Sequence to Sequence Word level Model

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

In [20]:

```
from __future__ import print_function
#import tensorflow as tf
from keras.models import Model
from keras.layers import Input, LSTM, Dense
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
import numpy as np
import pandas as pd
import nltk
import matplotlib.pyplot as plt
pd.set_option('display.max_columns', None)
```

1. What do the variables below mean? How do they effect the model?

In [21]:

```
batch_size = 64  # Batch size for training.
epochs = 50  # Number of epochs to train for.
latent_dim = 512  # Latent dimensionality of the encoding space.
num_samples = 7000  # Number of samples to train on.
# Path to the data txt file on disk.
data_path = 'cleaned_data.txt'
#run_opts = tf.RunOptions(report_tensor_allocations_upon_oom = True)
```

Vectorize data

to encode every character

In [22]:

```
# Vectorize the data.
input_texts = []
target_texts = []
input_words = set()
target_words = set()

with open(data_path, 'r', encoding='utf-8') as f:
    lines = f.read().split('\n')
for line in lines[: min(num_samples, len(lines) - 1)]:
    index, input_text, target_text = line.split('\t')
    # We use "tab" as the "start sequence" character
    # for the targets, and "\n" as "end sequence" character.
    target_text = 'START_ '+target_text+ ' _END'
    input_texts.append(input_text)
    target_texts.append(target_text)

    input_word_tokens=nltk.word_tokenize(input_text)
    target_word_tokens=nltk.word_tokenize(target_text)

    for word in input_word_tokens:
        if word not in input_words:
            input_words.add(word)
    for word in target_word_tokens:
        if word not in target_words:
            target_words.add(word)
#input_words.add('')
#target_words.add('')
input_words = sorted(list(input_words))

target_words = sorted(list(target_words))

num_encoder_tokens = len(input_words)
num_decoder_tokens = len(target_words)
max_encoder_seq_length = max([len(nltk.word_tokenize(txt)) for txt in input_texts])
max_decoder_seq_length = max([len(nltk.word_tokenize(txt)) for txt in target_texts])

print('Number of samples:', len(input_texts))
print('Number of unique input tokens:', num_encoder_tokens)
print('Number of unique output tokens:', num_decoder_tokens)
print('Max sequence length for inputs:', max_encoder_seq_length)
print('Max sequence length for outputs:', max_decoder_seq_length)
print('-------Word corpus-------')
#print(input_words)
#print(target_words)
```

```
Number of samples: 7000
Number of unique input tokens: 6567
Number of unique output tokens: 6463
Max sequence length for inputs: 43
Max sequence length for outputs: 43
-------Word corpus-------
```

What are the dimensions of the encoder input, decoder input and decoder target? How many features and timesteps?

- encoder_input_data is a 3D array of shape (num_pairs, max_english_sentence_length, num_english_characters) containing a one-hot vectorization of the English sentences.

- decoder_input_data is a 3D array of shape (num_pairs, max_french_sentence_length, num_french_characters) containg a one-hot vectorization of the French sentences.
- decoder_target_data is the same as decoder_input_data but offset by one timestep. decoder_target_data[:, t, :] will be the same as decoder_input_data[:, t + 1, :].

In [23]:

```
input_token_index = dict(
    [(word, i) for i, word in enumerate(input_words)])
target_token_index = dict(
    [(word, i) for i, word in enumerate(target_words)])

encoder_input_data = np.zeros(
    (len(input_texts), max_encoder_seq_length, num_encoder_tokens),
    dtype='float16')
decoder_input_data = np.zeros(
    (len(input_texts), max_decoder_seq_length, num_decoder_tokens),
    dtype='float16')

decoder_target_data = np.zeros(
    (len(input_texts), max_decoder_seq_length, num_decoder_tokens),
    dtype='float16')

for i, (input_text, target_text) in enumerate(zip(input_texts, target_texts)):
    for t, word in enumerate(nltk.word_tokenize(input_text)):
        encoder_input_data[i, t, input_token_index[word]] = 1.

    for t, word in enumerate(nltk.word_tokenize(target_text)):
        # decoder_target_data is ahead of decoder_input_data by one timestep
        decoder_input_data[i, t, target_token_index[word]] = 1.
        if t > 0:
            # decoder_target_data will be ahead by one timestep
            # and will not include the start character.
            decoder_target_data[i, t - 1, target_token_index[word]] = 1.
```

In [ ]:

```

```

Simple Word to Word Model

Encode-Decoder Model.

In [24]:

```
#EARLY STOPPING
#early_stopping = EarlyStopping(monitor='val_loss', patience=25)
#MODEL CHECKPOINT
ckpt_file = 'model.h1.27_jul_19'
checkpoint = ModelCheckpoint(ckpt_file, monitor='val_loss', verbose=1, save_best_only=True,
# Define an input sequence and process it.
encoder_inputs = Input(shape=(None, num_encoder_tokens))
encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
# We discard `encoder_outputs` and only keep the states.
encoder_states = [state_h, state_c]

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None, num_decoder_tokens))
# We set up our decoder to return full output sequences,
# and to return internal states as well. We don't use the
# return states in the training model, but we will use them in inference.
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs,
                                     initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)

# Define the model that will turn
# `encoder_input_data` & `decoder_input_data` into `decoder_target_data`
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

```
WARNING: Logging before flag parsing goes to stderr.
W0728 15:54:05.161092 14868 deprecation_wrapper.py:119] From c:\users\robust
us\appdata\local\programs\python\python37\lib\site-packages\keras\backend\te
nsorflow_backend.py:74: The name tf.get_default_graph is deprecated. Please
use tf.compat.v1.get_default_graph instead.

W0728 15:54:05.184001 14868 deprecation_wrapper.py:119] From c:\users\robust
us\appdata\local\programs\python\python37\lib\site-packages\keras\backend\te
nsorflow_backend.py:517: The name tf.placeholder is deprecated. Please use t
f.compat.v1.placeholder instead.

W0728 15:54:05.188990 14868 deprecation_wrapper.py:119] From c:\users\robust
us\appdata\local\programs\python\python37\lib\site-packages\keras\backend\te
nsorflow_backend.py:4138: The name tf.random_uniform is deprecated. Please u
se tf.random.uniform instead.
```

DO NOT RUN when loading previously saved model

only run when running model afresh

In [25]:

```
# Run training
#model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['acc'],options =
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',metrics=['acc'])

model.summary()
```

```
W0728 15:54:11.749848 14868 deprecation_wrapper.py:119] From c:\users\robust
us\appdata\local\programs\python\python37\lib\site-packages\keras\optimizer
s.py:790: The name tf.train.Optimizer is deprecated. Please use tf.compat.v
1.train.Optimizer instead.

W0728 15:54:11.771790 14868 deprecation_wrapper.py:119] From c:\users\robust
us\appdata\local\programs\python\python37\lib\site-packages\keras\backend\te
nsorflow_backend.py:3295: The name tf.log is deprecated. Please use tf.math.
log instead.


____________________________________________________________________________
______________________
Layer (type)                    Output Shape         Param #     Connected t
o
============================================================================
======================
input_1 (InputLayer)            (None, None, 6567)   0
____________________________________________________________________________
______________________
input_2 (InputLayer)            (None, None, 6463)   0
____________________________________________________________________________
______________________
lstm_1 (LSTM)                   [(None, 512), (None, 14499840    input_1[0]
[0]
____________________________________________________________________________
______________________
lstm_2 (LSTM)                   [(None, None, 512),  14286848    input_2[0]
[0]
                                                                 lstm_1[0]
[1]
                                                                 lstm_1[0]
[2]
____________________________________________________________________________
______________________
dense_1 (Dense)                 (None, None, 6463)   3315519     lstm_2[0]
[0]
============================================================================
======================
Total params: 32,102,207
Trainable params: 32,102,207
Non-trainable params: 0
____________________________________________________________________________
______________________
```

DO NOT RUN when loading previously saved model

only run when running model afresh

How to save and reload same model?

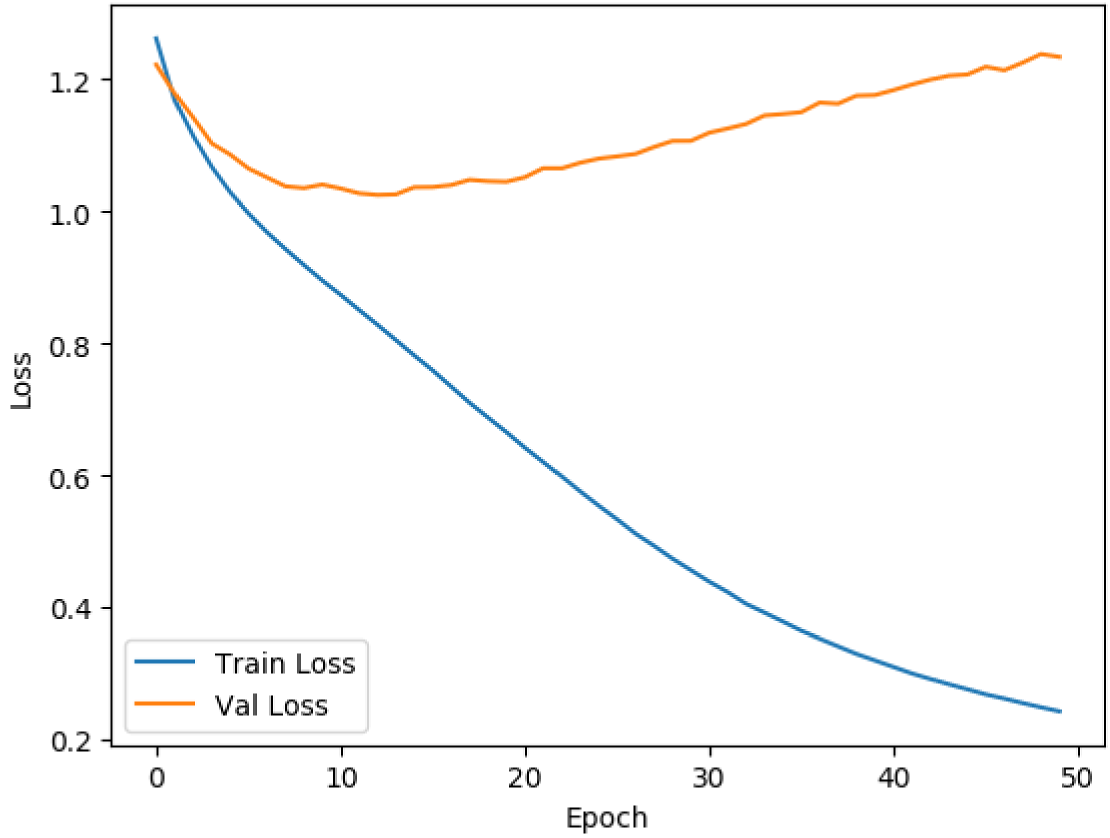
In [26]:

