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 [2]:

```
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
from keras import optimizers
from keras.layers import Dropout
import numpy as np
import pandas as pd
import nltk
import matplotlib.pyplot as plt
pd.set_option('display.max_columns', None)
```

```
Using TensorFlow backend.
```

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

In [2]:

```
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 [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-------
```

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 [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 [ ]:

```

```

Simple Word to Word Model

Encode-Decoder Model.

In [5]:

```
#EARLY STOPPING
#early_stopping = EarlyStopping(monitor='val_loss', patience=25)
#MODEL CHECKPOINT
ckpt_file = 'sgd_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 21:45:10.390530 15352 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 21:45:10.433416 15352 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 21:45:10.445384 15352 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 [6]:

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

model.summary()
```

```
W0728 21:45:11.952355 15352 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 21:45:11.973335 15352 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?

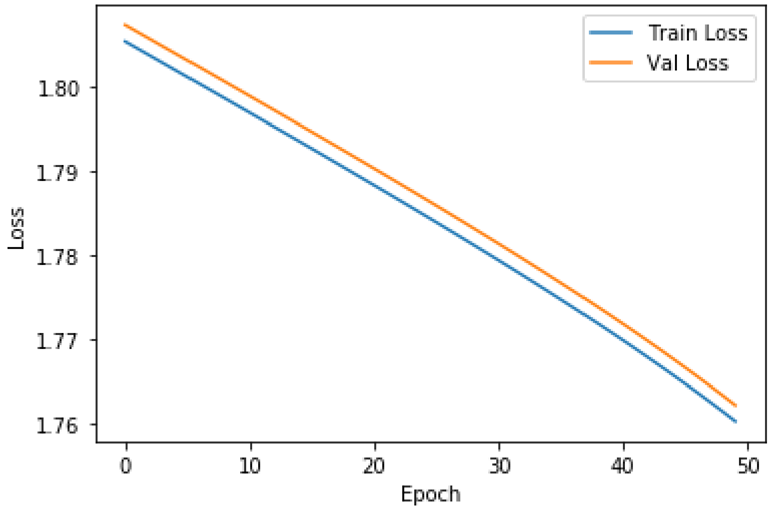
In [7]:

```
'''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_sgd_7000_w2w_s2s_64_512_50e.h5')
```

```
W0728 21:45:12.112825 15352 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 21:45:13.115144 15352 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: 1.8055
- acc: 0.0127 - val_loss: 1.8074 - val_acc: 0.0232

Epoch 00001: val_loss improved from inf to 1.80743, saving model to sgd_mo
del_h1_27_jul_19
```

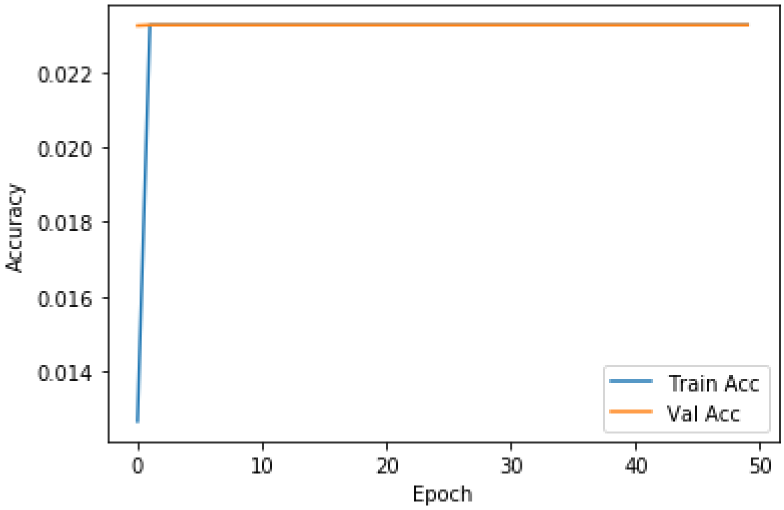
In [8]:

