In [32]:

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.moun
t("/content/drive/", force_remount=True).
```

In [33]:

```
import nltk
import pandas as pd
```

In [34]:

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[34]:

```
True
```

In [35]:

```
from nltk.tokenize import word_tokenize,sent_tokenize
filepath = '/content/drive/My Drive/English-Telugu/train.en'
corpus = open(filepath, 'r').read()
words = nltk.word_tokenize(corpus)
print("The number of tokens is", len(words))
average_tokens = round(len(words)/75000)
print("The average number of tokens per sentence is",average_tokens)
unique_tokens = set(words)
print("The number of unique tokens are", len(unique_tokens))
```

```
The number of tokens is 1809312
The average number of tokens per sentence is 24
The number of unique tokens are 21095
```

In [36]:

```
from nltk.tokenize import word_tokenize,sent_tokenize
filepath = nltk.data.find('/content/drive/My Drive/English-Telugu/train.te')
corpus = open(filepath, 'r').read()
words = nltk.word_tokenize(corpus)
print("The number of tokens is", len(words))
average_tokens = round(len(words)/75000)
print("The average number of tokens per sentence is",average_tokens)
unique_tokens = set(words)
print("The number of unique tokens are", len(unique_tokens))
```

```
The number of tokens is 1030924
The average number of tokens per sentence is 14
The number of unique tokens are 106016
```

In [37]:

```
f= open("/content/drive/My Drive/English-Telugu/train.en")
en=f.readlines()
len(en)
```

Out[37]:

```
75000
```

In [38]:

```
en[:6]
```

Out[38]:

```
['we just party , and we can do whatever we want .\n',
 'and ziza the son of shiphi , the son of allon , the son of jedaiah , the son of shimri
, the son of shemaiah;\n',
 'now a poor wise man was found in it , and he by his wisdom delivered the city; yet no m
an remembered that same poor man .\n',
 "and the child grew , and she brought him unto pharaoh's daughter , and he became her so
n . and she called his name moses: and she said , because i drew him out of the water .\n
",
 '- i was at the jeweler and nobody came .\n',
 'and the consecrated things were six hundred oxen and three thousand sheep .\n']
```

In [39]:

```
f= open("/content/drive/My Drive/English-Telugu/train.te")
te=f.readlines()
len(te)
```

Out[39]:

```
75000
```

In [40]:

```
import re
def remove_punc(x):
  return re.sub('[!#?,.:";\n]', '', x)
```

In [41]:

```
for i in range(len(te)):
  te[i]=remove_punc(te[i])
```

In [42]:

```
te[:6]
```

Out[42]:

```
['[illegible] ',
 '[illegible] ',
 '[illegible] ',
 '[illegible] ',
 '[illegible] ',
 '[illegible] ']
```

In [43]:

```
for i in range(len(en)):
  en[i]=remove_punc(en[i])
```

In [44]:

```
en[:6]
```

Out[44]:

```
['we just party  and we can do whatever we want ',
 'and ziza the son of shiphi  the son of allon  the son of jedaiah  the son of shimri  th
e son of shemaiah',
 'now a poor wise man was found in it  and he by his wisdom delivered the city yet no man
remembered that same poor man ',
 "and the child grew  and she brought him unto pharaoh's daughter  and he became her son
and she called his name moses and she said  because i drew him out of the water ",
 '- i was at the jeweler and nobody came ',
```

'and the consecrated things were six hundred oxen and three thousand sheep ']

In [45]:

```
words_en = []
for i in en:
  for word in i.split():
    words_en.append(word)

words_en[:10]
```

Out[45]:

```
['we', 'just', 'party', 'and', 'we', 'can', 'do', 'whatever', 'we', 'want']
```

In [46]:

```
words_te = []
for i in te:
  for word in i.split():
    words_te.append(word)

words_te[:10]
```

Out[46]:

```
['■■■■■',
 '■■■■',
 '■■■■■',
 '■■■■',
 '■■■■■■',
 '■■■■■■■■■',
 '■■■■■■■',
 '■■■■■■■',
 '■■■■',
 '■■■■■■■■■']
```

In [47]:

```
from collections import Counter
english_words_counts = Counter(words_en)
telugu_words_counts = Counter(words_te)
```

In [48]:

