Downloading the Zip file

In [1]:
```
!wget http://www.manythings.org/anki/fra-eng.zip
```

```
--2021-01-07 10:23:06--  http://www.manythings.org/anki/fra-eng.zip
Resolving www.manythings.org (www.manythings.org)... 104.24.108.196, 172.67.173.198, 10
4.24.109.196, ...
Connecting to www.manythings.org (www.manythings.org)|104.24.108.196|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6129887 (5.8M) [application/zip]
Saving to: ‘fra-eng.zip’

fra-eng.zip         100%[===================>]   5.85M  9.29MB/s    in 0.6s

2021-01-07 10:23:07 (9.29 MB/s) - ‘fra-eng.zip’ saved [6129887/6129887]
```

Extracting the Zip file

In [2]:
```
import zipfile
zip = zipfile.ZipFile('fra-eng.zip')
zip.extractall()
```

Dependencies

In [3]:
```
import string,re
from unicodedata import normalize
from numpy import array,argmax
from pickle import load,dump
from numpy.random import rand,shuffle
```

In [4]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.utils.vis_utils import plot_model
from keras.models import Sequential,load_model
from keras.layers import LSTM,Dense,Embedding,RepeatVector,TimeDistributed
from nltk.translate.bleu_score import SmoothingFunction,corpus_bleu
smoothie = SmoothingFunction().method4
```

Loading the file and reading the content of the file

In [5]:
```
# load file into memory
def load_file(filename):
        # open the file as read only
        file = open(filename, mode='rt', encoding='utf-8')
        # read all text
        text = file.read()
        # close the file
        file.close()
        return text
```

Splitting the sentence into pairs

In [6]:
```
# split a loaded document into sentences
def splitting_sentence(doc):
        sentences = doc.strip().split('\n')
```

```
        pairs = [sentence.split('\t') for sentence in  sentences]
        return pairs
```

Cleaning the pairs

In [7]:

```
# cleaning a list of sentences and creating pairs

def clean_pairs(sentences):
        cleaned = list()

        # preparing regex for char filtering
        re_print = re.compile('[^%s]' % re.escape(string.printable))

        # preparing translation table for removing punctuation
        table = str.maketrans('', '', string.punctuation)

  # iterating over each pair
        for pair in sentences:
                clean_pair = list()

                for sentence in pair:
                        # normalizing unicode characters
                        sentence = normalize('NFD', sentence).encode('ascii', 'ignore')
                        sentence = sentence.decode('UTF-8')
                        # tokenizing on white space
                        sentence = sentence.split()
                        # converting to lowercase
                        sentence = [word.lower() for word in sentence]
                        # removing punctuation from each token
                        sentence = [word.translate(table) for word in sentence]
                        # removing non-printable chars form each token
                        sentence = [re_print.sub('', w) for w in sentence]
                        # removing tokens with numbers in them
                        sentence = [word for word in sentence if word.isalpha()]
                        # storing as string
                        clean_pair.append(' '.join(sentence))
                cleaned.append(clean_pair)
        return array(cleaned)
```

Saving the Cleaned data

In [8]:

```
def saving_clean_data(sentences, filename):
        dump(sentences, open(filename, 'wb'))
        print(filename,': Saved')
```

Saving data in .pkl format

In [9]:

```
# load dataset

filename = 'fra.txt'
doc = load_file(filename)

# split into english-french pairs
pairs = splitting_sentence(doc)

# clean sentences
clean_pairs = clean_pairs(pairs)

# save clean pairs to file
saving_clean_data(clean_pairs, 'english-french.pkl')
```

```
print('English','-->',"French")
# spot check
for i in range(25):
        print(clean_pairs[i,0],'-->',clean_pairs[i,1])
```

```
english-french.pkl : Saved
English --> French
go --> va
hi --> salut
hi --> salut
run --> cours
run --> courez
who --> qui
wow --> ca alors
fire --> au feu
help --> a laide
jump --> saute
stop --> ca suffit
stop --> stop
stop --> arretetoi
wait --> attends
wait --> attendez
go on --> poursuis
go on --> continuez
go on --> poursuivez
hello --> bonjour
hello --> salut
i see --> je comprends
i try --> jessaye
i won --> jai gagne
i won --> je lai emporte
i won --> jai gagne
```

Loading the cleaned data

In [10]:

```
# load a clean dataset
def loading_cleaned_data(filename):
        return load(open(filename, 'rb'))
```

In [11]:

```
# load dataset
data = loading_cleaned_data('english-french.pkl')
print(data.shape)
```

```
(179904, 3)
```

Scaling of data

Size

1.Dataset - 20000

2.Training - 18000

3.Testing - 2000

In [12]:

```
# reducing dataset size (scaling)

new_data_size = 20000
dataset = data[:new_data_size, :]

# randomly shuffling the dataset to get proper training and testing data
```

```
shuffle(dataset)

# splitting into training and testing (90%-10%)
train, test = dataset[:18000], dataset[18000:]

# saving the cleaned data,train data and test data
saving_clean_data(dataset, 'english-french-both.pkl')
saving_clean_data(train, 'english-french-train.pkl')
saving_clean_data(test, 'english-french-test.pkl')
```

```
english-french-both.pkl : Saved
english-french-train.pkl : Saved
english-french-test.pkl : Saved
```

In [13]:

```
# loading datasets and saving it into variables
dataset = loading_cleaned_data('english-french-both.pkl')
train = loading_cleaned_data('english-french-train.pkl')
test = loading_cleaned_data('english-french-test.pkl')
```

Creating a tokenizer for the lines and finding the maximum length phrase

In [14]:

```
# fit a tokenizer
def create_tokenizer(lines):
        tokenizer = Tokenizer()
        tokenizer.fit_on_texts(lines)
        return tokenizer

# max sentence length
def max_length(lines):
        return max(len(line.split()) for line in lines)
```

Size of English & French vocabulary and their max phrase length

In [15]:

```
# preparing the english tokenizer

eng_tokenizer = create_tokenizer(dataset[:, 0])
eng_vocab_size = len(eng_tokenizer.word_index) + 1
eng_length = max_length(dataset[:, 0])

print('English Vocabulary Size: %d' % eng_vocab_size)
print('English Max Length: %d' % (eng_length))

# preparing the french tokenizer

fra_tokenizer = create_tokenizer(dataset[:, 1])
fra_vocab_size = len(fra_tokenizer.word_index) + 1
fra_length = max_length(dataset[:, 1])
print('French Vocabulary Size: %d' % fra_vocab_size)
print('French Max Length: %d' % (fra_length))
```

```
English Vocabulary Size: 3460
English Max Length: 5
French Vocabulary Size: 6922
French Max Length: 11
```

Encoding to integers and padding to the maximum phrase length

In [16]:

```
# Input and Output sequence must be encoded to integers and padded to the maximum phrase
def encode_sequences(tokenizer, length, lines):
        # integer encode sequences
        x = tokenizer.texts_to_sequences(lines)
```

```
        # pad sequences with 0 values
        x = pad_sequences(x, maxlen=length, padding='post')
        return x

# One hot encoding to max phrase length
def one_hot_encoding(sequences, vocab_size):
        y_1 = list()
        for sequence in sequences:
                encoded = to_categorical(sequence, num_classes=vocab_size)
                y_1.append(encoded)
        y = array(y_1)
        y = y.reshape(sequences.shape[0], sequences.shape[1], vocab_size)
        return y
```

Training and Testing Data

In [17]:

```
# preparing training data
trainX = encode_sequences(fra_tokenizer, fra_length, train[:, 1])
trainY = encode_sequences(eng_tokenizer, eng_length, train[:, 0])
trainY = one_hot_encoding(trainY, eng_vocab_size)

# prepare testing data
testX = encode_sequences(fra_tokenizer, fra_length, test[:, 1])
testY = encode_sequences(eng_tokenizer,eng_length, test[:, 0])
testY = one_hot_encoding(testY, eng_vocab_size)
```

In [18]:

```
print('training size:',trainX.shape,trainY.shape)
print('testing size:',testX.shape,testY.shape)
```

```
training size: (18000, 11) (18000, 5, 3460)
testing size: (2000, 11) (2000, 5, 3460)
```

Building the model

In [19]:

```
def model_building(source_vocab, target_vocab, source_len, target_len, units):
        model = Sequential()
        model.add(Embedding(source_vocab, units, input_length=source_len, mask_zero=True
        model.add(LSTM(units))
        model.add(RepeatVector(target_len))
        model.add(LSTM(units, return_sequences=True))
        model.add(TimeDistributed(Dense(target_vocab, activation='softmax')))
        return model
```

Defining and Compiling the model

In [20]:

```
model = model_building(fra_vocab_size, eng_vocab_size, fra_length, eng_length, 512)
model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['acc'])
```

Model Summary

In [21]:

```
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 11, 512)           3544064
_________________________________________________________________
lstm (LSTM)                  (None, 512)               2099200
```

```
repeat_vector (RepeatVector) (None, 5, 512)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 5, 512)           2099200
_________________________________________________________________
time_distributed (TimeDistri (None, 5, 3460)          1774980
=================================================================
Total params: 9,517,444
Trainable params: 9,517,444
Non-trainable params: 0
_________________________________________________________________
None
```

In [22]:

```
# Stop model if accuracy of the model doesn't changes by more than 0.01
# Patience = 5 : After each 5 epochs if no improvement is there then training will be st
from tensorflow.keras.callbacks import EarlyStopping
es = EarlyStopping(monitor='val_acc',patience= 5,min_delta=0.01)
```

Fitting the model

1.Epochs = 50

2.Batch_size = 25

In [23]:

```
# fit model
model.fit(trainX, trainY, epochs= 50, batch_size=25, validation_data=(testX, testY), ver
```

```
Epoch 1/50
720/720 - 35s - loss: 3.7328 - acc: 0.4385 - val_loss: 3.1504 - val_acc: 0.4987
Epoch 2/50
720/720 - 24s - loss: 2.7811 - acc: 0.5426 - val_loss: 2.5641 - val_acc: 0.5814
Epoch 3/50
720/720 - 24s - loss: 2.1818 - acc: 0.6056 - val_loss: 2.2248 - val_acc: 0.6138
Epoch 4/50
720/720 - 25s - loss: 1.7135 - acc: 0.6572 - val_loss: 2.0067 - val_acc: 0.6435
Epoch 5/50
720/720 - 24s - loss: 1.3279 - acc: 0.7101 - val_loss: 1.8262 - val_acc: 0.6674
Epoch 6/50
720/720 - 24s - loss: 0.9954 - acc: 0.7662 - val_loss: 1.7155 - val_acc: 0.6913
Epoch 7/50
720/720 - 25s - loss: 0.7395 - acc: 0.8162 - val_loss: 1.6456 - val_acc: 0.7129
Epoch 8/50
720/720 - 25s - loss: 0.5407 - acc: 0.8595 - val_loss: 1.6244 - val_acc: 0.7219
Epoch 9/50
720/720 - 25s - loss: 0.4048 - acc: 0.8923 - val_loss: 1.6326 - val_acc: 0.7295
Epoch 10/50
720/720 - 25s - loss: 0.3119 - acc: 0.9159 - val_loss: 1.6253 - val_acc: 0.7328
Epoch 11/50
720/720 - 25s - loss: 0.2497 - acc: 0.9304 - val_loss: 1.6645 - val_acc: 0.7293
Epoch 12/50
720/720 - 25s - loss: 0.2136 - acc: 0.9379 - val_loss: 1.6664 - val_acc: 0.7370
Epoch 13/50
720/720 - 25s - loss: 0.1917 - acc: 0.9434 - val_loss: 1.6865 - val_acc: 0.7379
Epoch 14/50
720/720 - 24s - loss: 0.1760 - acc: 0.9457 - val_loss: 1.7025 - val_acc: 0.7387
```

Out[23]:

```
<tensorflow.python.keras.callbacks.History at 0x7fd5a41055f8>
```

Evaluating model and calculating BLEU Score

Evaluation involves two steps:

1.Generating a translated output sequence, and

2.then repeating this process for many input examples and summarizing the skill of the model across multiple cases.

In [24]:
```
# mapping integer to a word
def word_for_id(integer, tokenizer):
        for word, index in tokenizer.word_index.items():
                if index == integer:
                        return word
        return None
```

In [25]:
```
# generating target given source sequence
def predict_sequence(model, tokenizer, source):
        prediction = model.predict(source, verbose=0)[0]
        integers = [argmax(vector) for vector in prediction]
        target = list()
        for i in integers:
                word = word_for_id(i, tokenizer)
                if word is None:
                        break
                target.append(word)
        return ' '.join(target)
```

In [26]:
```
# evaluating the skill of the model
def evaluate_model(model, tokenizer, sources, raw_dataset):