```
'''model.fit([encoder_input_data, decoder_input_data], decoder_target_data,
          batch_size=batch_size,
          epochs=20,callbacks=[early_stopping],
          validation_split=0.2)'''
history=model.fit([encoder_input_data, decoder_input_data], decoder_target_data,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2, callbacks=[checkpoint], verbose=1)
# Save model
#model.save('Ass3_s2s.h5')
model.save('Project_7000_w2w_s2s_64_512_50e.h5')
```

```
W0728 15:54:37.988819 14868 deprecation.py:323] From c:\users\robustus\app
data\local\programs\python\python37\lib\site-packages\tensorflow\python\op
s\math_grad.py:1250: add_dispatch_support.<locals>.wrapper (from tensorflo
w.python.ops.array_ops) is deprecated and will be removed in a future vers
ion.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
W0728 15:54:38.973189 14868 deprecation_wrapper.py:119] From c:\users\robu
stus\appdata\local\programs\python\python37\lib\site-packages\keras\backen
d\tensorflow_backend.py:986: The name tf.assign_add is deprecated. Please
use tf.compat.v1.assign_add instead.


Train on 5600 samples, validate on 1400 samples
Epoch 1/50
5600/5600 [==============================] - 100s 18ms/step - loss: 1.2623
- acc: 0.0258 - val_loss: 1.2225 - val_acc: 0.0227

Epoch 00001: val_loss improved from inf to 1.22245, saving model to model.
h1_27_jul_19
```

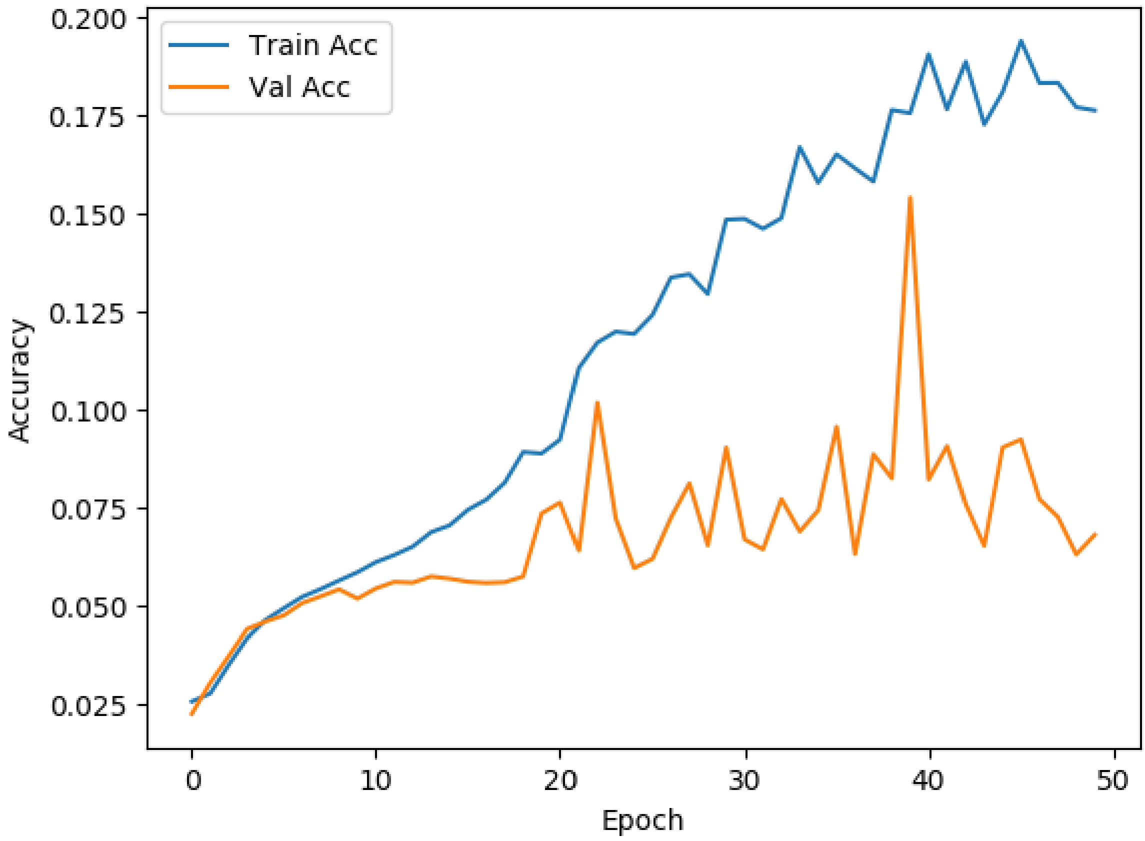
In [27]:

```
import matplotlib.pyplot as plt
def plot_loss_history(history):
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.plot(history.epoch, np.array(history.history['loss']),
               label='Train Loss')
    plt.plot(history.epoch, np.array(history.history['val_loss']),
           label = 'Val Loss')
    plt.legend()
    #plt.ylim([0.05, 1])

plot_loss_history(history)
```



In [28]:

```
import matplotlib.pyplot as plt
def plot_loss_history(history):
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.plot(history.epoch, np.array(history.history['acc']),
               label='Train Acc')
    plt.plot(history.epoch, np.array(history.history['val_acc']),
           label = 'Val Acc')
    plt.legend()
    #plt.ylim([0.05, 1])

plot_loss_history(history)
```



Run only when recalling new model after restarting kernel

When kernel crashes and for retraining with just 1 epoch

In [6]:

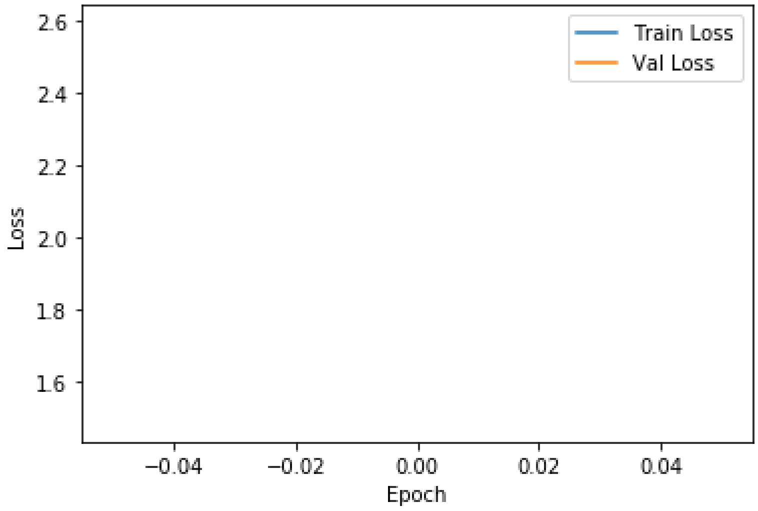
```
import tensorflow as tf
#Call a saved model
#tf.reset_default_graph()
from tensorflow.core.protobuf import rewriter_config_pb2
from tensorflow.keras.backend import set_session
tf.keras.backend.clear_session()  # For easy reset of notebook state.

config_proto = tf.ConfigProto()
off = rewriter_config_pb2.RewriterConfig.OFF
config_proto.graph_options.rewrite_options.arithmetic_optimization = off
session = tf.Session(config=config_proto)
set_session(session)
with tf.device('/cpu:0'):
    new_model = tf.keras.models.load_model('Project_7500_w2w_s2s_512_40e.h5')

#Run a new model with saved weights
#new_model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
    new_model.summary()

#to reinstate the model, running for just one epoch
    new_history=new_model.fit([encoder_input_data, decoder_input_data], decoder_target_data
              batch_size=batch_size,
              epochs=1,
              validation_split=0.2)
# Save model
#new_model.save('revised_Ass3_s2s_100.h5')
```

```
W0727 20:49:50.234421  4636 deprecation.py:323] From c:\users\robustus\appda
ta\local\programs\python\python37\lib\site-packages\tensorflow\python\ops\ma
th_grad.py:1250: add_dispatch_support.<locals>.wrapper (from tensorflow.pyth
on.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where

Model: "model_1"
____________________________________________________________________________
______________________
Layer (type)                    Output Shape         Param #     Connected t
o
============================================================================
======================
input_1 (InputLayer)            [(None, None, 6567)] 0
____________________________________________________________________________
______________________
input_2 (InputLayer)            [(None, None, 6463)] 0
____________________________________________________________________________
______________________
lstm_1 (LSTM)                   [(None, 512), (None, 14499840    input_1[0]
[0]
____________________________________________________________________________
______________________
lstm_2 (LSTM)                   [(None, None, 512),  14286848    input_2[0]
[0]
                                                                 lstm_1[0]
[1]
                                                                 lstm_1[0]
[2]
____________________________________________________________________________
______________________
```

```
dense_1 (Dense)                 (None, None, 6463)   3315519     lstm_2[0]
[0]
============================================================================
======================
Total params: 32,102,207
Trainable params: 32,102,207
Non-trainable params: 0
____________________________________________________________________________
______________________
Train on 5600 samples, validate on 1400 samples
5600/5600 [==============================] - 673s 120ms/sample - loss: 0.002
6 - acc: 0.2300 - val_loss: 1.6119 - val_acc: 0.0772
```

In [70]:

```python
import matplotlib.pyplot as plt
def plot_loss_new_history(new_history):
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.plot(new_history.epoch, np.array(new_history.history['loss']),
             label='Train Loss')
    plt.plot(new_history.epoch, np.array(new_history.history['val_loss']),
         label = 'Val Loss')
    plt.legend()
    #plt.ylim([0.05, 1])

plot_loss_new_history(new_history)
```



Inference Mode

Re-tuning the model to accept direct inputs to Decoder along with states from encoder

In [29]:

```
# Next: inference mode (sampling).
# Here's the drill:
# 1) encode input and retrieve initial decoder state
# 2) run one step of decoder with this initial state
# and a "start of sequence" token as target.
# Output will be the next target token
# 3) Repeat with the current target token and current states

# Define sampling models
encoder_model = Model(encoder_inputs, encoder_states)

decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
decoder_outputs, state_h, state_c = decoder_lstm(
    decoder_inputs, initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)
decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs] + decoder_states)
```

In [30]:

```
# Reverse-lookup token index to decode sequences back to
# something readable.
reverse_input_word_index = dict(
    (i, word) for word, i in input_token_index.items())
reverse_target_word_index = dict(
    (i, word) for word, i in target_token_index.items())
```

Why are we saving h, c from decoder?

In [33]:

```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    states_value = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1, 1, num_decoder_tokens))
    # Populate the first character of target sequence with the start character.
    target_seq[0, 0, target_token_index['START_']] = 1.

    # Sampling loop for a batch of sequences
    # (to simplify, here we assume a batch of size 1).
    stop_condition = False
    decoded_sentence = ''
    while stop_condition == False:
        output_tokens, h, c = decoder_model.predict(
            [target_seq] + states_value)

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_word = reverse_target_word_index[sampled_token_index]
        if (sampled_word != '_END'):
            decoded_sentence += ' '+sampled_word

        # Exit condition: either hit max length
        # or find stop character.
        if (sampled_word == '_END' or len(decoded_sentence) > max_decoder_seq_length):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1, 1, num_decoder_tokens))
        target_seq[0, 0, sampled_token_index] = 1.

        # Update states
        states_value = [h, c]

    return decoded_sentence
```

In [34]:

```
for seq_index in range(20):
    # Take one sequence (part of the training set)
    # for trying out decoding.
    input_seq = encoder_input_data[seq_index: seq_index + 1]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('Input sentence:', input_texts[seq_index])
    print('Target sentence:', target_texts[seq_index])
    print('Decoded sentence:', 'START_ '+decoded_sentence+' _END')
```

```
-
Input sentence: I do not want to die.
Target sentence: START_ मैं मरना नहीं चाहता. _END
Decoded sentence: START_  मैं तुम्हें पता है , डौग चाहते हैं , मैं एक _END
-
Input sentence: It's the same country I think.
Target sentence: START_ यह मुझे लगता है कि एक ही देश है. _END
Decoded sentence: START_  यह एक अच्छा है . _END
-
Input sentence: Then they'll be crying like babies.
Target sentence: START_ फिर ये नन्हें बच्चों की तरह रोएँगे। _END
Decoded sentence: START_  - हम अभी तक मदद के लिए पूछ नहीं है . _END
-
Input sentence: - No, I need power up!
Target sentence: START_ नहीं, मुझे पावर की जरुरत है ! _END
Decoded sentence: START_  - नहीं , मैं ऐसा नहीं करूंगा . _END
-
Input sentence: I will not eat him.
Target sentence: START_ मैं उसे नहीं खा जाएगा. _END
Decoded sentence: START_  मैं तुम्हें पता है , डौग चाहते हैं , मैं एक _END
-
Input sentence: You gotta get me to Charleston.
Target sentence: START_ आप चार्ल्सटन करने के लिए मुझे जाना होगा. _END
Decoded sentence: START_  आप मुझे पता नहीं था कि मैं जीवन या या मैं कहीं _END
-
Input sentence: - NO, HE'S NOT MY DAD.
Target sentence: START_ - नहीं, वह मेरे पिता नहीं है. _END
Decoded sentence: START_  - नहीं , वह नहीं नहीं है . _END
-
Input sentence: I told her we rest on Sundays.
Target sentence: START_ मैं रविवार को उसे हम बाकी बताया. _END
Decoded sentence: START_  मैं तुम्हें पता है , डौग चाहते हैं , मैं एक _END
-
Input sentence: You could've at least informed me, right?
Target sentence: START_ तुम्हें कम से कम मुझे तो बताना चाहिए था,ना? _END
Decoded sentence: START_  आप अपने आप को मार डाला मिलता है ? _END
-
Input sentence: Your little bitch says you're gonna put me in jail!
Target sentence: START_ तेरी कमीनी कहती है कि वो मुझे जेल भेजेगी ! _END
Decoded sentence: START_  एक आदमी को . _END
-
Input sentence: - You can call me whatever you like.
Target sentence: START_ - तुम मुझे फोन कर सकते हैं जो कुछ भी आप की तरह। _END
Decoded sentence: START_  - यह एक गहने की दुकान है . _END
-
Input sentence: - You don't just kill a guy like that!
Target sentence: START_ - तुम बस की तरह है कि एक आदमी को मार नहीं है! _END
Decoded sentence: START_  - आप यह जानते हैं . _END
-
```

```
Input sentence: You sent these?
Target sentence: START_ आप इन भेजा? _END
Decoded sentence: START_  आप एक चुड़ैल हैं ? _END
-
Input sentence: I really loved him.
Target sentence: START_ मैं वास्तव में उसे प्यार करता था। _END
Decoded sentence: START_  मैं अपने पिता की मौत पर आप का समर्थन किया . _END
-
Input sentence: I ain't much at guessing games.
Target sentence: START_ मैं अनुमान लगाने के खेल में ज्यादा नहीं है. _END
Decoded sentence: START_  मैं तुम्हें पता है , डौग चाहते हैं , मैं एक _END
-
Input sentence: You're sick and I can help you.
Target sentence: START_ तुम बीमार हो और मैं तुम्हारी मदद कर सकते हैं। _END
Decoded sentence: START_  आप एक चुड़ैल हैं ? _END
-
Input sentence: Mike, do I get to ride with you?
Target sentence: START_ माइक, मैं आप के साथ सवारी करने के लिए मिलता है? _END
Decoded sentence: START_  अरे , क्या आप इस आदमी को यह है ? _END
-
Input sentence: What do you fucking think?
Target sentence: START_ आपको क्या लगता है कि बकवास है? _END
Decoded sentence: START_  क्या आप के बारे में , आप किसी भी पैसा है तुम्हारे _END
-
Input sentence: I know that woman you love also is ready to forgive you.
Target sentence: START_ मैं आप उसे माफ करने के लिए तैयार भी प्यार औरत को जानते हैं.
_END
Decoded sentence: START_  मैं आप के साथ पर चढ़ा देना चाहिए था कभी नहीं _END
-
Input sentence: Don't do it, man.
Target sentence: START_ , आदमी ऐसा मत करो. _END
Decoded sentence: START_  क्या आप के लिए अपने माध्यम के लिए किसी की तरह _END
```

In [40]:

```
ip_seq=[]
op_seq=[]
dec_seq=[]
b1=[]
b2=[]
b3=[]
b4=[]
b_cum=[]
```

In [ ]:

In [41]:

```
# n-gram individual BLEU
from nltk.translate.bleu_score import sentence_bleu
for seq_index in range(100):
    # Take one sequence (part of the training set)
    # for trying out decoding.
    input_seq = encoder_input_data[seq_index: seq_index + 1]
    target_sentence = target_texts[seq_index]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('Input sentence:', input_texts[seq_index])
    print('Target sentence:',target_sentence)
    print('Decoded sentence:','START_ '+decoded_sentence+' _END')
    x1=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(1, 0,
    print('Individual 1-gram: %f' % x1)
    x2=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0, 1,
    print('Individual 2-gram: %f' % x2)
    x3=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0, 0,
    print('Individual 3-gram: %f' % x3)
    x4=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0,0,0,
    print('Individual 4-gram: %f' % x4)
    score = sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0
    print('4-gram cummulative score: ',score)
    ip_seq.append(input_texts[seq_index])
    op_seq.append(target_sentence)
    dec_seq.append('START_ '+decoded_sentence+' _END')
    b1.append(x1)
    b2.append(x2)
    b3.append(x3)
    b4.append(x4)
    b_cum.append(score)
```

```
-
Input sentence: I do not want to die.
Target sentence: START_ मैं मरना नहीं चाहता. _END
Decoded sentence: START_  मैं तुम्हें पता है , डौंग चाहते हैं , मैं एक _END
Individual 1-gram: 0.464286
Individual 2-gram: 0.400000
Individual 3-gram: 0.259259
Individual 4-gram: 0.188679
4-gram cummulative score:  0.3087279361290344
-
Input sentence: It's the same country I think.
Target sentence: START_ यह मुझे लगता है कि एक ही देश है. _END
Decoded sentence: START_  यह एक अच्छा है . _END
Individual 1-gram: 0.513934
Individual 2-gram: 0.447122
Individual 3-gram: 0.353282
Individual 4-gram: 0.252223
4-gram cummulative score:  0.3782767339010976
-
```

In [42]:

```
# n-gram individual BLEU
from nltk.translate.bleu_score import sentence_bleu
for seq_index in range(3450,3550):
    # Take one sequence (part of the training set)
    # for trying out decoding.
    input_seq = encoder_input_data[seq_index: seq_index + 1]
    target_sentence = target_texts[seq_index]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('Input sentence:', input_texts[seq_index])
    print('Target sentence:',target_sentence)
    print('Decoded sentence:','START_ '+decoded_sentence+' _END')
    x1=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(1, 0,
    print('Individual 1-gram: %f' % x1)
    x2=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0, 1,
    print('Individual 2-gram: %f' % x2)
    x3=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0, 0,
    print('Individual 3-gram: %f' % x3)
    x4=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0,0,0,
    print('Individual 4-gram: %f' % x4)
    score = sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0
    print('4-gram cummulative score: ',score)
    ip_seq.append(input_texts[seq_index])
    op_seq.append(target_sentence)
    dec_seq.append('START_ '+decoded_sentence+' _END')
    b1.append(x1)
    b2.append(x2)
    b3.append(x3)
    b4.append(x4)
    b_cum.append(score)
```

```
-
Input sentence: I've been away from them for far too long.
Target sentence: START_ मैं उनसे दूर अभी तक बहुत लंबे समय के लिए किया गया है। _E
ND
Decoded sentence: START_  मैं तुम्हें पता है कि लोगों को उसके लिए शोक _END
Individual 1-gram: 0.714681
Individual 2-gram: 0.464474
Individual 3-gram: 0.299614
Individual 4-gram: 0.208867
4-gram cummulative score:  0.37964405419136216
-
Input sentence: Hank, he tells me that he's found the answer to your cosme
tic problem.
Target sentence: START_ हांक .. वह मुझसे कहता है, वह अपने अंगराग समस्या का जवाब
मिल गया है. _END
Decoded sentence: START_  ठीक है , लेकिन आप सभी ने सुना है कि कि , _END
Individual 1-gram: 0.485203
Individual 2-gram: 0.259041
Individual 3-gram: 0.144066
Individual 4-gram: 0.085718
```

In [43]:

```
# n-gram individual BLEU
from nltk.translate.bleu_score import sentence_bleu
for seq_index in range(6900,7000):
    # Take one sequence (part of the training set)
    # for trying out decoding.
    input_seq = encoder_input_data[seq_index: seq_index + 1]
    target_sentence = target_texts[seq_index]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('Input sentence:', input_texts[seq_index])
    print('Target sentence:',target_sentence)
    print('Decoded sentence:','START_ '+decoded_sentence+' _END')
    x1=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(1, 0,
    print('Individual 1-gram: %f' % x1)
    x2=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0, 1,
    print('Individual 2-gram: %f' % x2)
    x3=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0, 0,
    print('Individual 3-gram: %f' % x3)
    x4=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0,0,0,
    print('Individual 4-gram: %f' % x4)
    score = sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(
    print('4-gram cummulative score: ',score)
    ip_seq.append(input_texts[seq_index])
    op_seq.append(target_sentence)
    dec_seq.append('START_ '+decoded_sentence+' _END')
    b1.append(x1)
    b2.append(x2)
    b3.append(x3)
    b4.append(x4)
    b_cum.append(score)
```

```
-
Input sentence: What the fuck?
Target sentence: START_ बकवास क्या? अरे यार! _END
Decoded sentence: START_  क्या तुम सच में करने के लिए क्या कर रहे हैं _END
Individual 1-gram: 0.464286
Individual 2-gram: 0.290909
Individual 3-gram: 0.203704
Individual 4-gram: 0.150943
4-gram cummulative score:  0.25385686603792507
-
Input sentence: It's over?
Target sentence: START_ यह खत्म हो गया है? _END
Decoded sentence: START_  यह एक अच्छा है . _END
Individual 1-gram: 0.732907
Individual 2-gram: 0.552060
Individual 3-gram: 0.429380
Individual 4-gram: 0.297263
4-gram cummulative score:  0.4767102796077954
-
```

In [44]:

```
df_bleu=pd.DataFrame()
df_bleu["ip_seq"]=ip_seq
df_bleu["op_seq"]=op_seq
df_bleu["dec_seq"]=dec_seq
df_bleu["bleu_1-gram"]=b1
df_bleu["bleu_2-gram"]=b2
df_bleu["bleu_3-gram"]=b3
df_bleu["bleu_4-gram"]=b4
df_bleu["bleu_cumm_4-gram"]=b_cum
```

In [45]:

```
df_bleu.to_csv('G:\\CSUEB\\MSBA\\Summer 19\\DL_BAN676\\Project\\LSTM_1_Layer_BLEU.csv',inde
```

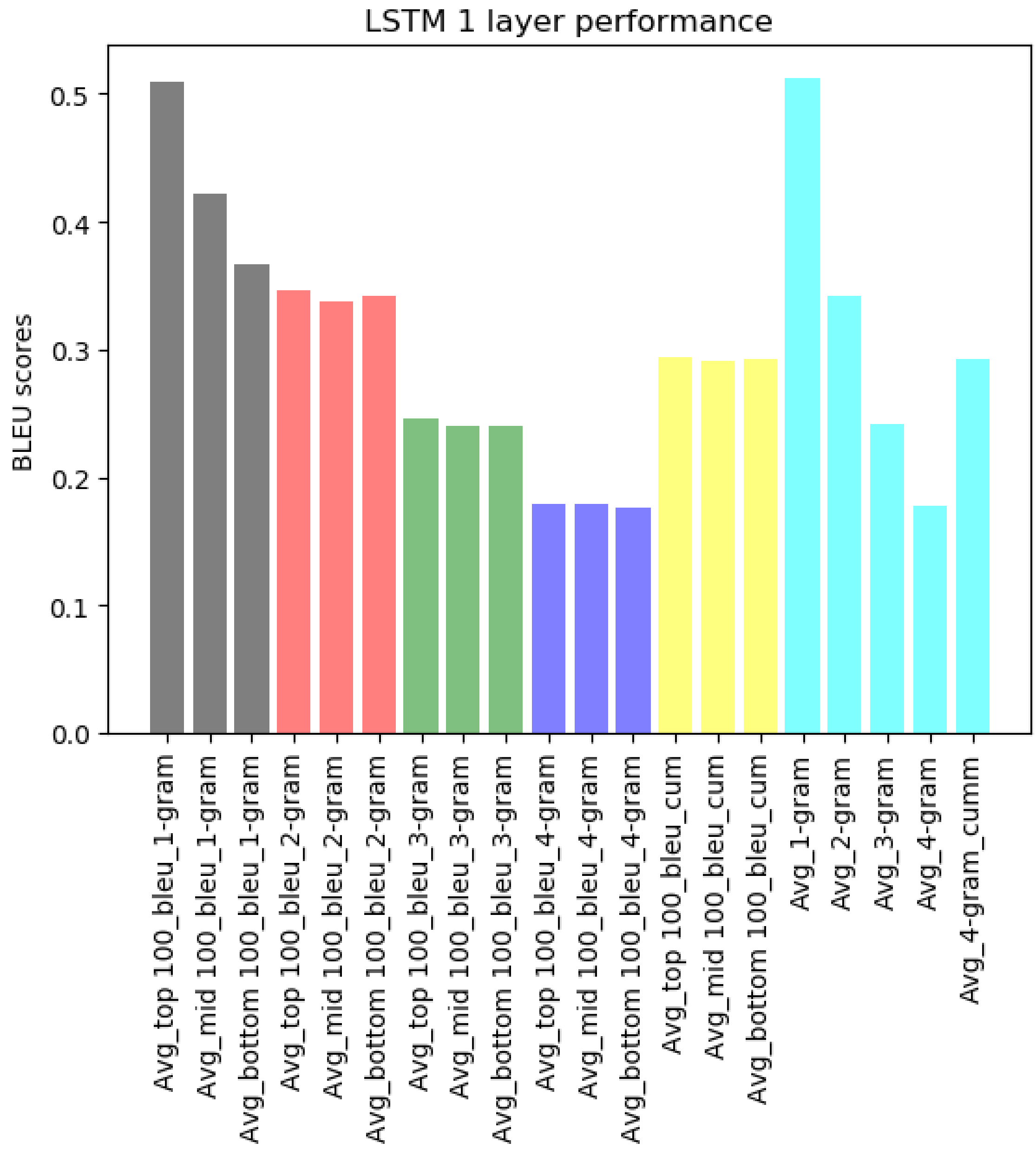
After editing the csv to reflect averages

In [47]:

```
df_bleu_compute=pd.read_csv('G:\\CSUEB\\MSBA\\Summer 19\\DL_BAN676\\Project\\LSTM_1_Layer_B
```

In [48]:

```
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = ('Avg_top 100_bleu_1-gram','Avg_mid 100_bleu_1-gram','Avg_bottom 100_bleu_1-gram'
y_pos = np.arange(len(objects))

performance = [df_bleu_compute['Avg_top 100_bleu_1-gram'].iloc[0],df_bleu_compute['Avg_mid
               df_bleu_compute['Avg_top 100_bleu_2-gram'].iloc[0],df_bleu_compute['Avg_mid
               df_bleu_compute['Avg_top 100_bleu_3-gram'].iloc[0],df_bleu_compute['Avg_mid
               df_bleu_compute['Avg_top 100_bleu_4-gram'].iloc[0],df_bleu_compute['Avg_mid
               df_bleu_compute['Avg_top 100_bleu_cum'].iloc[0],df_bleu_compute['Avg_mid 100
               df_bleu_compute['Avg_1-gram'].iloc[0],df_bleu_compute['Avg_2-gram'].iloc[0],

plt.bar(y_pos, performance, align='center', alpha=0.5, color=['black','black','black', 'rec
plt.xticks(y_pos, objects,rotation=90)
plt.ylabel('BLEU scores')
plt.title('LSTM 1 layer performance')

plt.show()
```



The INFERENCE for simple LSTM seq2 seq model saturates quite soon. So we will try to introduces changes to the model

---

Refining model

1. Adding additional LSTM encoder layer
2. Adding additional LSTM decoder layer
3. Adding Model checkpoint based on Validation loss

   ### ---

In [2]:

```
batch_size = 64  # Batch size for training.
epochs = 100  # Number of epochs to train for.
latent_dim = 512  # Latent dimensionality of the encoding space.
num_samples = 7000  # Number of samples to train on.
# Path to the data txt file on disk.
data_path = 'cleaned_data.txt'
```

In [3]:

```
# Vectorize the data.
input_texts = []
target_texts = []
input_words = set()
target_words = set()

with open(data_path, 'r', encoding='utf-8') as f:
    lines = f.read().split('\n')
for line in lines[: min(num_samples, len(lines) - 1)]:
    index, input_text, target_text = line.split('\t')
    # We use "tab" as the "start sequence" character
    # for the targets, and "\n" as "end sequence" character.
    target_text = 'START_ '+target_text+ ' _END'
    input_texts.append(input_text)
    target_texts.append(target_text)

    input_word_tokens=nltk.word_tokenize(input_text)
    target_word_tokens=nltk.word_tokenize(target_text)

    for word in input_word_tokens:
        if word not in input_words:
            input_words.add(word)
    for word in target_word_tokens:
        if word not in target_words:
            target_words.add(word)
#input_words.add('')
#target_words.add('')
input_words = sorted(list(input_words))

target_words = sorted(list(target_words))

num_encoder_tokens = len(input_words)
num_decoder_tokens = len(target_words)
max_encoder_seq_length = max([len(nltk.word_tokenize(txt)) for txt in input_texts])
max_decoder_seq_length = max([len(nltk.word_tokenize(txt)) for txt in target_texts])

print('Number of samples:', len(input_texts))
print('Number of unique input tokens:', num_encoder_tokens)
print('Number of unique output tokens:', num_decoder_tokens)
print('Max sequence length for inputs:', max_encoder_seq_length)
print('Max sequence length for outputs:', max_decoder_seq_length)
print('-------Word corpus-------')
#print(input_words)
#print(target_words)
```

```
Number of samples: 7000
Number of unique input tokens: 6567
Number of unique output tokens: 6463
Max sequence length for inputs: 43
Max sequence length for outputs: 43
-------Word corpus-------
```

In [4]:

```
input_token_index = dict(
    [(word, i) for i, word in enumerate(input_words)])
target_token_index = dict(
    [(word, i) for i, word in enumerate(target_words)])

encoder_input_data = np.zeros(
    (len(input_texts), max_encoder_seq_length, num_encoder_tokens),
    dtype='float16')
decoder_input_data = np.zeros(
    (len(input_texts), max_decoder_seq_length, num_decoder_tokens),
    dtype='float16')

decoder_target_data = np.zeros(
    (len(input_texts), max_decoder_seq_length, num_decoder_tokens),
    dtype='float16')

for i, (input_text, target_text) in enumerate(zip(input_texts, target_texts)):
    for t, word in enumerate(nltk.word_tokenize(input_text)):
        encoder_input_data[i, t, input_token_index[word]] = 1.

    for t, word in enumerate(nltk.word_tokenize(target_text)):
        # decoder_target_data is ahead of decoder_input_data by one timestep
        decoder_input_data[i, t, target_token_index[word]] = 1.
        if t > 0:
            # decoder_target_data will be ahead by one timestep
            # and will not include the start character.
            decoder_target_data[i, t - 1, target_token_index[word]] = 1.
```

In [5]:

```
from keras.models import Model
from keras.layers import Input, LSTM, Dense, RNN
#layers = [256,128] # we loop LSTMCells then wrap them in an RNN layer

#EARLY STOPPING
#early_stopping = EarlyStopping(monitor='val_acc', patience=25)

#MODEL CHECKPOINT
ckpt_file = 'complex_model.h1.28_jul_19'
checkpoint = ModelCheckpoint(ckpt_file, monitor='val_loss', verbose=1, save_best_only=True,

encoder_inputs = Input(shape=(None, num_encoder_tokens))

e_outputs, h1, c1 = LSTM(latent_dim, return_state=True, return_sequences=True)(encoder_inpu
_, h2, c2 = LSTM(latent_dim, return_state=True)(e_outputs)
encoder_states = [h1, c1, h2, c2]

decoder_inputs = Input(shape=(None, num_decoder_tokens))

out_layer1 = LSTM(latent_dim, return_sequences=True, return_state=True)
d_outputs, dh1, dc1 = out_layer1(decoder_inputs,initial_state= [h1, c1])
out_layer2 = LSTM(latent_dim, return_sequences=True, return_state=True)
final, dh2, dc2 = out_layer2(d_outputs, initial_state= [h2, c2])
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(final)


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

complex_model.summary()
```

```
WARNING: Logging before flag parsing goes to stderr.
W0728 10:32:23.673350 14164 deprecation_wrapper.py:119] From c:\users\robust
us\appdata\local\programs\python\python37\lib\site-packages\keras\backend\te
nsorflow_backend.py:74: The name tf.get_default_graph is deprecated. Please
use tf.compat.v1.get_default_graph instead.

W0728 10:32:23.694314 14164 deprecation_wrapper.py:119] From c:\users\robust
us\appdata\local\programs\python\python37\lib\site-packages\keras\backend\te
nsorflow_backend.py:517: The name tf.placeholder is deprecated. Please use t
f.compat.v1.placeholder instead.

W0728 10:32:23.699292 14164 deprecation_wrapper.py:119] From c:\users\robust
us\appdata\local\programs\python\python37\lib\site-packages\keras\backend\te
nsorflow_backend.py:4138: The name tf.random_uniform is deprecated. Please u
se tf.random.uniform instead.


____________________________________________________________________________
______________________
Layer (type)                    Output Shape         Param #     Connected t
o
============================================================================
======================
input_1 (InputLayer)            (None, None, 6567)   0
____________________________________________________________________________
______________________
input_2 (InputLayer)            (None, None, 6463)   0
____________________________________________________________________________
```

```
______________________
lstm_1 (LSTM)                   [(None, None, 512),  14499840    input_1[0]
[0]
____________________________________________________________________________
______________________
lstm_3 (LSTM)                   [(None, None, 512),  14286848    input_2[0]
[0]
                                                                 lstm_1[0]
[1]
                                                                 lstm_1[0]
[2]
____________________________________________________________________________
______________________
lstm_2 (LSTM)                   [(None, 512), (None, 2099200     lstm_1[0]
[0]
____________________________________________________________________________
______________________
lstm_4 (LSTM)                   [(None, None, 512),  2099200     lstm_3[0]
[0]
                                                                 lstm_2[0]
[1]
                                                                 lstm_2[0]
[2]
____________________________________________________________________________
______________________
dense_1 (Dense)                 (None, None, 6463)   3315519     lstm_4[0]
[0]
============================================================================
======================
Total params: 36,300,607
Trainable params: 36,300,607
Non-trainable params: 0
____________________________________________________________________________
______________________
```

In [6]:

```
# Run training
complex_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
complex_model.summary()
```

```
W0728 10:32:26.275163 14164 deprecation_wrapper.py:119] From c:\users\robust
us\appdata\local\programs\python\python37\lib\site-packages\keras\optimizer
s.py:790: The name tf.train.Optimizer is deprecated. Please use tf.compat.v
1.train.Optimizer instead.

W0728 10:32:26.295082 14164 deprecation_wrapper.py:119] From c:\users\robust
us\appdata\local\programs\python\python37\lib\site-packages\keras\backend\te
nsorflow_backend.py:3295: The name tf.log is deprecated. Please use tf.math.
log instead.



____________________________________________________________________________
______________________
Layer (type)                    Output Shape         Param #     Connected t
o
============================================================================
======================
input_1 (InputLayer)            (None, None, 6567)   0
____________________________________________________________________________
______________________
input_2 (InputLayer)            (None, None, 6463)   0
____________________________________________________________________________
______________________
lstm_1 (LSTM)                   [(None, None, 512),  14499840    input_1[0]
[0]
____________________________________________________________________________
______________________
lstm_3 (LSTM)                   [(None, None, 512),  14286848    input_2[0]
[0]
                                                                 lstm_1[0]
[1]
                                                                 lstm_1[0]
[2]
____________________________________________________________________________
______________________
lstm_2 (LSTM)                   [(None, 512), (None, 2099200     lstm_1[0]
[0]
____________________________________________________________________________
______________________
lstm_4 (LSTM)                   [(None, None, 512),  2099200     lstm_3[0]
[0]
                                                                 lstm_2[0]
[1]
                                                                 lstm_2[0]
[2]
____________________________________________________________________________
______________________
dense_1 (Dense)                 (None, None, 6463)   3315519     lstm_4[0]
[0]
============================================================================
======================
Total params: 36,300,607
Trainable params: 36,300,607
Non-trainable params: 0
```

Running complex model with dual encoder and decoder layers and 512 latent_dims and batch size 64 for 100 epochs

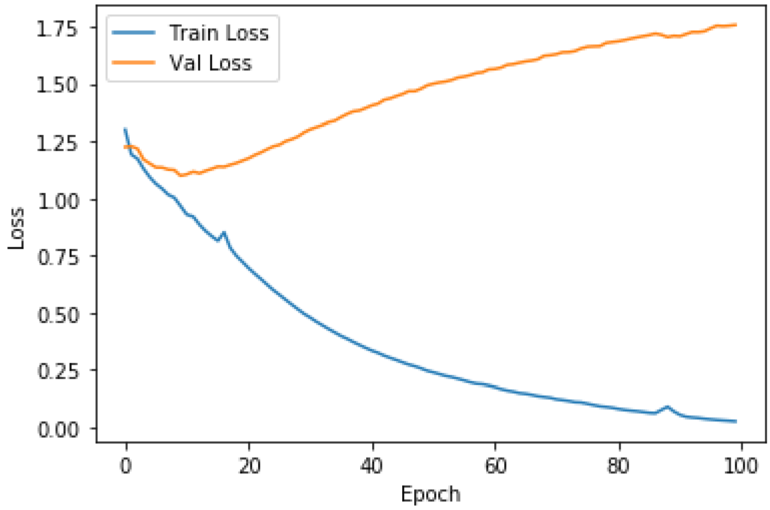
In [7]:

```
complex_history=complex_model.fit([encoder_input_data, decoder_input_data], decoder_target_
#model.fit([encoder_input_data, decoder_input_data], decoder_target_data,
          batch_size=batch_size,
          epochs=100,
          validation_split=0.2, callbacks=[checkpoint], verbose=1)
# Save model
#model.save('Ass3_w2w_s2s_2000_complex.h5')
complex_model.save('Proj_w2w_complex512_s2s_64b_100e_complex_rerun.h5')
```

```
5600/5600 [==============================] - 107s 19ms/step - loss: 0.0327
- acc: 0.2107 - val_loss: 1.7530 - val_acc: 0.0569

Epoch 00097: val_loss did not improve from 1.09888
Epoch 98/100
5600/5600 [==============================] - 107s 19ms/step - loss: 0.0307
- acc: 0.2107 - val_loss: 1.7510 - val_acc: 0.0543

Epoch 00098: val_loss did not improve from 1.09888
Epoch 99/100
5600/5600 [==============================] - 108s 19ms/step - loss: 0.0288
- acc: 0.2111 - val_loss: 1.7530 - val_acc: 0.0542

Epoch 00099: val_loss did not improve from 1.09888
Epoch 100/100
5600/5600 [==============================] - 107s 19ms/step - loss: 0.0267
- acc: 0.2112 - val_loss: 1.7561 - val_acc: 0.0580

Epoch 00100: val_loss did not improve from 1.09888
```

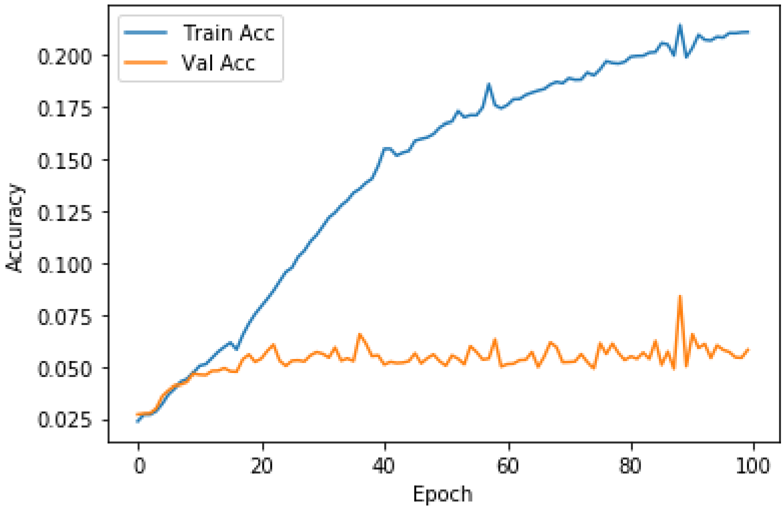
In [8]:

```
import matplotlib.pyplot as plt
def plot_loss_history(complex_history):
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.plot(complex_history.epoch, np.array(complex_history.history['loss']),
               label='Train Loss')
    plt.plot(complex_history.epoch, np.array(complex_history.history['val_loss']),
           label = 'Val Loss')
    plt.legend()
    #plt.ylim([0.05, 1])

plot_loss_history(complex_history)
```



In [9]:

```
import matplotlib.pyplot as plt
def plot_loss_history(complex_history):
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.plot(complex_history.epoch, np.array(complex_history.history['acc']),
             label='Train Acc')
    plt.plot(complex_history.epoch, np.array(complex_history.history['val_acc']),
          label = 'Val Acc')
    plt.legend()
    #plt.ylim([0.05, 1])

plot_loss_history(complex_history)
```



In [ ]:

In [ ]:

Run only when recalling new model after restarting kernel¶

When kernel crashes and for retraining with just 1 epoch

In [7]:

```
import tensorflow as tf
#Call a saved model
#resume_complex_model = tf.keras.models.load_model('Ass3_s2s_2000_complex.h5')
resume_complex_model = tf.keras.models.load_model('Ass3_w2w_s2s_250_complex.h5')

resume_complex_model.summary()

#to reinstate the model, running for just one epoch
resume_complex_history=resume_complex_model.fit([encoder_input_data, decoder_input_data], d
          batch_size=batch_size,
          epochs=1,
          validation_split=0.2)
# Save model
#new_model.save('revised_Ass3_s2s_200_with_space.h5')
```

```
W0721 20:09:49.464467  8816 deprecation.py:323] From c:\users\robustus\appda
ta\local\programs\python\python37\lib\site-packages\tensorflow\python\ops\ma
th_grad.py:1250: add_dispatch_support.<locals>.wrapper (from tensorflow.pyth
on.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where

Model: "model_1"
____________________________________________________________________________
______________________
Layer (type)                    Output Shape         Param #     Connected t
o
============================================================================
======================
input_1 (InputLayer)            [(None, None, 2568)] 0
____________________________________________________________________________
______________________
input_2 (InputLayer)            [(None, None, 3025)] 0
____________________________________________________________________________
______________________
lstm_1 (LSTM)                   [(None, None, 512),  6309888     input_1[0]
[0]
____________________________________________________________________________
______________________
lstm_3 (LSTM)                   [(None, None, 512),  7245824     input_2[0]
[0]
                                                                 lstm_1[0]
[1]
                                                                 lstm_1[0]
[2]
____________________________________________________________________________
______________________
lstm_2 (LSTM)                   [(None, 512), (None, 2099200     lstm_1[0]
[0]
____________________________________________________________________________
______________________
lstm_4 (LSTM)                   [(None, None, 512),  2099200     lstm_3[0]
[0]
                                                                 lstm_2[0]
[1]
                                                                 lstm_2[0]
[2]
____________________________________________________________________________
______________________
```

```
dense_1 (Dense)                 (None, None, 3025)   1551825     lstm_4[0]
[0]
============================================================================
======================
Total params: 19,305,937
Trainable params: 19,305,937
Non-trainable params: 0

____________________________________________________________________________
______________________
Train on 2248 samples, validate on 562 samples
2248/2248 [==============================] - 17s 8ms/sample - loss: 0.0096 -
acc: 0.2453 - val_loss: 3.7231 - val_acc: 0.0681
```

In [ ]:

```
import matplotlib.pyplot as plt
def plot_loss_history(resume_complex_history):
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.plot(resume_complex_history.epoch, np.array(resume_complex_history.history['loss'])
               label='Train Loss')
    plt.plot(resume_complex_history.epoch, np.array(resume_complex_history.history['val_los
           label = 'Val Loss')
    plt.legend()
    #plt.ylim([0.05, 1])

plot_loss_history(resume_complex_history)
```

In [ ]:

```
import matplotlib.pyplot as plt
def plot_loss_history(resume_complex_history):
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Acc')
    plt.plot(resume_complex_history.epoch, np.array(resume_complex_history.history['acc']),
               label='Train Acc')
    plt.plot(resume_complex_history.epoch, np.array(resume_complex_history.history['val_acc
           label = 'Val Acc')
    plt.legend()
    #plt.ylim([0.05, 1])

plot_loss_history(resume_complex_history)
```

Complex Model Inference

In [10]:

```
# Next: inference mode (sampling).
# Here's the drill:
# 1) encode input and retrieve initial decoder state
# 2) run one step of decoder with this initial state
# and a "start of sequence" token as target.
# Output will be the next target token
# 3) Repeat with the current target token and current states

# Define sampling models (modified for n-layer deep network)
encoder_model = Model(encoder_inputs, encoder_states)

decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_state_input_h1 = Input(shape=(latent_dim,))
decoder_state_input_c1 = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c,
                         decoder_state_input_h1, decoder_state_input_c1]
d_o, state_h, state_c = out_layer1(
    decoder_inputs, initial_state=decoder_states_inputs[:2])
d_o, state_h1, state_c1 = out_layer2(
    d_o, initial_state=decoder_states_inputs[-2:])
decoder_states = [state_h, state_c, state_h1, state_c1]
decoder_outputs = decoder_dense(d_o)
decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs] + decoder_states)

decoder_model.summary()
```

```
____________________________________________________________________________
______________________
Layer (type)                    Output Shape         Param #     Connected t
o
============================================================================
======================
input_2 (InputLayer)            (None, None, 6463)   0
____________________________________________________________________________
______________________
input_3 (InputLayer)            (None, 512)          0
____________________________________________________________________________
______________________
input_4 (InputLayer)            (None, 512)          0
____________________________________________________________________________
______________________
lstm_3 (LSTM)                   [(None, None, 512),  14286848    input_2[0]
[0]
                                                                 input_3[0]
[0]
                                                                 input_4[0]
[0]
____________________________________________________________________________
______________________
input_5 (InputLayer)            (None, 512)          0
____________________________________________________________________________
______________________
input_6 (InputLayer)            (None, 512)          0
____________________________________________________________________________
______________________
lstm_4 (LSTM)                   [(None, None, 512),  2099200     lstm_3[1]
```

```
[0]
                                                                 input_5[0]
[0]
                                                                 input_6[0]
[0]
____________________________________________________________________________
______________________
dense_1 (Dense)                 (None, None, 6463)   3315519     lstm_4[1]
[0]
============================================================================
======================
Total params: 19,701,567
Trainable params: 19,701,567
Non-trainable params: 0
____________________________________________________________________________
______________________
```

In [ ]:

In [11]:

```python
# Reverse-lookup token index to decode sequences back to
# something readable.
reverse_input_word_index = dict(
    (i, word) for word, i in input_token_index.items())
reverse_target_word_index = dict(
    (i, word) for word, i in target_token_index.items())
```

In [12]:

```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    states_value = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1, 1, num_decoder_tokens))
    # Populate the first character of target sequence with the start character.
    target_seq[0, 0, target_token_index['START_']] = 1.

    # Sampling loop for a batch of sequences
    # (to simplify, here we assume a batch of size 1).
    stop_condition = False
    decoded_sentence = ''
    while stop_condition == False:
        output_tokens, h, c, h1, c1 = decoder_model.predict(
            [target_seq] + states_value) #######NOTICE THE ADDITIONAL HIDDEN STATES

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_word = reverse_target_word_index[sampled_token_index]
        if (sampled_word != '_END'):
            decoded_sentence += ' '+sampled_word

        # Exit condition: either hit max length
        # or find stop character.
        elif (sampled_word == '_END' or len(decoded_sentence) > max_decoder_seq_length):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1, 1, num_decoder_tokens))
        target_seq[0, 0, sampled_token_index] = 1.

        # Update states
        states_value = [h, c, h1, c1]#######NOTICE THE ADDITIONAL HIDDEN STATES

    return decoded_sentence
```

In [13]:

```
for seq_index in range(20):
    # Take one sequence (part of the training set)
    # for trying out decoding.
    input_seq = encoder_input_data[seq_index: seq_index + 1]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('Input sentence:', input_texts[seq_index])
    print('Target sentence:', target_texts[seq_index])
    print('Decoded sentence:', decoded_sentence)
```

```
-
Input sentence: I do not want to die.
Target sentence: START_ मैं मरना नहीं चाहता. _END
Decoded sentence:  मैं मरना नहीं चाहता .
-
Input sentence: It's the same country I think.
Target sentence: START_ यह मुझे लगता है कि एक ही देश है. _END
Decoded sentence:  यह मुझे लगता है कि एक ही देश है .
-
Input sentence: Then they'll be crying like babies.
Target sentence: START_ फिर ये नन्हें बच्चों की तरह रोएँगे। _END
Decoded sentence:  फिर ये नन्हें बच्चों की तरह रोएँगे।
-
Input sentence: - No, I need power up!
Target sentence: START_ नहीं, मुझे पावर की जरुरत है ! _END
Decoded sentence:  नहीं , मैं नहीं होगा .
-
Input sentence: I will not eat him.
Target sentence: START_ मैं उसे नहीं खा जाएगा. _END
Decoded sentence:  मैं उसे नहीं खा जाएगा .
-
Input sentence: You gotta get me to Charleston.
Target sentence: START_ आप चार्ल्सटन करने के लिए मुझे जाना होगा. _END
Decoded sentence:  आप चार्ल्सटन करने के लिए मुझे जाना होगा .
-
Input sentence: - NO, HE'S NOT MY DAD.
Target sentence: START_ - नहीं, वह मेरे पिता नहीं है. _END
Decoded sentence:  - नहीं , वह मेरे पिता नहीं है .
-
Input sentence: I told her we rest on Sundays.
Target sentence: START_ मैं रविवार को उसे हम बाकी बताया. _END
Decoded sentence:  मैं रविवार को उसे हम बाकी बताया .
-
Input sentence: You could've at least informed me, right?
Target sentence: START_ तुम्हें कम से कम मुझे तो बताना चाहिए था,ना? _END
Decoded sentence:  तुम्हें कम से कम मुझे तो बताना चाहिए था , ना ?
-
Input sentence: Your little bitch says you're gonna put me in jail!
Target sentence: START_ तेरी कमीनी कहती है कि वो मुझे जेल भेजेगी ! _END
Decoded sentence:  तेरी कमीनी कहती है कि वो मुझे जेल भेजेगी !
-
Input sentence: - You can call me whatever you like.
Target sentence: START_ - तुम मुझे फोन कर सकते हैं जो कुछ भी आप की तरह। _END
Decoded sentence:  - तुम मुझे फोन कर सकते हैं जो कुछ भी आप की तरह।
-
Input sentence: - You don't just kill a guy like that!
Target sentence: START_ - तुम बस की तरह है कि एक आदमी को मार नहीं है! _END
Decoded sentence:  - तुम बस की तरह है कि एक आदमी को मार नहीं है !
-
```

```
Input sentence: You sent these?
Target sentence: START_ आप इन भेजा? _END
Decoded sentence:  आप इन भेजा ?
-
Input sentence: I really loved him.
Target sentence: START_ मैं वास्तव में उसे प्यार करता था। _END
Decoded sentence:  मैं वास्तव में उसे प्यार करता था।
-
Input sentence: I ain't much at guessing games.
Target sentence: START_ मैं अनुमान लगाने के खेल में ज्यादा नहीं है. _END
Decoded sentence:  मैं अनुमान लगाने के खेल में ज्यादा नहीं है .
-
Input sentence: You're sick and I can help you.
Target sentence: START_ तुम बीमार हो और मैं तुम्हारी मदद कर सकते हैं। _END
Decoded sentence:  तुम बीमार हो और मैं तुम्हारी मदद कर सकते हैं।
-
Input sentence: Mike, do I get to ride with you?
Target sentence: START_ माइक, मैं आप के साथ सवारी करने के लिए मिलता है? _END
Decoded sentence:  माइक , मैं आप के साथ सवारी करने के लिए मिलता है ?
-
Input sentence: What do you fucking think?
Target sentence: START_ आपको क्या लगता है कि बकवास है? _END
Decoded sentence:  क्या मैं यह पता है ?
-
Input sentence: I know that woman you love also is ready to forgive you.
Target sentence: START_ मैं आप उसे माफ करने के लिए तैयार भी प्यार औरत को जानते हैं.
_END
Decoded sentence:  मैं आप किसी भी विचार है देखते हैं कि कितने चींटियों ?
-
Input sentence: Don't do it, man.
Target sentence: START_ , आदमी ऐसा मत करो. _END
Decoded sentence:  एक बार जो मुझे लगता है कि वह भी काम ले लिया ...
```

Introducing BLEU score metric at following levels:

(Individual 1-gram, 2-gram, 3-gram, 4-gram as well as cumulative 4-gram)

1. Top 100 samples

2. middle 100 samples

3. last 100 samples

In [21]:

```
ip_seq=[]
op_seq=[]
dec_seq=[]
b1=[]
b2=[]
b3=[]
b4=[]
b_cum=[]
```

In [22]:

```
# n-gram individual BLEU
from nltk.translate.bleu_score import sentence_bleu
for seq_index in range(100):
    # Take one sequence (part of the training set)
    # for trying out decoding.
    input_seq = encoder_input_data[seq_index: seq_index + 1]
    target_sentence = target_texts[seq_index]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('Input sentence:', input_texts[seq_index])
    print('Target sentence:',target_sentence)
    print('Decoded sentence:','START_ '+decoded_sentence+' _END')
    x1=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(1, 0,
    print('Individual 1-gram: %f' % x1)
    x2=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0, 1,
    print('Individual 2-gram: %f' % x2)
    x3=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0, 0,
    print('Individual 3-gram: %f' % x3)
    x4=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0,0,0,
    print('Individual 4-gram: %f' % x4)
    score = sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(
    print('4-gram cummulative score: ',score)
    ip_seq.append(input_texts[seq_index])
    op_seq.append(target_sentence)
    dec_seq.append('START_ '+decoded_sentence+' _END')
    b1.append(x1)
    b2.append(x2)
    b3.append(x3)
    b4.append(x4)
    b_cum.append(score)
```

```
-
Input sentence: It's the same country I think.
Target sentence: START_ यह मुझे लगता है कि एक ही देश है. _END
Decoded sentence: START_  यह मुझे लगता है कि एक ही देश है . _END
Individual 1-gram: 0.956522
Individual 2-gram: 0.933333
Individual 3-gram: 0.886364
Individual 4-gram: 0.837209
4-gram cummulative score:  0.9021825013122124
-
Input sentence: Then they'll be crying like babies.
Target sentence: START_ फिर ये नन्हें बच्चों की तरह रोएँगे। _END
Decoded sentence: START_  फिर ये नन्हें बच्चों की तरह रोएँगे। _END
Individual 1-gram: 0.979167
Individual 2-gram: 0.978723
Individual 3-gram: 0.956522
Individual 4-gram: 0.933333
4-gram cummulative score:  0.961749687653068
-
Input sentence: - No, I need power up!
```

In [23]:

```
# n-gram individual BLEU
from nltk.translate.bleu_score import sentence_bleu
for seq_index in range(3450,3550):
    # Take one sequence (part of the training set)
    # for trying out decoding.
    input_seq = encoder_input_data[seq_index: seq_index + 1]
    target_sentence = target_texts[seq_index]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('Input sentence:', input_texts[seq_index])
    print('Target sentence:',target_sentence)
    print('Decoded sentence:','START_ '+decoded_sentence+' _END')
    x1=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(1, 0,
    print('Individual 1-gram: %f' % x1)
    x2=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0, 1,
    print('Individual 2-gram: %f' % x2)
    x3=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0, 0,
    print('Individual 3-gram: %f' % x3)
    x4=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0,0,0,
    print('Individual 4-gram: %f' % x4)
    score = sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(
    print('4-gram cummulative score: ',score)
    ip_seq.append(input_texts[seq_index])
    op_seq.append(target_sentence)
    dec_seq.append('START_ '+decoded_sentence+' _END')
    b1.append(x1)
    b2.append(x2)
    b3.append(x3)
    b4.append(x4)
    b_cum.append(score)
```

```
-
Input sentence: I've been away from them for far too long.
Target sentence: START_ मैं उनसे दूर अभी तक बहुत लंबे समय के लिए किया गया है। _E
ND
Decoded sentence: START_  मैं कई एक आदमी को खो दिया . _END
Individual 1-gram: 0.441591
Individual 2-gram: 0.301942
Individual 3-gram: 0.183116
Individual 4-gram: 0.130199
4-gram cummulative score:  0.23744831369286126
-
Input sentence: Hank, he tells me that he's found the answer to your cosme
tic problem.
Target sentence: START_ हांक .. वह मुझसे कहता है, वह अपने अंगराग समस्या का जवाब
मिल गया है. _END
Decoded sentence: START_  हांक .. वह मुझसे कहता है , वह अपने अंगराग समस्या का
जवाब मिल गया है . _END
Individual 1-gram: 0.963415
Individual 2-gram: 0.938272
```

In [24]:

```
# n-gram individual BLEU
from nltk.translate.bleu_score import sentence_bleu
for seq_index in range(6900,7000):
    # Take one sequence (part of the training set)
    # for trying out decoding.
    input_seq = encoder_input_data[seq_index: seq_index + 1]
    target_sentence = target_texts[seq_index]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('Input sentence:', input_texts[seq_index])
    print('Target sentence:',target_sentence)
    print('Decoded sentence:','START_ '+decoded_sentence+' _END')
    x1=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(1, 0,
    print('Individual 1-gram: %f' % x1)
    x2=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0, 1,
    print('Individual 2-gram: %f' % x2)
    x3=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0, 0,
    print('Individual 3-gram: %f' % x3)
    x4=sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(0,0,0,
    print('Individual 4-gram: %f' % x4)
    score = sentence_bleu([target_sentence], 'START_ '+decoded_sentence+' _END', weights=(
    print('4-gram cummulative score: ',score)
    ip_seq.append(input_texts[seq_index])
    op_seq.append(target_sentence)
    dec_seq.append('START_ '+decoded_sentence+' _END')
    b1.append(x1)
    b2.append(x2)
    b3.append(x3)
    b4.append(x4)
    b_cum.append(score)
```

```
-
Input sentence: What the fuck?
Target sentence: START_ बकवास क्या? अरे यार! _END
Decoded sentence: START_  हम सही बात है , लेकिन वह जिम्मेदारी अपने कहीं और रात
करता है कि उन्हें अपने देश यात्रा , बचने ... और आप एक अजीब कर ली . _END
Individual 1-gram: 0.219697
Individual 2-gram: 0.129771
Individual 3-gram: 0.069231
Individual 4-gram: 0.046512
4-gram cummulative score:  0.09788487654303055
-
Input sentence: It's over?
Target sentence: START_ यह खत्म हो गया है? _END
Decoded sentence: START_  क्या नहीं है ? _END
Individual 1-gram: 0.762271
Individual 2-gram: 0.516253
Individual 3-gram: 0.393729
Individual 4-gram: 0.260995
4-gram cummulative score:  0.44843595037925266
```

In [28]:

```
df_bleu=pd.DataFrame()
df_bleu["ip_seq"]=ip_seq
df_bleu["op_seq"]=op_seq
df_bleu["dec_seq"]=dec_seq
df_bleu["bleu_1-gram"]=b1
df_bleu["bleu_2-gram"]=b2
df_bleu["bleu_3-gram"]=b3
df_bleu["bleu_4-gram"]=b4
df_bleu["bleu_cumm_4-gram"]=b_cum
```

In [29]:

```
df_bleu.to_csv('G:\\CSUEB\\MSBA\\Summer 19\\DL_BAN676\\Project\\LSTM_2_Layer_BLEU.csv',inde
```

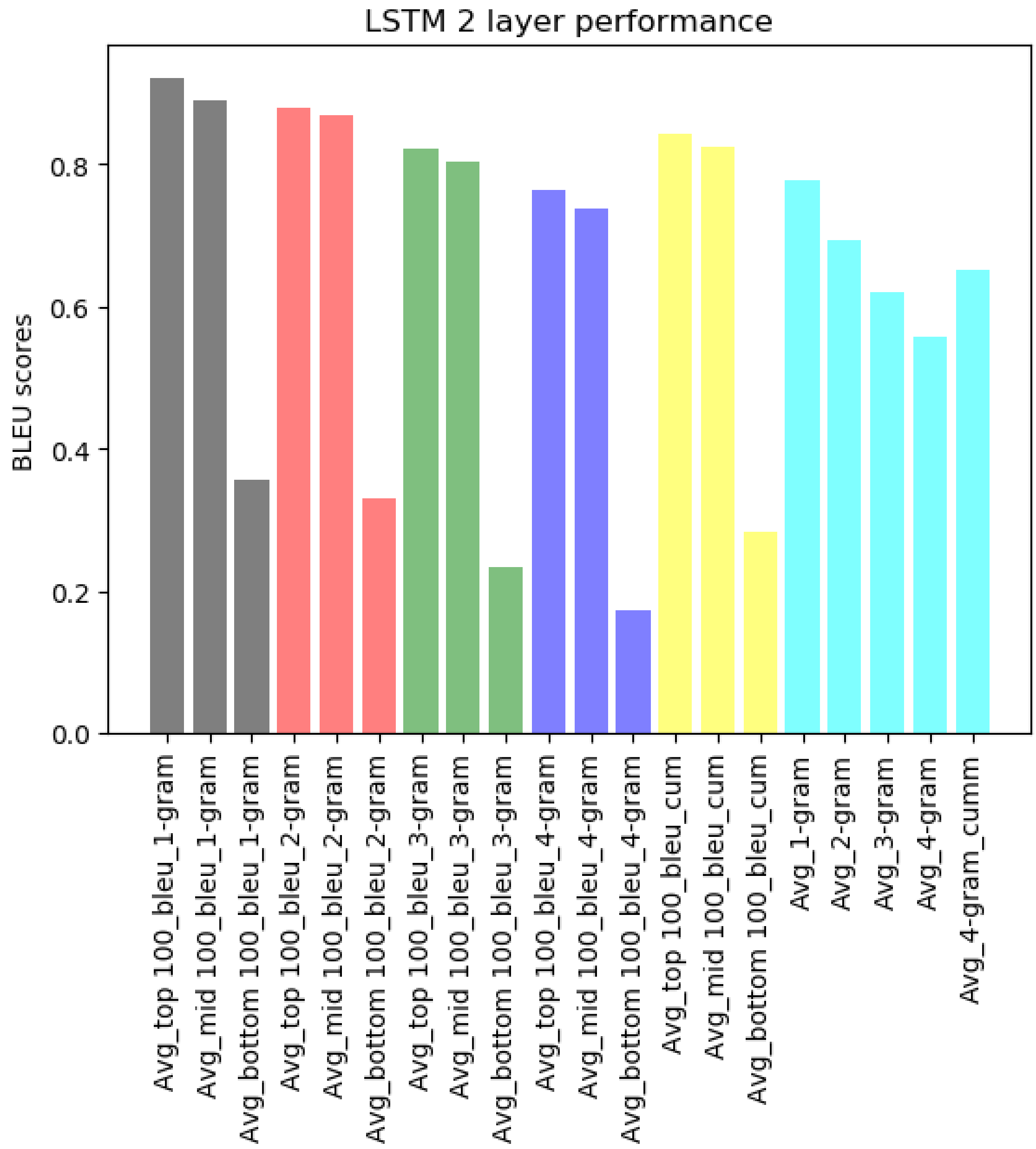
After editing the csv to reflect average values

In [8]:

```
df_bleu_compute=pd.read_csv('G:\\CSUEB\\MSBA\\Summer 19\\DL_BAN676\\Project\\LSTM_2_Layer_B
```

In [19]:

```
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

objects = ('Avg_top 100_bleu_1-gram','Avg_mid 100_bleu_1-gram','Avg_bottom 100_bleu_1-gram'
y_pos = np.arange(len(objects))

performance = [df_bleu_compute['Avg_top 100_bleu_1-gram'].iloc[0],df_bleu_compute['Avg_mid
               df_bleu_compute['Avg_top 100_bleu_2-gram'].iloc[0],df_bleu_compute['Avg_mid
               df_bleu_compute['Avg_top 100_bleu_3-gram'].iloc[0],df_bleu_compute['Avg_mid
               df_bleu_compute['Avg_top 100_bleu_4-gram'].iloc[0],df_bleu_compute['Avg_mid
               df_bleu_compute['Avg_top 100_bleu_cum'].iloc[0],df_bleu_compute['Avg_mid 10
               df_bleu_compute['Avg_1-gram'].iloc[0],df_bleu_compute['Avg_2-gram'].iloc[0],

plt.bar(y_pos, performance, align='center', alpha=0.5, color=['black','black','black', 'rec
plt.xticks(y_pos, objects,rotation=90)
plt.ylabel('BLEU scores')
plt.title('LSTM 2 layer performance')

plt.show()
```


LSTM 2 layer performance
BLEU scores
Avg_top 100_bleu_1-gram
Avg_mid 100_bleu_1-gram
Avg_bottom 100_bleu_1-gram
Avg_top 100_bleu_2-gram
Avg_mid 100_bleu_2-gram
Avg_bottom 100_bleu_2-gram
Avg_top 100_bleu_3-gram
Avg_mid 100_bleu_3-gram
Avg_bottom 100_bleu_3-gram
Avg_top 100_bleu_4-gram
Avg_mid 100_bleu_4-gram
Avg_bottom 100_bleu_4-gram
Avg_top 100_bleu_cum
Avg_mid 100_bleu_cum
Avg_bottom 100_bleu_cum
Avg_1-gram
Avg_2-gram
Avg_3-gram
Avg_4-gram
Avg_4-gram_cumm