```
import operator
english_words_counts = sorted(english_words_counts.items(), key = operator.itemgetter(1),
reverse = True)
telugu_words_counts = sorted(telugu_words_counts.items(), key = operator.itemgetter(1), r
everse = True)
```

In [49]:

```
english_words_counts[:10]
```

Out[49]:

```
[('the', 113593),
 ('and', 80265),
 ('of', 62955),
 ('to', 34296),
 ('in', 24579),
 ('that', 21052),
 ('you', 20618),
 ('he', 19632),
 ('i', 19186),
 ('a', 17782)]
```

In [50]:

```
telugu_words_counts[:10]
```

Out[50]:

```
[('■■■■', 12085),
 ('■■■■■■', 9127),
 ('■■', 9006),
 ('■■', 8842),
 ('■', 8681),
 ('■■■■', 8569),
 ('■■■■', 6680),
 ('■■', 5784),
 ('■■■■', 5701),
 ('■■■', 5691)]
```

In [51]:

```
maxlen_english = -1
for doc in en:
    tokens = nltk.word_tokenize(doc)
    if(maxlen_english < len(tokens)):
        maxlen_english = len(tokens)
print("The maximum number of words in any document = ", maxlen_english)
```

```
The maximum number of words in any document =  74
```

In [52]:

```
maxlen_telugu = -1
for doc in te:
    tokens = nltk.word_tokenize(doc)
    if(maxlen_telugu < len(tokens)):
        maxlen_telugu = len(tokens)
print("The maximum number of words in any document = ", maxlen_telugu)
```

```
The maximum number of words in any document =  47
```

In [53]:

```
def tokenize_and_pad(x, maxlen):
  #  a tokenier to tokenize the words and create sequences of tokenized words
  tokenizer = Tokenizer(char_level = False)
  tokenizer.fit_on_texts(x)
  sequences = tokenizer.texts_to_sequences(x)
  padded = pad_sequences(sequences, maxlen = maxlen, padding = 'post')
  return tokenizer, sequences, padded
```

In [54]:

```
from tensorflow.keras.preprocessing.text import one_hot, Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
x_tokenizer, x_sequences, x_padded = tokenize_and_pad(en, maxlen_english)
y_tokenizer, y_sequences, y_padded = tokenize_and_pad(te,  maxlen_telugu)
```

In [55]:

```
[print("The tokenized version for document\n", en[-1:][0],"\n", x_padded[-1:][0])]
```

```
The tokenized version for document
 how could you stand for it
 [149 391   7 316  11  16   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0]
```

Out[55]:

```
[None]
```

In [56]:

```
print("The tokenized version for document\n", te[-1:][0],"\n ", y_padded[-1:][0])
```

```
The tokenized version for document
```

```
[   6  296 8102  887    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0]
```

In [57]:

```
from tensorflow.keras.preprocessing.text import one_hot, Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, TimeDistributed, RepeatVector, Embed
ding, Input, LSTM, Conv1D, MaxPool1D, Bidirectional
from tensorflow.keras.models import Model
```

In [58]:

```
english_vocab_size = len(english_words_counts)
telugu_vocab_size = len(telugu_words_counts)
```

In [59]:

```
# Sequential Model
model = Sequential()
# embedding layer
model.add(Embedding(english_vocab_size, 256, input_length = maxlen_english, mask_zero = T
rue))
# encoder
model.add(LSTM(256, return_sequences= True))
model.add(LSTM(128))
# decoder
# repeatvector repeats the input for the desired number of times to change
# 2D-array to 3D array. For example: (1,256) to (1,23,256)
model.add(RepeatVector(maxlen_telugu))
model.add(LSTM(256, return_sequences= True ))
model.add(LSTM(128, return_sequences= True ))
model.add(TimeDistributed(Dense(telugu_vocab_size, activation ='softmax')))
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accurac
y'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 74, 256)           5810688
_________________________________________________________________
lstm_2 (LSTM)                (None, 256)               525312
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 47, 256)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 47, 256)           525312
_________________________________________________________________
time_distributed_1 (TimeDist (None, 47, 106518)        27375126
=================================================================
Total params: 34,236,438
Trainable params: 34,236,438
Non-trainable params: 0
_________________________________________________________________
```

In [60]:

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x_padded, y_padded, test_size = 0.1)
```

In [61]:

```
import numpy as np
y_train = np.expand_dims(y_train, axis = 2)
```

```
y_train.shape
```

