoch 19/20
313/313 [==============================] - 175s 560ms/step - loss: 0.6680 - ac
c: 0.0582 - val_loss: 1.0531 - val_acc: 0.0279
Epoch 20/20
313/313 [==============================] - 176s 564ms/step - loss: 0.6625 - ac
c: 0.0591 - val_loss: 1.0550 - val_acc: 0.0280
```

In [53]:
```
model.load_weights("./models/seq2seq_100L_100D_16weights.20-1.05.hdf5")
```

In [ ]:
```
#Traning the Model
history = model.fit_generator(generator=train_gen, steps_per_epoch=train_num_batc
                                                epochs= 150,
                                                verbose=VERBOSE, validation_data=test_
                                                callbacks=[checkpoint])
```

```
Epoch 1/150
313/313 [==============================] - 176s 561ms/step - loss: 0.6595 - a
cc: 0.0595 - val_loss: 1.0537 - val_acc: 0.0278
Epoch 2/150

C:\Users\nitin\AppData\Local\Continuum\anaconda3\lib\site-packages\keras\engi
ne\network.py:877: UserWarning: Layer decoder_lstm was passed non-serializabl
e keyword arguments: {'initial_state': [<tf.Tensor 'encoder_lstm/while/Exit_
2:0' shape=(?, 100) dtype=float32>, <tf.Tensor 'encoder_lstm/while/Exit_3:0'
shape=(?, 100) dtype=float32>]}. They will not be included in the serialized
model (and thus will be missing at deserialization time).
  '. They will not be included '
```

In [27]:
```
def summarize(input_text):
        input_seq = np.zeros(shape=(1, max_input_seq_length, GLOVE_EMBEDDING_SIZE
        for idx, word in enumerate(input_text.lower().split(' ')):
            if idx >= max_input_seq_length:
                break
            emb = unknown_emb  # default [UNK]
            if word in word2em:
                emb = word2em[word]
            input_seq[0, idx, :] = emb
        states_value = encoder_model.predict(input_seq)
        target_seq = np.zeros((1, 1, num_target_tokens))
        target_seq[0, 0, target_word2idx['START']] = 1
        target_text = ''
        target_text_len = 0
        terminated = False
        while not terminated:
            output_tokens, h, c = decoder_model.predict([target_seq] + states_val

            sample_token_idx = np.argmax(output_tokens[0, -1, :])
            sample_word = target_idx2word[sample_token_idx]
            target_text_len += 1

            if sample_word != 'START' and sample_word != 'END':
                target_text += ' ' + sample_word

            if sample_word == 'END' or target_text_len >= max_target_seq_length:
                terminated = True

            target_seq = np.zeros((1, 1, num_target_tokens))
            target_seq[0, 0, sample_token_idx] = 1

            states_value = [h, c]
        return target_text.strip()
```

In [20]:
```
print('start predicting ...')
for i in np.random.permutation(np.arange(len(X)))[0:20]:
    x = X[i]
    actual_headline = Y[i]
    headline = summarize(x)

    print('Generated Headline: ', headline)
    print('Original Headline: ', actual_headline)
```

```
Generated Headline:  california today: today: the u.s. in the the world
Original Headline:  Bill Paxton, Star of ‘Big Love’ and Movie Blockbusters, Die
s at 61
Generated Headline:  north and china is killed by the watch: china
Original Headline:  Oakland Fire Victims Included Performers, Educators and Law
yers
Generated Headline:  china says u.s. american american have
Original Headline:  ‘Hamilton’ Inc.: The Path to a Billion-Dollar Broadway 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 Friends
Generated Headline:  breitbart news daily: daily: trump
Original Headline:  The Budget Funds 99 Things and a Wall Ain’t One
Generated Headline:  to donald trump and a new york ban
Original Headline:  Cities Vow to Fight Trump on Immigration, Even if They Lose
Millions
Generated Headline:  watch: attorney general is not to be days
Original Headline:  WATCH: Anti-MILO Protesters Tear Down Barricades At UC Davi
s
Generated Headline:  china and china is charged in china
Original Headline:  Egyptian Court Clears Way for Hosni Mubarak’s Release
Generated Headline:  a new york times with a new york
Original Headline:  ‘Here Lies’: A Clue in Hebrew Points to Rome’s Medieval Jew
ish Cemetery
Generated Headline:  a new york times with a new york city
Original Headline:  Green Water Lingers in Olympic Pools as the Excuses Pile Up
Generated Headline:  china to u.s. to make it to be back on
Original Headline:  Google Faces New Round of Antitrust Charges in Europe
Generated Headline:  watch: attorney general is not to be days
Original Headline:  Pelosi: Case Being Made ’In a Very Scientific, Methodical W
ay’ to Impeach Trump
Generated Headline:  a new york times with a new york times
Original Headline:  DELINGPOLE: Facebook Banned Me For Defending Milo
Generated Headline:  in a new york times are isis with isis
Original Headline:  Opinion Transforms Texas’ Abortion Landscape
Generated Headline:  u.s. says u.s. to north korea on u.s. to fight to email em
ail email
Original Headline:  House Challenge to Health Law Could Raise Premiums, Adminis
tration Says
Generated Headline:  trump to wednesday at wall
Original Headline:  California Today: The Tale of the Laguna Beach Jumper
Generated Headline:  new york times at least pain
Original Headline:  Review: ‘Hairspray Live!’ Had Power Voices but Still Lacked
Power
Generated Headline:  u.s. u.s. will be killed in isis
Original Headline:  Belgium Says It Prevented a Terror Attack on Soccer Fans
Generated Headline:  china and a new new york times with the times
Original Headline:  Colin Kaepernick’s Anthem Protest Underlines Union of Sport
```

```
s and Patriotism
Generated Headline:  to donald trump
Original Headline:  Zika, Olympics, U.S. Presidential Race: Your Weekend Briefi
ng
```