  # Creating empty lists for actual phrases(French) and predicted phrases(English)
  actual,predicted = list(),list()
  a,b,c = list(),list(),list()
  for i,source in enumerate(sources):

    # reshaping to the required size
    source = source.reshape((1, source.shape[0]))

    # predicting for the english tokenizer
    translation = predict_sequence(model, eng_tokenizer, source)
    # raw_dataset = raw_dataset[i].split(' ')
    # print(raw_dataset[i][1])

    raw_src,raw_target = raw_dataset[i][1],raw_dataset[i][0]

    # First 10 Predictions
    if i <= 10:
      print('source = ',raw_src,'<--->', ' target = ',raw_target,'<--->','  predicted =

    actual.append([raw_target.split()])
    predicted.append(translation.split())

  # calculating BLEU score
  print('-------------------------------------------')
  print('BLEU Score :')
  print('BLEU score-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0),smoot
  print('BLEU score-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0),smo
  print('BLEU score-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0),s
  print('BLEU score-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, (
```

Evaluating Model on training data

In [27]:
```
evaluate_model(model,eng_tokenizer,trainX,train)
```

```
source =  je suis divorce <--->  target =  i am divorced <--->   predicted =  im divorce
d
source =  la vie nest pas facile <--->  target =  life aint easy <--->   predicted =  li
fes not easy
source =  tom est finalement parti <--->  target =  tom finally left <--->   predicted =
tom finally left
source =  tom ecrit bien <--->  target =  tom writes well <--->   predicted =  tom write
s well
source =  je ne laime pas <--->  target =  i dont like him <--->   predicted =  i dont l
ike her
source =  attache tes chaussures <--->  target =  tie your shoes <--->   predicted =  ti
e your shoes
source =  appelons tom <--->  target =  lets call tom <--->   predicted =  lets call tom
source =  fermez la boite <--->  target =  close the box <--->   predicted =  close the
box
source =  cest enorme <--->  target =  this is huge <--->   predicted =  this is huge
source =  donneznous une seconde <--->  target =  give us a second <--->   predicted =
give us a second
source =  jetais horrifiee <--->  target =  i was horrified <--->   predicted =  i was h
orrified
--------------------------------------------
BLEU Score :
BLEU score-1: 0.948733
BLEU score-2: 0.930221
BLEU score-3: 0.881637
BLEU score-4: 0.661279
```

Evaluating Model on testing data

In [28]:
```
evaluate_model(model, eng_tokenizer, testX, test)
```

```
source =  ce sont des melons <--->  target =  they are melons <--->   predicted =  theyr
e garbage
source =  les jumeaux sourirent <--->  target =  the twins smiled <--->   predicted =  t
he is fell
source =  elle tirait la langue <--->  target =  she was panting <--->   predicted =  sh
e stood him
source =  je me suis trompe sur tom <--->  target =  i misjudged tom <--->   predicted =
i got at tom
source =  que ressenstu <--->  target =  how does it feel <--->   predicted =  what do y
ou
source =  cest un soulagement <--->  target =  its a relief <--->   predicted =  thats a
relief
source =  vous pouvez me lacher <--->  target =  can you skip me <--->   predicted =  ca
n you skip me
source =  ce sont des amateurs <--->  target =  theyre amateurs <--->   predicted =  the
yre garbage
source =  je suis trop petit <--->  target =  i am too short <--->   predicted =  im too
too
source =  ils ont termine <--->  target =  theyre done <--->   predicted =  theyre done
source =  nous sommes engagees <--->  target =  were committed <--->   predicted =  were
canadians
--------------------------------------------
BLEU Score :
BLEU score-1: 0.645399
BLEU score-2: 0.548542
BLEU score-3: 0.488032
BLEU score-4: 0.303227
```