Out[61]:

```
(67500, 47, 1)
```

In [62]:

```
history=model.fit(x_train, y_train, batch_size=128, validation_split= 0.1, epochs=10)
```

```
Epoch 1/10
475/475 [==============================] - 357s 739ms/step - loss: 4.2315 - accuracy: 0.7
335 - val_loss: 2.6125 - val_accuracy: 0.7445
Epoch 2/10
475/475 [==============================] - 349s 735ms/step - loss: 2.5565 - accuracy: 0.7
457 - val_loss: 2.5877 - val_accuracy: 0.7461
Epoch 3/10
475/475 [==============================] - 349s 734ms/step - loss: 2.5164 - accuracy: 0.7
472 - val_loss: 2.5633 - val_accuracy: 0.7466
Epoch 4/10
475/475 [==============================] - 349s 734ms/step - loss: 2.4884 - accuracy: 0.7
464 - val_loss: 2.5288 - val_accuracy: 0.7471
Epoch 5/10
475/475 [==============================] - 348s 734ms/step - loss: 2.4277 - accuracy: 0.7
477 - val_loss: 2.4891 - val_accuracy: 0.7470
Epoch 6/10
475/475 [==============================] - 348s 733ms/step - loss: 2.3801 - accuracy: 0.7
480 - val_loss: 2.4632 - val_accuracy: 0.7470
Epoch 7/10
475/475 [==============================] - 348s 733ms/step - loss: 2.3364 - accuracy: 0.7
488 - val_loss: 2.4319 - val_accuracy: 0.7477
Epoch 8/10
475/475 [==============================] - 348s 734ms/step - loss: 2.2938 - accuracy: 0.7
487 - val_loss: 2.3907 - val_accuracy: 0.7487
Epoch 9/10
475/475 [==============================] - 349s 734ms/step - loss: 2.2224 - accuracy: 0.7
505 - val_loss: 2.3501 - val_accuracy: 0.7492
Epoch 10/10
475/475 [==============================] - 349s 734ms/step - loss: 2.1502 - accuracy: 0.7
522 - val_loss: 2.3210 - val_accuracy: 0.7496
```

In [63]:

```
y_test = np.expand_dims(y_test, axis = 2)
y_test.shape
```

Out[63]:

```
(7500, 47, 1)
```

In [64]:

```
model.evaluate(x_test, y_test, batch_size=32)
```

```
235/235 [==============================] - 61s 260ms/step - loss: 2.3084 - accuracy: 0.75
07
```

Out[64]:

```
[2.308351516723633, 0.7507376074790955]
```

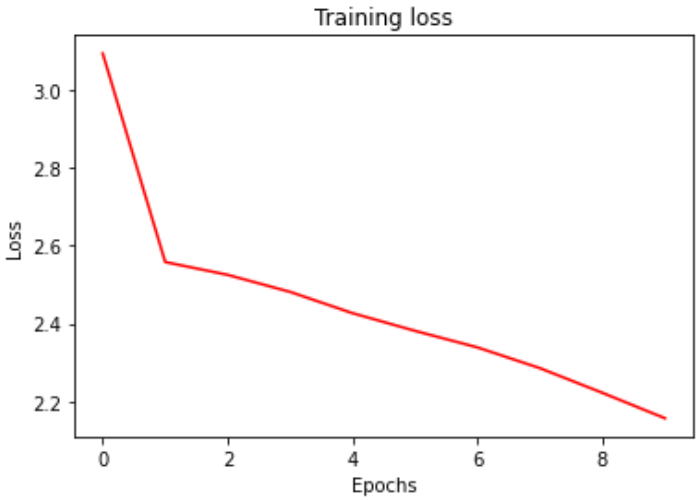
In [65]:

```
model.save("NMT2.h5")
```

In [67]:

```
import matplotlib.pyplot as plt
loss=history.history['loss']
acc=history.history['accuracy']
epochs=range(len(loss))
plt.plot(epochs, loss, 'r')
plt.title('Training loss')
```

```
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()
plt.plot(epochs, acc, 'b')
plt.title('Training loss')
plt.xlabel("Epochs")
plt.ylabel("accuracy")
plt.legend(["accuracy"])
plt.legend(["accuracy"])
plt.show()
```



In [ ]: