Neural Machine Translation from Hindi to English

Assignment is to build a Neural Machine Translation (NMT) model to translate Hindi Sentences into machine English.

We will do this using by creating attention model as in Neural Machine Translation by Jointly Learning to Align and Translate: Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio https://arxiv.org/pdf/1409.0473.pdf (https://arxiv.org/pdf/1409.0473.pdf)

We will be using following small parallel corpus "http://www.manythings.org/anki/hin-eng.zip (http://www.manythings.org/anki/hin-eng.zip)"

Notes: In Appendix section at end we have given additional helper functions which can be used to improve model as future improvement effort.

Loading Libraries.

In [1]:
```
from keras.layers import Bidirectional, Concatenate, Permute, Dot, Input, LSTM, Multiply
from keras.layers import RepeatVector, Dense, Activation, Lambda
from keras.optimizers import Adam
from keras.utils import to_categorical
from keras.models import load_model, Model
import keras.backend as K
import numpy as np

import random

#nmt utils has functions which will be used for Softmax or Data Procesing.
from nmt_utils import *
import matplotlib.pyplot as plt
%matplotlib inline
```

```
Using TensorFlow backend.
```

1 - Translating human readable dates into machine readable dates

The model we will build here could be used to translate from from English to Hindi or any other parallel corpus.

1.1 - Dataset

In this section we are going to download the dataset and prepare the dataset with Padding & Integer Encoding & One hot encoding.

In [2]:
```
import requests, zipfile, io,os
r = requests.get("http://www.manythings.org/anki/hin-eng.zip")
z = zipfile.ZipFile(io.BytesIO(r.content))
z.extractall()
#Verifying if the file hin.txt are downloaded properly.
if os.path.isfile("hin.txt"):
    print('hin.txt exists')
```

```
hin.txt exists
```

In [3]:
```
#Reading the file
file=open("hin.txt",'r',encoding='utf-8')
content=file.read()
file.close()
```

In [4]:
```
import string
import re
from pickle import dump
from unicodedata import normalize
from numpy import array

# load doc into memory
def load_doc(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

# split a loaded document into sentences
def to_pairs(doc):
    lines = doc.strip().split('\n')
    pairs = [line.split('\t') for line in  lines]
    return pairs

# clean a list of lines
def clean_pairs(lines):
    cleaned = list()
    # prepare regex for char filtering
    re_punc = re.compile('[ |%s]' % re.escape(string.punctuation))
    re_print = re.compile('[^%s]' % re.escape(string.printable))
    for pair in lines:
        clean_pair = list()
        for line in pair:
            # tokenize on white space
            line = line.split()
            # remove punctuation from each token
            line = [re_punc.sub('', w) for w in line]
            # remove tokens with numbers in them
            #line = [word for word in line if word.isalpha()]
            #line=re.sub('[ /]', '', line)
            # store as string
            clean_pair.append(' '.join(line))
        cleaned.append(clean_pair)
    return array(cleaned)
```

```
# save a list of clean sentences to file
def save_clean_data(sentences, filename):
    dump(sentences, open(filename, 'wb'))
    print('Saved: %s' % filename)

# load dataset
filename = 'hin.txt'
doc = load_doc(filename)
# split into english-hindi pairs
pairs = to_pairs(doc)
# clean sentences
clean_pairs = clean_pairs(pairs)
# save clean pairs to file
print ("Number of clean pairs",clean_pairs.shape[0])
save_clean_data(clean_pairs, 'english-hindi.pkl')
# spot check
for i in range(10):
    print('[%s] => [%s]' % (clean_pairs[i,0], clean_pairs[i,1]))
```

```
Number of clean pairs 2867
Saved: english-hindi.pkl
[Help] => [बचाओ]
[Jump] => [उछलो]
[Jump] => [कूदो]
[Jump] => [छलांग]
[Hello] => [नमस्ते]
[Hello] => [नमस्कार]
[Cheers] => [वाहवाह]
[Cheers] => [चियर्स]
[Got it] => [समझे कि नहीं]
[Im OK] => [मैं ठीक हूँ]
```

In [5]:
```
from pickle import load
from pickle import dump
from numpy.random import shuffle

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

# save a list of clean sentences to file
def save_clean_data(sentences, filename):
    dump(sentences, open(filename, 'wb'))
    print('Saved: %s' % filename)

# load dataset
raw_dataset = load_clean_sentences('english-hindi.pkl')

# reduce dataset size
n_sentences = raw_dataset.shape[0]
print (n_sentences)
dataset = raw_dataset[:n_sentences, :]
# random shuffle
shuffle(dataset)
# split into train/test
train, test = dataset[:2800], dataset[2800:]
# save
save_clean_data(dataset, 'english-hindi-both.pkl')
```

```
2867
Saved: english-hindi-both.pkl
```

In [6]:
```
#Check the data sample
dataset.shape
```

Out[6]: (2867, 2)

In [7]:
```
#Converting it to tuples.
dataset_list=(list(tuple(map(tuple, dataset))))
```

In [8]:
```
#English Sentence List
english_sentences_list=list(dataset[:,0])
```

In [9]:
```
english_sentences_list[0]='Please make yourself at home'
english_sentences_list[0]
```

Out[9]: 'Please make yourself at home'

In [10]:
```
#English Sentence Unique Word List and Length of Vocabulary
english_unique_words=set((' '.join(english_sentences_list)).split())
english_vocab_len=len(set((' '.join(english_sentences_list)).split()))
```

In [11]:
```
#Hindi Sentence List
hindi_sentences_list=list(dataset[:,1])
hindi_sentences_list[0]='इसको अपना घर ही समझो'
```

In [12]:
```
#Hindi Sentence Unique Word List and Length of Vocabulary
hindi_unique_words=set((' '.join(hindi_sentences_list)).split())
hindi_vocab_len=len(set((' '.join(hindi_sentences_list)).split()))
```

In [13]:
```
#Creating Dictionary with Unknown and Pad elements
english_dictionary=dict(zip(sorted(english_unique_words) + ['<unk>', '<pad>'], list(range(len(english_unique
hindi_dictionary=dict(zip(sorted(hindi_unique_words) + ['<unk>', '<pad>'], list(range(len(hindi_unique_words
```

In [14]:
```
#Reverse Dictionary for both languages
revere_dictionary_hindi=dict((v,k) for k,v in hindi_dictionary.items())
revere_dictionary_english=dict((v,k) for k,v in english_dictionary.items())
```

In [15]:
```
#Storing the index of padding value in variables to add it going ahead.
english_padding_value=english_dictionary['<pad>']
hindi_padding_value=hindi_dictionary['<pad>']
```

In [16]:
```
#This going to be the global variable with maximum number of words found in a sentence
max_english_words=max(len(line.split()) for line in english_sentences_list)
max_hindi_words=max(len(line.split()) for line in hindi_sentences_list)
print(max_english_words,max_hindi_words)
```

```
22 25
```

In [17]:
```
def get_padded_encoding(sentences_list,language_dictionary,max_language_words):
    padding_value=language_dictionary['<pad>']
    language_array=[]
    #Iterate over List.
    for sentence in sentences_list:
        #Replaces English words with English Vocabulary Indexes and Hindi with Hindi Vocabulary Indexes.
        #logic: if a word not in dictionary enters, it will be replaced by unk key value.
        single_sentence_array=[]
        for word in sentence.split():
            try:
                #single_sentence_array=([language_dictionary[word] for word in sentence.split()])
                single_sentence_array.append(language_dictionary[word])
            except KeyError:
                unk='<unk>'
                single_sentence_array.append(language_dictionary[unk])
        #Find the length of english_single_sentence_array
        length_single_sentence=(len(single_sentence_array))
        #So how many times padding dictionary key needs to be appended, if we say maximum length of sentence
        if (max_language_words>length_single_sentence):
            padding_count=(max_language_words-length_single_sentence)
        else:
            padding_count=0
        if (padding_count>0):
            for pad in range(0,padding_count):
                single_sentence_array.append(padding_value)
        else:
            single_sentence_array=single_sentence_array[0:max_language_words]
        #Append to main array
        language_array.append(single_sentence_array)
    #Convert to Numpy array at the end
    language_array=np.array(language_array)
    return(language_array)
```

Instead of doing a padding over large sentence size, emperically it is found that it is better to do for a short sentences considering the limitation we are having with respect to corpus size.

In [18]:
```
sentence_length=6
```

In [19]:
```
#Get encoded sentences
hindi_encoding=get_padded_encoding(hindi_sentences_list,hindi_dictionary,sentence_length)
english_encoding=get_padded_encoding(english_sentences_list,english_dictionary,sentence_length)
print(hindi_encoding.shape,english_encoding.shape)
```

```
(2867, 6) (2867, 6)
```

In [20]:
```
#Verifying the encoding and decoding for a sample data.
print(english_sentences_list[1],hindi_sentences_list[1])
print(english_encoding[1],hindi_encoding[1])
#Check if encoding gives back the same answer
for key in english_encoding[1]:
    print(revere_dictionary_english[key])
for key in hindi_encoding[1]:
    print(revere_dictionary_hindi[key])
english_dictionary['<pad>']
hindi_dictionary['<pad>']
```

```
I am tired of my work मैं अपने काम से थक चुका हूँ
[ 185  487 2377 1711 1646 2580] [2189   61  468 2707 1221  852]
I
am
tired
of
my
work
मैं
अपने
काम
से
थक
चुका
```

Out[20]:
```
2872
```

In [21]:
```
#We will convert the english and hindi encodings to one hot encodings.
#Please note Input is of the dimension (number of sentences,max_length_language(every column is a word))
#Output is (number of sentences,Max_length_language(every row is a word),length of vocabulary)
#Basically every row of the onehotcode matrix must be for one word.
#How=1 => 1 0 0
#Are=2 => 0 1 0
#You=3 => 0 0 1
#We are trying to translate hindi to english, so our X is Hindi and Y is English
X=hindi_encoding
Y=english_encoding
#Note: Instead of one hot we can use word embeddings for Xoh
Xoh=np.array(list(map(lambda x: to_categorical(x, num_classes=len(hindi_dictionary)), X)))
Yoh=np.array(list(map(lambda x: to_categorical(x, num_classes=len(english_dictionary)), Y)))
print("X.shape:", X.shape)
print("Y.shape:", Y.shape)
print("Xoh.shape:", Xoh.shape)
print("Yoh.shape:", Yoh.shape)
```

```
X.shape: (2867, 6)
Y.shape: (2867, 6)
Xoh.shape: (2867, 6, 2873)
Yoh.shape: (2867, 6, 2620)
```

In [22]:
```
Tx = hindi_encoding.shape[1]
Ty = english_encoding.shape[1]
Tx,Ty
```

Out[22]:
```
(6, 6)
```

You we have:

- X: a processed version of the hindi in the data set, where each character is replaced by an index mapped to the character via `hindi_vocab`. Each date is further padded to T_x values with a special character (< pad >). `X.shape = (m, Tx)`
- Y: a processed version of the english sentences in the data set, where each character is replaced by the index it is mapped to in `english_vocab`. You should have `Y.shape = (m, Ty)`.
- Xoh: one-hot version of `X`, the "1" entry's index is mapped to the character thanks to `hindi_vocab`. `Xoh.shape = (m, Tx, len(hindi_vocab))`
- Yoh: one-hot version of `Y`, the "1" entry's index is mapped to the character thanks to `machine_vocab`. `Yoh.shape = (m, Tx, len(english_vocab))`.

Lets also look at some examples of preprocessed training examples.

In [23]:
```
index = 0
#The dataset is english -> Hindi
#Our target is to generate English given Hindi
print("Source:", dataset_list[index][1])
print("Target:", dataset_list[index][0])
print()
print("Source after preprocessing (indices):", X[index])
print("Target after preprocessing (indices):", Y[index])
print()
print("Source after preprocessing (one-hot):", Xoh[index])
print("Target after preprocessing (one-hot):", Yoh[index])
```

```
Source: वह हमेशा मन लगाकर काम करती है
Target: She always works hard

Source after preprocessing (indices): [ 218   59  752 2819 2579 2872]
Target after preprocessing (indices): [ 273 1565 2615  550 1345 2619]

Source after preprocessing (one-hot): [[ 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.  1.]]
Target after preprocessing (one-hot): [[ 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.  1.]]
```

2 - Neural machine translation with attention

The attention mechanism tells a Neural Machine Translation model where it should pay attention to at any step.

2.1 - Attention mechanism

In this part, we will implement the attention mechanism. The diagram on the left shows the attention model. The diagram on the right shows what one "Attention" step does to calculate the attention variables $\alpha^{\langle t,t' \rangle}$, which are used to compute the context variable $context^{\langle t \rangle}$ for each timestep in the output ($t = 1, \ldots, T_y$). One change that we have in our implementation compared to diagram below here is the Post Attention LSTM will be feeding the previous predicted output also by utilizing return_sequences=True feature of Keras.

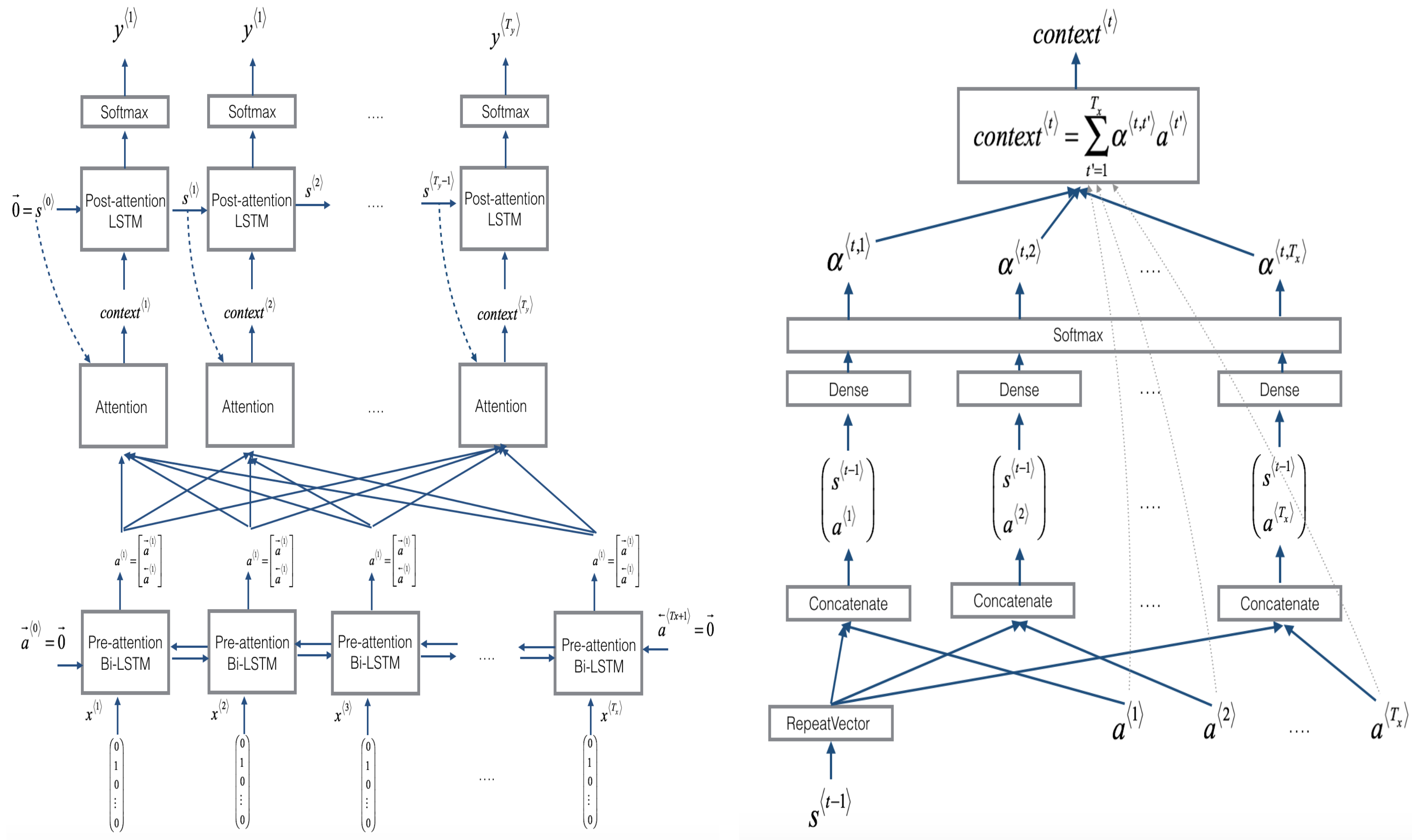

Figure 1: Neural machine translation with attention

Here is the summary of model:

- There are two separate LSTMs in this model (see diagram on the left). Because the one at the bottom of the picture is a Bi-directional LSTM and comes *before* the attention mechanism, we will call it *pre-attention* Bi-LSTM. The LSTM at the top of the diagram comes *after* the attention mechanism, so we will call it the *post-attention* LSTM. The pre-attention Bi-LSTM goes through T_x time steps; the post-attention LSTM goes through T_y time steps.
- The post-attention LSTM passes $s^{\langle t \rangle}, c^{\langle t \rangle}$ from one time step to the next. We are using an LSTM here, the LSTM has both the output activation $s^{\langle t \rangle}$ and the hidden cell state $c^{\langle t \rangle}$.
- We use $a^{\langle t \rangle} = [\overrightarrow{a}^{\langle t \rangle}; \overleftarrow{a}^{\langle t \rangle}]$ to represent the concatenation of the activations of both the forward-direction and backward-directions of the pre-attention Bi-LSTM.
- The diagram on the right uses a `RepeatVector` node to copy $s^{\langle t-1 \rangle}$'s value T_x times, and then `Concatenation` to concatenate $s^{\langle t-1 \rangle}$ and $a^{\langle t \rangle}$ to compute $e^{\langle t, t'}$, which is then passed through a softmax to compute $\alpha^{\langle t, t' \rangle}$. We'll explain how to use `RepeatVector` and `Concatenation` in Keras below.

Implementation detail of the model.

We will start by implementing two functions: `one_step_attention()` and `model()`.

1) `one_step_attention()`: At step t, given all the hidden states of the Bi-LSTM ($[a^{<1>},a^{<2>}, ..., a^{<T_x>}]$) and the previous hidden state of the second LSTM ($s^{<t-1>}$), `one_step_attention()` will compute the attention weights ($[\alpha^{<t,1>},\alpha^{<t,2>}, ..., \alpha^{<t,T_x>}]$) and output the context vector (see Figure 1 (right) for details):

$$context^{<t>} = \sum_{t' = 0}^{T_x} \alpha^{<t,t'>}a^{<t'>}\tag{1}$$

Note that we are denoting the attention in this notebook $context^{\langle t \rangle}$. In the lecture videos, the context was denoted $c^{\langle t \rangle}$, but here we are calling it $context^{\langle t \rangle}$ to avoid confusion with the (post-attention) LSTM's internal memory cell variable, which is sometimes also denoted $c^{\langle t \rangle}$.

2) `model()`: Implements the entire model. It first runs the input through a Bi-LSTM to get back $[a^{<1>},a^{<2>}, ..., a^{<T_x>}]$. Then, it calls `one_step_attention()` T_y times (`for` loop). At each iteration of this loop, it gives the computed context vector $c^{<t>}$ to the second LSTM, and runs the output of the LSTM through a dense layer with softmax activation to generate a prediction $\hat{y}^{<t>}$.

Implementation of `one_step_attention()`. The function `model()` will call the layers in `one_step_attention()` T_y using a for-loop, and it is important that all T_y copies have the same weights. I.e., it should not re-initiaiize the weights every time. In other words, all T_y steps should have shared weights. Here's how we implement layers with shareable weights in Keras:

1. Define the layer objects (as global variables for examples).
2. Call these objects when propagating the input.

We have defined the layers we need as global variables. Please run the following cells to create them.

In [24]:
```
# Defined shared layers as global variables
repeator = RepeatVector(Tx)
concatenator = Concatenate(axis=-1)
densor = Dense(1,activation = "relu")
activator = Activation(softmax, name='attention_weights')
dotor = Dot(axes = 1)
```

Now you can use these layers to implement `one_step_attention()`.

In [25]:
```
def one_step_attention(a, s_prev):
    """
    Performs one step of attention: Outputs a context vector computed as a dot product of the attention weig
    "alphas" and the hidden states "a" of the Bi-LSTM.

    Arguments:
    a -- hidden state output of the Bi-LSTM, numpy-array of shape (m, Tx, 2*n_a)
    s_prev -- previous hidden state of the (post-attention) LSTM, numpy-array of shape (m, n_s)

    Returns:
    context -- context vector, input of the next (post-attetion) LSTM cell
    """

    # Use repeator to repeat s_prev to be of shape (m, Tx, n_s) so that you can concatenate it with all hidd
    print ("s_prev.shape before repeator",s_prev.shape)
    s_prev = repeator(s_prev)
    print ("s_prev.shape after repeator",s_prev.shape)
    print ("a.shape",a.shape)
    # Use concatenator to concatenate a and s_prev on the last axis
    concat = concatenator([a, s_prev])
    print ("concat.shape",concat.shape)
    # Use densor to propagate concat through a small fully-connected neural network to compute the "energies
    e = densor(concat)
    print ("e.shape",e.shape)
    # Use activator and e to compute the attention weights "alphas"
    alphas = activator(e)
    print ("alphas.shape",alphas.shape)
    # Use dotor together with "alphas" and "a" to compute the context vector to be given to the next (post-a
    context = dotor([alphas, a])
    print ("context.shape",context.shape)

    return context
```

We have defined global layers that will share weights to be used in `model()`.

In [26]:
```
#n_a and n_s are the LSTM internal states. Can be selected arbitarily.
n_a = 500
n_s = 500
#We have added dropout to avoid overfiting
post_activation_LSTM_cell = (LSTM(n_s, activation='relu',return_sequences=True,return_state = True,dropout=0
output_layer = Dense(len(english_dictionary), activation=softmax)
```

Now you can use these layers T_y times in a `for` loop to generate the outputs, and their parameters will not be reinitialized. We will have to carry out the following steps:

1. Propagate the input into a Bidirectional LSTM
2. Iterate for $t = 0, \dots, T_y - 1$:
   A. Call `one_step_attention()` on $[\alpha^{<t,1>}, \alpha^{<t,2>}, \dots, \alpha^{<t,T_x>}]$ and $s^{<t-1>}$ to get the context vector $context^{<t>}$.
   B. Give $context^{<t>}$ to the post-attention LSTM cell. We will pass in the previous hidden-state $s^{\langle t-1\rangle}$ and cell-states $c^{\langle t-1\rangle}$ of this LSTM using `initial_state= [previous hidden state, previous cell state]`. To predict the next LSTM cell, output are fed again by using return_sequences=True. Get back the new hidden state $s^{<t>}$ and the new cell state $c^{<t>}$.
   C. Apply a softmax layer to $s^{<t>}$, get the output.
   D. Save the output by adding it to the list of outputs.
3. Create Keras model instance, it should have three inputs ("inputs", $s^{<0>}$ and $c^{<0>}$) and output the list of "outputs".

In [27]:
```
def model(Tx, Ty, n_a, n_s, source_dictionary_size, target_dictionary_size):
    """
    Arguments:
    Tx -- length of the input sequence
    Ty -- length of the output sequence
    n_a -- hidden state size of the Bi-LSTM
    n_s -- hidden state size of the post-attention LSTM
    human_vocab_size -- size of the python dictionary "human_vocab"
    machine_vocab_size -- size of the python dictionary "machine_vocab"

    Returns:
    model -- Keras model instance
    """

    # Define the inputs of your model with a shape (Tx,)
    # Define s0 and c0, initial hidden state for the decoder LSTM of shape (n_s,)
    X = Input(shape=(Tx, source_dictionary_size))
    s0 = Input(shape=(n_s,), name='s0')
    c0 = Input(shape=(n_s,), name='c0')
    s = s0
    c = c0

    # Initialize empty list of outputs
    outputs = []

    # Step 1: Define pre-attention Bi-LSTM.
    a1 = Bidirectional(LSTM(n_a, activation='relu',return_sequences=True,dropout=0.4))(X)
    a = Bidirectional(LSTM(n_a, activation='relu',return_sequences=True,dropout=0.4))(a1)
    print(a,a.shape)
    # Step 2: Iterate for Ty steps
    for t in range(Ty):

        # Step 2.A: Perform one step of the attention mechanism to get back the context vector at step t
        print("Before getting Context: a.shape,s.shape",a.shape,s.shape)
        context = one_step_attention(a, s)
        print("context.shape,s.shape,c.shape ",context.shape,s.shape,c.shape)
        # Step 2.B: Apply the post-attention LSTM cell to the "context" vector.
        _,s, c = post_activation_LSTM_cell(context, initial_state = [s,c])
        # Step 2.C: Apply Dense layer to the hidden state output of the post-attention LSTM
        out = output_layer(s)

        # Step 2.D: Append "out" to the "outputs" list
```

```
        outputs.append(out)

    # Step 3: Create model instance taking three inputs and returning the list of outputs.
    model = Model(inputs = [X, s0, c0], outputs = outputs)

    return model
```

Run the following cell to create your model.

In [28]:
```
#We are also printing the shapes, just for the purpose of debug.
model = model(Tx, Ty, n_a, n_s, len(hindi_dictionary), len(english_dictionary))

#We will need copy of the model which will use the weights from model.fit.
#This is done as it is observed there has been issues model.load_weights(weightFile)
loaded_model = model
```

```
Tensor("bidirectional_2/concat_2:0", shape=(?, ?, 1000), dtype=float32) (?, ?, 1000)
Before getting Context: a.shape,s.shape (?, ?, 1000) (?, 500)
s_prev.shape before repeator (?, 500)
s_prev.shape after repeator (?, 6, 500)
a.shape (?, ?, 1000)
concat.shape (?, 6, 1500)
e.shape (?, 6, 1)
alphas.shape (?, 6, 1)
context.shape (?, 1, 1000)
context.shape,s.shape,c.shape  (?, 1, 1000) (?, 500) (?, 500)
Before getting Context: a.shape,s.shape (?, ?, 1000) (?, 500)
s_prev.shape before repeator (?, 500)
s_prev.shape after repeator (?, 6, 500)
a.shape (?, ?, 1000)
concat.shape (?, 6, 1500)
e.shape (?, 6, 1)
alphas.shape (?, 6, 1)
context.shape (?, 1, 1000)
context.shape,s.shape,c.shape  (?, 1, 1000) (?, 500) (?, 500)
Before getting Context: a.shape,s.shape (?, ?, 1000) (?, 500)
s_prev.shape before repeator (?, 500)
s_prev.shape after repeator (?, 6, 500)
a.shape (?, ?, 1000)
concat.shape (?, 6, 1500)
e.shape (?, 6, 1)
alphas.shape (?, 6, 1)
context.shape (?, 1, 1000)
context.shape,s.shape,c.shape  (?, 1, 1000) (?, 500) (?, 500)
Before getting Context: a.shape,s.shape (?, ?, 1000) (?, 500)
s_prev.shape before repeator (?, 500)
s_prev.shape after repeator (?, 6, 500)
a.shape (?, ?, 1000)
concat.shape (?, 6, 1500)
e.shape (?, 6, 1)
alphas.shape (?, 6, 1)
context.shape (?, 1, 1000)
```

```
context.shape,s.shape,c.shape  (?, 1, 1000) (?, 500) (?, 500)
Before getting Context: a.shape,s.shape (?, ?, 1000) (?, 500)
s_prev.shape before repeator (?, 500)
s_prev.shape after repeator (?, 6, 500)
a.shape (?, ?, 1000)
concat.shape (?, 6, 1500)
e.shape (?, 6, 1)
alphas.shape (?, 6, 1)
context.shape (?, 1, 1000)
context.shape,s.shape,c.shape  (?, 1, 1000) (?, 500) (?, 500)
Before getting Context: a.shape,s.shape (?, ?, 1000) (?, 500)
s_prev.shape before repeator (?, 500)
s_prev.shape after repeator (?, 6, 500)
a.shape (?, ?, 1000)
concat.shape (?, 6, 1500)
e.shape (?, 6, 1)
alphas.shape (?, 6, 1)
context.shape (?, 1, 1000)
context.shape,s.shape,c.shape  (?, 1, 1000) (?, 500) (?, 500)
```

In [29]:
```
model.summary()
```

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 6, 2873)       0
____________________________________________________________________________________________________
bidirectional_1 (Bidirectional)  (None, 6, 1000)       13496000    input_1[0][0]
____________________________________________________________________________________________________
s0 (InputLayer)                  (None, 500)           0
____________________________________________________________________________________________________
bidirectional_2 (Bidirectional)  (None, 6, 1000)       6004000     bidirectional_1[0][0]
____________________________________________________________________________________________________
repeat_vector_1 (RepeatVector)   (None, 6, 500)        0           s0[0][0]
                                                                   lstm_1[0][1]
                                                                   lstm_1[1][1]
                                                                   lstm_1[2][1]
                                                                   lstm_1[3][1]
                                                                   lstm_1[4][1]
____________________________________________________________________________________________________
concatenate_1 (Concatenate)      (None, 6, 1500)       0           bidirectional_2[0][0]
                                                                   repeat_vector_1[0][0]
                                                                   bidirectional_2[0][0]
                                                                   repeat_vector_1[1][0]
                                                                   bidirectional_2[0][0]
                                                                   repeat_vector_1[2][0]
                                                                   bidirectional_2[0][0]
                                                                   repeat_vector_1[3][0]
                                                                   bidirectional_2[0][0]
                                                                   repeat_vector_1[4][0]
                                                                   bidirectional_2[0][0]
                                                                   repeat_vector_1[5][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 6, 1)          1501        concatenate_1[0][0]
                                                                   concatenate_1[1][0]
                                                                   concatenate_1[2][0]
                                                                   concatenate_1[3][0]
                                                                   concatenate_1[4][0]
                                                                   concatenate_1[5][0]
____________________________________________________________________________________________________
attention_weights (Activation)   (None, 6, 1)          0           dense_1[0][0]
                                                                   dense_1[1][0]
                                                                   dense_1[2][0]
```

```
                                                                dense_1[3][0]
                                                                dense_1[4][0]
                                                                dense_1[5][0]
____________________________________________________________________________________________________
dot_1 (Dot)                      (None, 1, 1000)      0         attention_weights[0][0]
                                                                bidirectional_2[0][0]
                                                                attention_weights[1][0]
                                                                bidirectional_2[0][0]
                                                                attention_weights[2][0]
                                                                bidirectional_2[0][0]
                                                                attention_weights[3][0]
                                                                bidirectional_2[0][0]
                                                                attention_weights[4][0]
                                                                bidirectional_2[0][0]
                                                                attention_weights[5][0]
                                                                bidirectional_2[0][0]
____________________________________________________________________________________________________
c0 (InputLayer)                  (None, 500)          0
____________________________________________________________________________________________________
lstm_1 (LSTM)                    [(None, 1, 500), (Non 3002000  dot_1[0][0]
                                                                s0[0][0]
                                                                c0[0][0]
                                                                dot_1[1][0]
                                                                lstm_1[0][1]
                                                                lstm_1[0][2]
                                                                dot_1[2][0]
                                                                lstm_1[1][1]
                                                                lstm_1[1][2]
                                                                dot_1[3][0]
                                                                lstm_1[2][1]
                                                                lstm_1[2][2]
                                                                dot_1[4][0]
                                                                lstm_1[3][1]
                                                                lstm_1[3][2]
                                                                dot_1[5][0]
                                                                lstm_1[4][1]
                                                                lstm_1[4][2]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 2620)         1312620   lstm_1[0][1]
                                                                lstm_1[1][1]
                                                                lstm_1[2][1]
                                                                lstm_1[3][1]
                                                                lstm_1[4][1]
```

```
                                                lstm_1[5][1]
====================================================================================================
Total params: 23,816,121
Trainable params: 23,816,121
Non-trainable params: 0
____________________________________________________________________________________________________
```

After creating your model in Keras, we need to compile it and define what loss, optimizer and metrics your are want to use. Compile your model using `categorical_crossentropy` loss, and optimizer rmsprop or Adam.

In [30]:
```
from keras.optimizers import RMSprop
out = model.compile(optimizer='rmsprop'#(lr=0.001, beta_1=0.7, beta_2=0.8, decay=0.02)
                    ,metrics=['accuracy'],
                    loss='categorical_crossentropy')
out
```

The last step is to define all your inputs and outputs to fit the model:

- We already have X of shape (m, T_x) containing the training examples.
- We need to create `s0` and `c0` to initialize your `post_activation_LSTM_cell` with 0s.
- Given the `model()` you coded, you need the "outputs" to be a list of elements of shape (m, T_y). So that: `outputs[i][0], ..., outputs[i][Ty]` represent the true labels (characters) corresponding to the i^{th} training example (`X[i]`). More generally, `outputs[i][j]` is the true label of the j^{th} character in the i^{th} training example.

In [31]:
```
s0 = np.zeros((len(dataset_list), n_s))
c0 = np.zeros((len(dataset_list), n_s))
outputs = list(Yoh.swapaxes(0,1))
```

In [32]:
```
#Divide data in to train & test
#How much percentage of total data you need enter the number for training_sample_percentage
training_sample_percentage=98
training_sample_count=(round(X.shape[0]*training_sample_percentage/100))
#training_sample_count=2
#For to cover rest of data
testing_sample_count=X.shape[0]-training_sample_count
#testing_sample_count
testing_sample_index=training_sample_count+testing_sample_count

print("Total Samples,Training,Testing",X.shape[0],training_sample_count,testing_sample_count)
trainXoh=Xoh[0:training_sample_count]
trainYoh=Yoh[0:training_sample_count]
testXoh=Xoh[training_sample_count:testing_sample_index]
testYoh=Yoh[training_sample_count:testing_sample_index]
print("Training X Shape and Y Shape",trainXoh.shape,trainYoh.shape)
print("Testing X Shape and Y Shape",testXoh.shape,testYoh.shape)
train_outputs = list(trainYoh.swapaxes(0,1))
test_outputs = list(testYoh.swapaxes(0,1))
```

```
Total Samples,Training,Testing 2867 2810 57
Training X Shape and Y Shape (2810, 6, 2873) (2810, 6, 2620)
Testing X Shape and Y Shape (57, 6, 2873) (57, 6, 2620)
```

In [33]:
```
train_s0 = np.zeros((training_sample_count, n_s))
train_c0 = np.zeros((training_sample_count, n_s))
trainX=[trainXoh, train_s0, train_c0]
trainY=train_outputs
test_s0 = np.zeros((testing_sample_count, n_s))
test_c0 = np.zeros((testing_sample_count, n_s))
testX=[testXoh, test_s0, test_c0]
testY=test_outputs
print(s0.shape,c0.shape)
```

```
(2867, 500) (2867, 500)
```

In [ ]:
```
#Run the model.fit. If only best validation model needs to be saved, then change save_best_only=True in chec
from keras.callbacks import ModelCheckpoint
checkpoint = ModelCheckpoint('main_model_weights_new.h5', monitor='val_loss', verbose=1, save_best_only=Fals
model.fit(trainX, trainY, epochs=200, batch_size=20, validation_data=(testX, testY), callbacks=[checkpoint])
#Hoping to save model without errors.
model.save('main_model.h5')
model.save_weights('final_model_weights.h5')
```

```
Train on 2810 samples, validate on 57 samples
Epoch 1/200
2800/2810 [============================>.] - ETA: 0s - loss: 31.8713 - dense_2_loss_1: 4.2636 - dense_2
_loss_2: 6.0400 - dense_2_loss_3: 5.9195 - dense_2_loss_4: 6.0314 - dense_2_loss_5: 5.3107 - dense_2_lo
ss_6: 4.3060 - dense_2_acc_1: 0.1707 - dense_2_acc_2: 0.0754 - dense_2_acc_3: 0.0600 - dense_2_acc_4:
0.1007 - dense_2_acc_5: 0.2325 - dense_2_acc_6: 0.4171Epoch 00000: saving model to main_model_weights_n
ew.h5
2810/2810 [==============================] - 129s - loss: 31.8792 - dense_2_loss_1: 4.2607 - dense_2_lo
ss_2: 6.0408 - dense_2_loss_3: 5.9215 - dense_2_loss_4: 6.0289 - dense_2_loss_5: 5.3158 - dense_2_loss_
6: 4.3115 - dense_2_acc_1: 0.1705 - dense_2_acc_2: 0.0754 - dense_2_acc_3: 0.0601 - dense_2_acc_4: 0.10
07 - dense_2_acc_5: 0.2320 - dense_2_acc_6: 0.4167 - val_loss: 33.0448 - val_dense_2_loss_1: 4.3732 - v
al_dense_2_loss_2: 6.3249 - val_dense_2_loss_3: 6.1032 - val_dense_2_loss_4: 6.3106 - val_dense_2_loss_
5: 5.4926 - val_dense_2_loss_6: 4.4403 - val_dense_2_acc_1: 0.1754 - val_dense_2_acc_2: 0.0175 - val_de
nse_2_acc_3: 0.0702 - val_dense_2_acc_4: 0.1228 - val_dense_2_acc_5: 0.2632 - val_dense_2_acc_6: 0.4211
Epoch 2/200
2800/2810 [============================>.] - ETA: 0s - loss: 30.8560 - dense_2_loss_1: 4.0229 - dense_2
_loss_2: 5.8179 - dense_2_loss_3: 5.7485 - dense_2_loss_4: 5.9012 - dense_2_loss_5: 5.1645 - dense_2_lo
ss_6: 4.2010 - dense_2_acc_1: 0.1732 - dense_2_acc_2: 0.0811 - dense_2_acc_3: 0.0718 - dense_2_acc_4:
0.1129 - dense_2_acc_5: 0.2389 - dense_2_acc_6: 0.4154Epoch 00001: saving model to main_model_weights_n
```

3. Quantitative Analysis

While training you can see the loss as well as the accuracy on each of the positions of the output. The output snapshot below gives you an example of what the accuracies could be at 100th iteration in above settings:

Epoch 100/100 2800/2810 [============================>.] - ETA: 0s - loss: 5.1898 - dense_2_loss_1: 0.7850 - dense_2_loss_2: 0.8572 - dense_2_loss_3: 0.8971 - dense_2_loss_4: 0.8881 - dense_2_loss_5: 0.9539 - dense_2_loss_6: 0.8085 - dense_2_acc_1: 0.7875 - dense_2_acc_2: 0.7571 - dense_2_acc_3: 0.7443 - dense_2_acc_4: 0.7479 - dense_2_acc_5: 0.7379 - dense_2_acc_6: 0.7975Epoch 00099: saving model to main_model_weights.h5 2810/2810 [==============================] - 17s - loss: 5.1868 - dense_2_loss_1: 0.7848 - dense_2_loss_2: 0.8570 - dense_2_loss_3: 0.8973 - dense_2_loss_4: 0.8891 - dense_2_loss_5: 0.9526 - dense_2_loss_6: 0.8061 - dense_2_acc_1: 0.7872 - dense_2_acc_2: 0.7569 - dense_2_acc_3: 0.7438 -

dense_2_acc_4: 0.7480 - dense_2_acc_5: 0.7384 - dense_2_acc_6: 0.7982 - val_loss: 40.8334 - val_dense_2_loss_1: 4.3684 - val_dense_2_loss_2: 7.5746 - val_dense_2_loss_3: 6.9904 - val_dense_2_loss_4: 8.9484 - val_dense_2_loss_5: 6.6466 - val_dense_2_loss_6: 6.3049 - val_dense_2_acc_1: 0.4561 - val_dense_2_acc_2: 0.2807 - val_dense_2_acc_3: 0.1754 - val_dense_2_acc_4: 0.1579 - val_dense_2_acc_5: 0.2456 - val_dense_2_acc_6: 0.3684

##
Thus at 100-th iteration with unaltered settings above, `dense_2_acc_6: 0.7975` means that you are predicting the 6th word of the output correctly 79% of the time in the current batch of data. Also val_dense_2_acc_6: 0.3684 means the 6th digit prediction accuracy is 36%

4. Qualitative Analysis

Following code will load the saved weights which will be used to do a qualitative analysis.

In [ ]:
```
from keras.models import load_model
loaded_model.load_weights('main_model_weights.h5')
```

We can now see the results on new examples.

In [ ]:
```
EXAMPLES=hindi_sentences_list[0:1]
true_test="हमने खरीदी"
EXAMPLES.append(true_test)
EXAMPLES
```

In [ ]:
```
EXAMPLES_CODED=get_padded_encoding(EXAMPLES,hindi_dictionary,6)
print(EXAMPLES_CODED,EXAMPLES_CODED.shape,hindi_encoding.shape)
```

In [ ]:
```
i=0
for example in EXAMPLES_CODED:
    iteration=i+1
    source = example
    source = np.array(list(map(lambda x: to_categorical(x, num_classes=len(hindi_dictionary)), source))).swa
    prediction = model.predict([source,train_s0, train_c0])
    #print ("Prediction, Type & Shape:",prediction,type(prediction),len(prediction))
    prediction = np.argmax(prediction, axis = -1)
    #print ("Prediction, After Argmax:",prediction)
    output = [revere_dictionary_english[int(i)] for i in prediction]
    print("\n ##### \n")
    print("Hindi",EXAMPLES[i])
    if (iteration!=EXAMPLES_CODED.shape[0]):
        print("Expected:",english_sentences_list[i])
    print("Predicted output:", ' '.join(output))
    #print ("Prediction:",list(prediction))
    i=i+1
```

We should be able to see following results. We have first sentence from training example and another one from true test. We can see that there is a pretty good translation for the data from training and for true test it was able to predict first place pretty accurately but failed in following portions.

#####

Hindi इसको अपना घर ही समझो Expected: Please make yourself at home Predicted output: Please yourself yourself home

#####

Hindi हमने खरीदी Predicted output: We leave up

5 References


```
Neural Machine Translation by Jointly Learning to Align and Translate: Dzmitry Bahdanau, Kyunghyun Cho,
Yoshua Bengio https://arxiv.org/pdf/1409.0473.pdf
```


Appendix

One thing we can do to improve the model is instead of one hot encodings of words of length vocabulary, get the word2vec vectors for each word with fixed length.

Another thing that can be done is train only short sentences.

In below section we will provide the functions to help to do the tasks.

In [ ]:
```
#Converting input to word2vec.
def sentences_to_word2vec_input_format(language_sentences_list):
    word2vec_sentence_feed=list()
    for sentence in language_sentences_list:
        word2vec_sentence_feed.append(sentence.split())
    return(word2vec_sentence_feed)
english_sentences_w2v_format=sentences_to_word2vec_input_format(english_sentences_list)
hindi_sentences_w2v_format=sentences_to_word2vec_input_format(hindi_sentences_list)
```

In [ ]:
```
from gensim.models import Word2Vec
# train model
english_model = Word2Vec(english_sentences_w2v_format, min_count=1)
english_words_vocab = list(english_model.wv.vocab)
hindi_model = Word2Vec(hindi_sentences_w2v_format, min_count=1)
english_words_vocab = list(hindi_model.wv.vocab)
```

In [ ]:
```
def sentences_to_w2vec(language_encoding,revere_dictionary_language,language_model):
    import numpy as np
    sentence_level_w2vec_list=[]
    #arr = np.empty((2,), float)
    number_of_sentences=language_encoding.shape[0]
    for i in range(0,number_of_sentences):
        language_list_padded=[]
        #print (english_encoding[i])
        for key in language_encoding[i]:
            #print(revere_dictionary_english[key])
            word=(revere_dictionary_language[key])
            try:
                #print("Found word Shape of word vector",(english_model[word]).shape,arr.shape)
                language_list_padded.append(language_model[word])
            except KeyError:
                unk='<unk>'
                #print("not found! Assigning Unknown Vector",  (english_model[unk]).shape)
                language_list_padded.append(language_model[unk])
        #print(np.array(language_list_padded))
        sentence_level_w2vec_list.append((np.array(language_list_padded)))
    sentence_level_w2vec=np.array(sentence_level_w2vec_list)
    return(sentence_level_w2vec)
```

In [ ]:
```
X=hindi_encoding
Y=english_encoding
#Y will remain the same.
Yoh=np.array(list(map(lambda x: to_categorical(x, num_classes=len(english_dictionary)), Y)))
print("X.shape:", X.shape)
print("Y.shape:", Y.shape)
print("Yoh.shape:", Yoh.shape)
```

In [ ]:
```
#Run this if you want word2vec instead of One hot encoding
#Naming it still as X0h and Yoh to avoid changes in too many places further.
#Yoh
Xoh=sentences_to_w2vec(hindi_encoding,revere_dictionary_hindi,hindi_model)
print("Xoh.shape:", Xoh.shape)
print("Yoh.shape:", Yoh.shape)
```

One might also like to get the sentences of only specific length from source as well as target, for example get all sentences which has maximum 5 words and in hindi maximum 8 words. Use below function and feed the length you need.

In [ ]:
```
#dataset=Ndarray with following dimentions (sentence_length, 2)
#source_len is the length of language in dataset[0][1]
#target_len is the length of language in dataset[0][0]
def get_sentences_subset(dataset,source_len,target_len):
    limited=dataset
    indexes_list=[]
    for indexes in range(0,limited.shape[0]):
        #print(len(limited[i][0].split()),len(limited[i][1].split()))
        eng_len=len(limited[indexes][0].split())
        hin_len=len(limited[indexes][1].split())
        state1=(eng_len<target_len)
        state2=(hin_len<source_len)
        final=state2&state1
        #print(eng_len,hin_len,final)
        #print(state1,state2,final)
        if (final):
            indexes_list.append(indexes)
    #print(indexes_list,type(indexes_list))
    return(limited[indexes_list])
```