```
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 [9]:

```
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]:

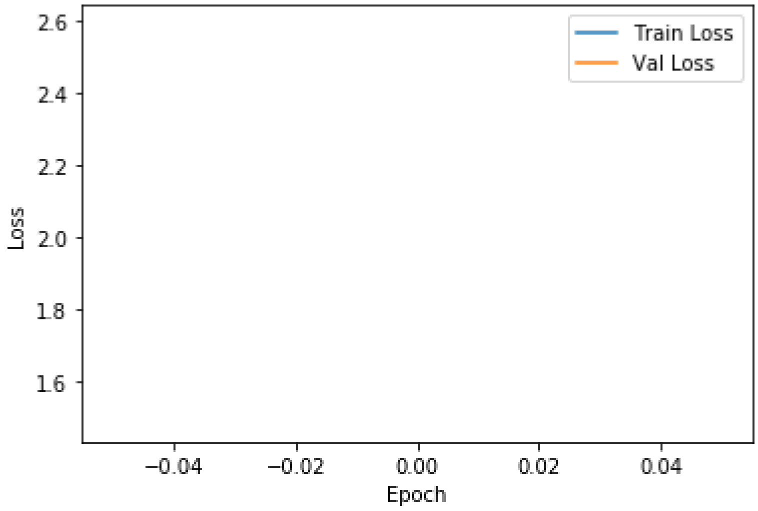
```
import tensorflow as tf
#Call a saved model
#tf.reset_default_graph()
from tensorflow.core.protobuf import rewriter_config_pb2
from tensorflow.keras.backend import set_session
tf.keras.backend.clear_session()  # For easy reset of notebook state.

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 [14]:

```
# 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 [15]:

```
# 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 [16]:

```
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 [17]:

```
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_  _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_  _END
-
Input sentence: Then they'll be crying like babies.
Target sentence: START_ फिर ये नन्हें बच्चों की तरह रोएँगे। _END
Decoded sentence: START_  _END
-
Input sentence: - No, I need power up!
Target sentence: START_ नहीं, मुझे पावर की जरुरत है ! _END
Decoded sentence: START_  _END
-
Input sentence: I will not eat him.
Target sentence: START_ मैं उसे नहीं खा जाएगा. _END
Decoded sentence: START_  _END
-
Input sentence: You gotta get me to Charleston.
Target sentence: START_ आप चार्ल्सटन करने के लिए मुझे जाना होगा. _END
Decoded sentence: START_  _END
-
Input sentence: - NO, HE'S NOT MY DAD.
Target sentence: START_ - नहीं, वह मेरे पिता नहीं है. _END
Decoded sentence: START_  _END
-
Input sentence: I told her we rest on Sundays.
Target sentence: START_ मैं रविवार को उसे हम बाकी बताया. _END
Decoded sentence: START_  _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_  _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_  _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_  _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_  _END
-
```

```
Input sentence: You sent these?
Target sentence: START_ आप इन भेजा? _END
Decoded sentence: START_  _END
-
Input sentence: I really loved him.
Target sentence: START_ मैं वास्तव में उसे प्यार करता था। _END
Decoded sentence: START_  _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_  _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_  _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_  _END
-
Input sentence: What do you fucking think?
Target sentence: START_ आपको क्या लगता है कि बकवास है? _END
Decoded sentence: START_  _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_  _END
-
Input sentence: Don't do it, man.
Target sentence: START_ , आदमी ऐसा मत करो. _END
Decoded sentence: START_  _END
```

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 [18]:

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

In [ ]:

```
```

In [19]:

```
# 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_  _END
Individual 1-gram: 0.188876
Individual 2-gram: 0.171705
Individual 3-gram: 0.151100
Individual 4-gram: 0.125917
4-gram cummulative score:  0.15760767926048347
-
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_  _END
Individual 1-gram: 0.069483
Individual 2-gram: 0.063167
Individual 3-gram: 0.055587
Individual 4-gram: 0.046322
4-gram cummulative score:  0.05798062497067458
-
Input sentence: Then they'll be crying like babies.
```

In [20]:

```
# 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_  _END
Individual 1-gram: 0.012074
Individual 2-gram: 0.010977
Individual 3-gram: 0.009660
Individual 4-gram: 0.008050
4-gram cummulative score:  0.010075521844875467
-
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_  _END
Individual 1-gram: 0.003760
Individual 2-gram: 0.003418
Individual 3-gram: 0.003008
Individual 4-gram: 0.002507
```

In [21]:

```
# 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_  _END
Individual 1-gram: 0.188876
Individual 2-gram: 0.171705
Individual 3-gram: 0.151100
Individual 4-gram: 0.125917
4-gram cummulative score:  0.15760767926048347
-
Input sentence: It's over?
Target sentence: START_ यह खत्म हो गया है? _END
Decoded sentence: START_  _END
Individual 1-gram: 0.223130
Individual 2-gram: 0.202846
Individual 3-gram: 0.178504
Individual 4-gram: 0.148753
4-gram cummulative score:  0.18619147304196104
-
```

In [22]:

```
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 [23]:

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

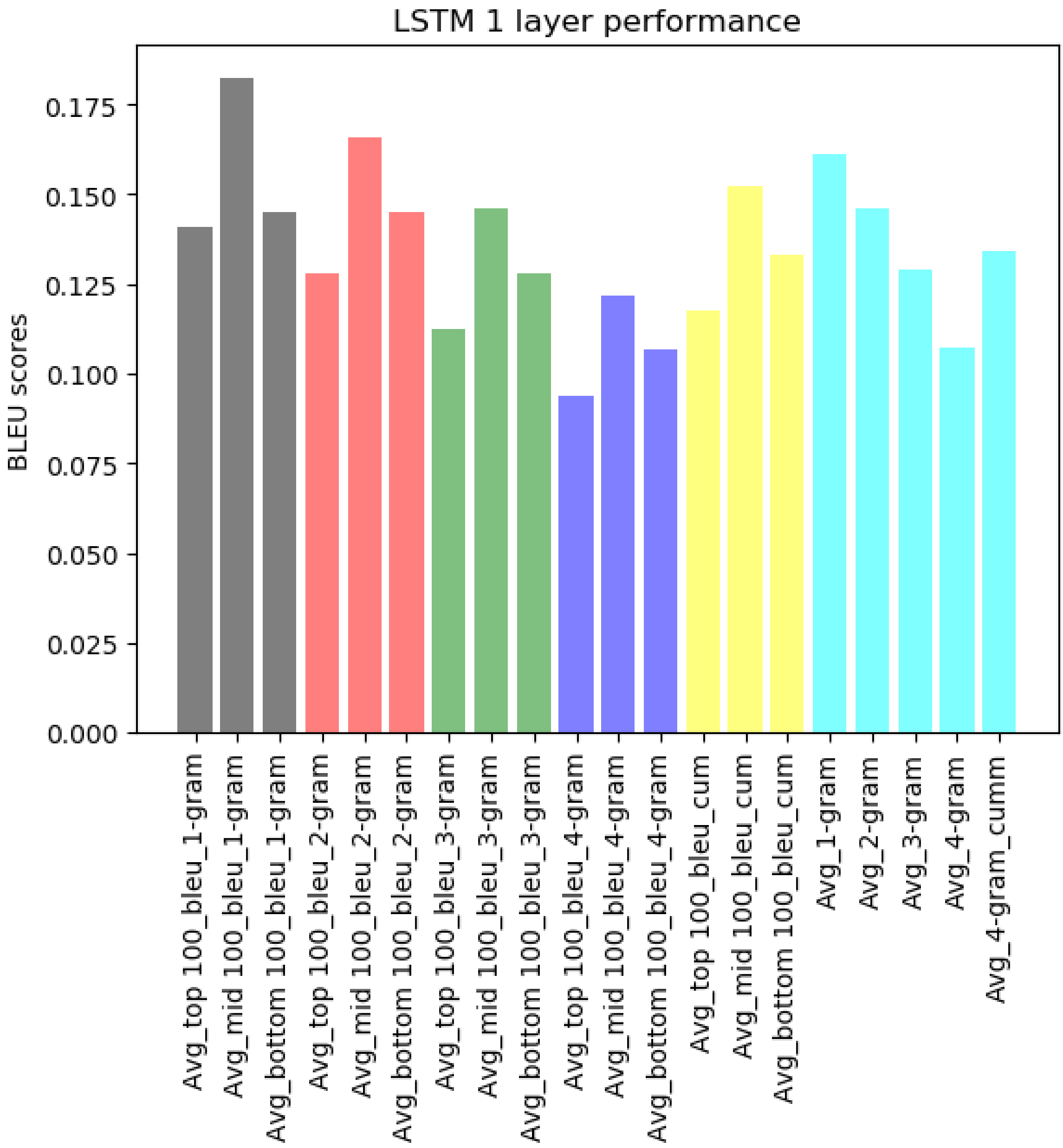
After editing the csv to reflect averages

In [24]:

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

In [25]:

```
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 very soon and is stubborn towards learning. So we will try to introduces changes to the model

--

Refining model

1. Increasinging the learning rate
2. Increasing the momentum
3. Toggling the Nesterov momentum on or off

    --

In [7]:

```
batch_size = 32  # Batch size for training.
epochs = 70  # Number of epochs to train for.
latent_dim = 256  # Latent dimensionality of the encoding space.
num_samples = 1000  # 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)
```

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: 1000
Number of unique input tokens: 1861
Number of unique output tokens: 1871
Max sequence length for inputs: 29
Max sequence length for outputs: 32
-------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]:

```
#EARLY STOPPING
#early_stopping = EarlyStopping(monitor='val_loss', patience=25)
#MODEL CHECKPOINT
ckpt_file = 'sgd_model.h1.29_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)
decoder_outputs = Dropout(0.3)(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.
W0730 00:36:44.600887  3008 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.

W0730 00:36:44.614878  3008 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.

W0730 00:36:44.618840  3008 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.

W0730 00:36:45.194707  3008 deprecation_wrapper.py:119] From c:\users\robust
us\appdata\local\programs\python\python37\lib\site-packages\keras\backend\te
nsorflow_backend.py:133: The name tf.placeholder_with_default is deprecated.
Please use tf.compat.v1.placeholder_with_default instead.

W0730 00:36:45.200658  3008 deprecation.py:506] From c:\users\robustus\appda
ta\local\programs\python\python37\lib\site-packages\keras\backend\tensorflow
_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with k
eep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
```

In [8]:

```
# Run training
#model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['acc'],options =
#model.compile(optimizer='sgd', loss='categorical_crossentropy',metrics=['acc'])
model.compile(optimizer=optimizers.sgd(lr=4, momentum=0.6, decay=0.2, nesterov=False), loss

model.summary()
```

```
____________________________________________________________________________
______________________
Layer (type)                    Output Shape          Param #     Connected t
o
============================================================================
======================
input_1 (InputLayer)            (None, None, 1861)    0
____________________________________________________________________________
______________________
input_2 (InputLayer)            (None, None, 1871)    0
____________________________________________________________________________
______________________
lstm_1 (LSTM)                   [(None, 256), (None,  2168832     input_1[0]
[0]
____________________________________________________________________________
______________________
lstm_2 (LSTM)                   [(None, None, 256),   2179072     input_2[0]
[0]
                                                                  lstm_1[0]
[1]
                                                                  lstm_1[0]
[2]
____________________________________________________________________________
______________________
dense_1 (Dense)                 (None, None, 1871)    480847      lstm_2[0]
[0]
____________________________________________________________________________
______________________
dropout_1 (Dropout)             (None, None, 1871)    0           dense_1[0]
[0]
============================================================================
======================
Total params: 4,828,751
Trainable params: 4,828,751
Non-trainable params: 0
____________________________________________________________________________
______________________
```

Running simple model with 256 latent_dims and batch size 32 for 200 epochs with 0.2 dropout rate

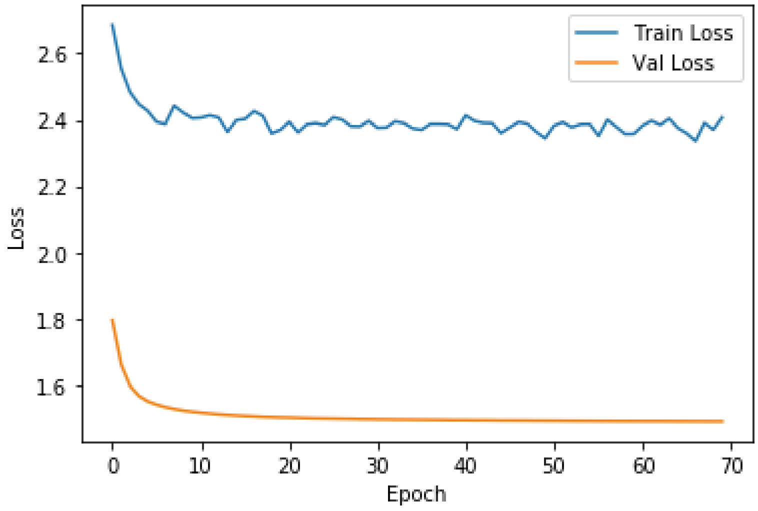
In [9]:

```
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_1000_w2w_s2s_64_256_70e.h5')
```

```
W0730 00:38:04.472416  3008 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

Train on 800 samples, validate on 200 samples
Epoch 1/70
800/800 [==============================] - 7s 9ms/step - loss: 2.6862 - ac
c: 0.0252 - val_loss: 1.7947 - val_acc: 0.0312

Epoch 00001: val_loss improved from inf to 1.79471, saving model to sgd_mo
del.h1.29_jul_19

c:\users\robustus\appdata\local\programs\python\python37\lib\site-packages
\keras\engine\network.py:877: UserWarning: Layer lstm_2 was passed non-ser
ializable keyword arguments: {'initial_state': [<tf.Tensor 'lstm_1/while/E
xit 2:0' shape=(?, 256) dtype=float32>, <tf.Tensor 'lstm 1/while/Exit 3:0'
```

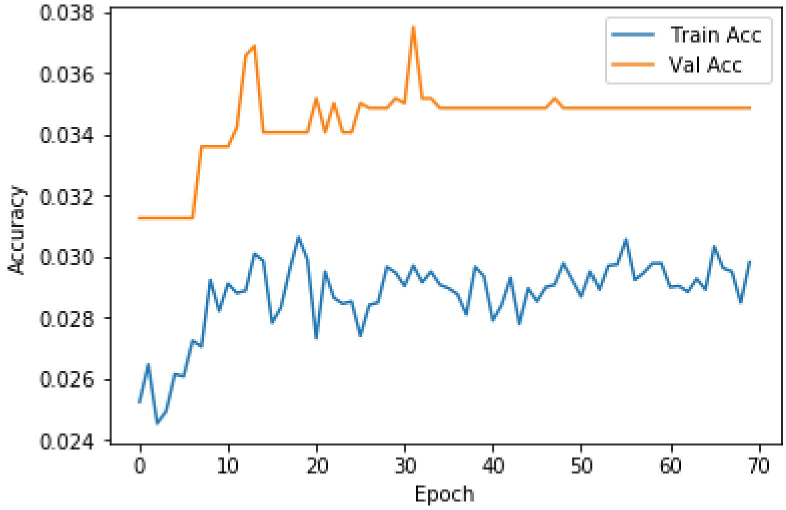
In [10]:

```
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 [11]:

```
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 [ ]:

```
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')
```

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)
```

Revised Model Inference

In [12]:

```
# 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 [ ]:

```
```

In [13]:

```
# 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 [14]:

```
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, :])
        #print ('sampled_token_index: ',sampled_token_index)
        sampled_word = reverse_target_word_index[sampled_token_index]
        #print ('sampled_word: ',sampled_word)
        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 [15]:

```
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 मैं , _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_  मैं मैं , _END
-
Input sentence: Then they'll be crying like babies.
Target sentence: START_ फिर ये नन्हें बच्चों की तरह रोएँगे। _END
Decoded sentence: START_  मैं मैं , _END
-
Input sentence: - No, I need power up!
Target sentence: START_ नहीं, मुझे पावर की जरुरत है ! _END
Decoded sentence: START_  मैं मैं , _END
-
Input sentence: I will not eat him.
Target sentence: START_ मैं उसे नहीं खा जाएगा. _END
Decoded sentence: START_  मैं मैं , _END
-
Input sentence: You gotta get me to Charleston.
Target sentence: START_ आप चार्ल्सटन करने के लिए मुझे जाना होगा. _END
Decoded sentence: START_  मैं मैं , _END
-
Input sentence: - NO, HE'S NOT MY DAD.
Target sentence: START_ - नहीं, वह मेरे पिता नहीं है. _END
Decoded sentence: START_  मैं मैं , _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 मैं , _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_  मैं मैं , _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_  मैं मैं , _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_  मैं मैं , _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_  मैं मैं , _END
-
```

```
Input sentence: You sent these?
Target sentence: START_ आप इन भेजा? _END
Decoded sentence: START_  मैं मैं , _END
-
Input sentence: I really loved him.
Target sentence: START_ मैं वास्तव में उसे प्यार करता था। _END
Decoded sentence: START_  मैं मैं , _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 मैं , _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_  मैं मैं , _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_  मैं मैं , _END
-
Input sentence: What do you fucking think?
Target sentence: START_ आपको क्या लगता है कि बकवास है? _END
Decoded sentence: START_  मैं मैं , _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 मैं , _END
-
Input sentence: Don't do it, man.
Target sentence: START_ , आदमी ऐसा मत करो. _END
Decoded sentence: START_  मैं मैं , _END
```

In [16]:

```
import nltk
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')
    BLEUscore1 = nltk.translate.bleu_score.sentence_bleu([target_sentence], decoded_sentenc
    BLEUscore2 = nltk.translate.bleu_score.sentence_bleu([target_sentence], decoded_sentenc
    BLEUscore3 = nltk.translate.bleu_score.sentence_bleu([target_sentence], decoded_sentenc
    BLEUscore4 = nltk.translate.bleu_score.sentence_bleu([target_sentence], decoded_sentenc
    print('BLEU score 1 gram',BLEUscore1)
    print('BLEU score 2 gram',BLEUscore2)
    print('BLEU score 3 gram',BLEUscore3)
    print('BLEU score 4 gram',BLEUscore4)
```

```
-
Input sentence: I do not want to die.
Target sentence: START_ मैं मरना नहीं चाहता. _END
Decoded sentence: START_  मैं मैं , _END
BLEU score 1 gram 0.08864252668986711
BLEU score 2 gram 0.08091918570382325
BLEU score 3 gram 0.07134456738837246
BLEU score 4 gram 0.0644243620250181
-
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_  मैं मैं , _END
BLEU score 1 gram 0.020023961976195638
BLEU score 2 gram 0.00861694125098847
BLEU score 3 gram 3.984463780026941e-104
BLEU score 4 gram 2.5295785065619195e-156
-
Input sentence: Then they'll be crying like babies.
Target sentence: START_ फिर ये नन्हें बच्चों की तरह रोएँगे। _END
Decoded sentence: START_  मैं मैं , _END
```

In [17]:

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

In [18]:

```
# 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 मैं , _END
Individual 1-gram: 0.577033
Individual 2-gram: 0.483609
Individual 3-gram: 0.380842
Individual 4-gram: 0.300665
4-gram cummulative score:  0.422795598767736
-
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_  मैं मैं , _END
Individual 1-gram: 0.300992
Individual 2-gram: 0.192699
Individual 3-gram: 0.147152
Individual 4-gram: 0.116172
4-gram cummulative score:  0.1774498889766137
-
```

In [19]:

```
# n-gram individual BLEU
from nltk.translate.bleu_score import sentence_bleu
for seq_index in range(450,550):
    # 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: Those guys, they just don't know how much I love 'em.
Target sentence: START_ उन लोगों को, वे तो बस मैं उन्हें कितना प्यार करता हूँ पता नहीं
है. _END
Decoded sentence: START_  मैं मैं , _END
Individual 1-gram: 0.074872
Individual 2-gram: 0.059762
Individual 3-gram: 0.043141
Individual 4-gram: 0.033026
4-gram cummulative score:  0.05024839927396918
-
Input sentence: Were you watching me?
Target sentence: START_ तुम मुझे देख रहे थे? _END
Decoded sentence: START_  मैं मैं , _END
Individual 1-gram: 0.490478
Individual 2-gram: 0.332481
Individual 3-gram: 0.253895
Individual 4-gram: 0.200443
4-gram cummulative score:  0.30182675548217774
```

In [20]:

```
# n-gram individual BLEU
from nltk.translate.bleu_score import sentence_bleu
for seq_index in range(900,1000):
    # 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: This hellfire club, it's got to be something else.
Target sentence: START_ वहाँ कुछ और होगा. _END
Decoded sentence: START_  मैं मैं , _END
Individual 1-gram: 0.496003
Individual 2-gram: 0.346415
Individual 3-gram: 0.290988
Individual 4-gram: 0.229728
4-gram cummulative score:  0.32737244591956594
-
Input sentence: Stay behind me.
Target sentence: START_ मेरे पीछे रहें। _END
Decoded sentence: START_  मैं मैं , _END
Individual 1-gram: 0.579421
Individual 2-gram: 0.417321
Individual 3-gram: 0.318681
Individual 4-gram: 0.251591
4-gram cummulative score:  0.37314600122213337
-
```

In [21]:

```
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 [22]:

```
df_bleu.to_csv('G:\\CSUEB\\MSBA\\Summer 19\\DL_BAN676\\Project\\LSTM_SGD_Dropout_Layer_BLEU
```

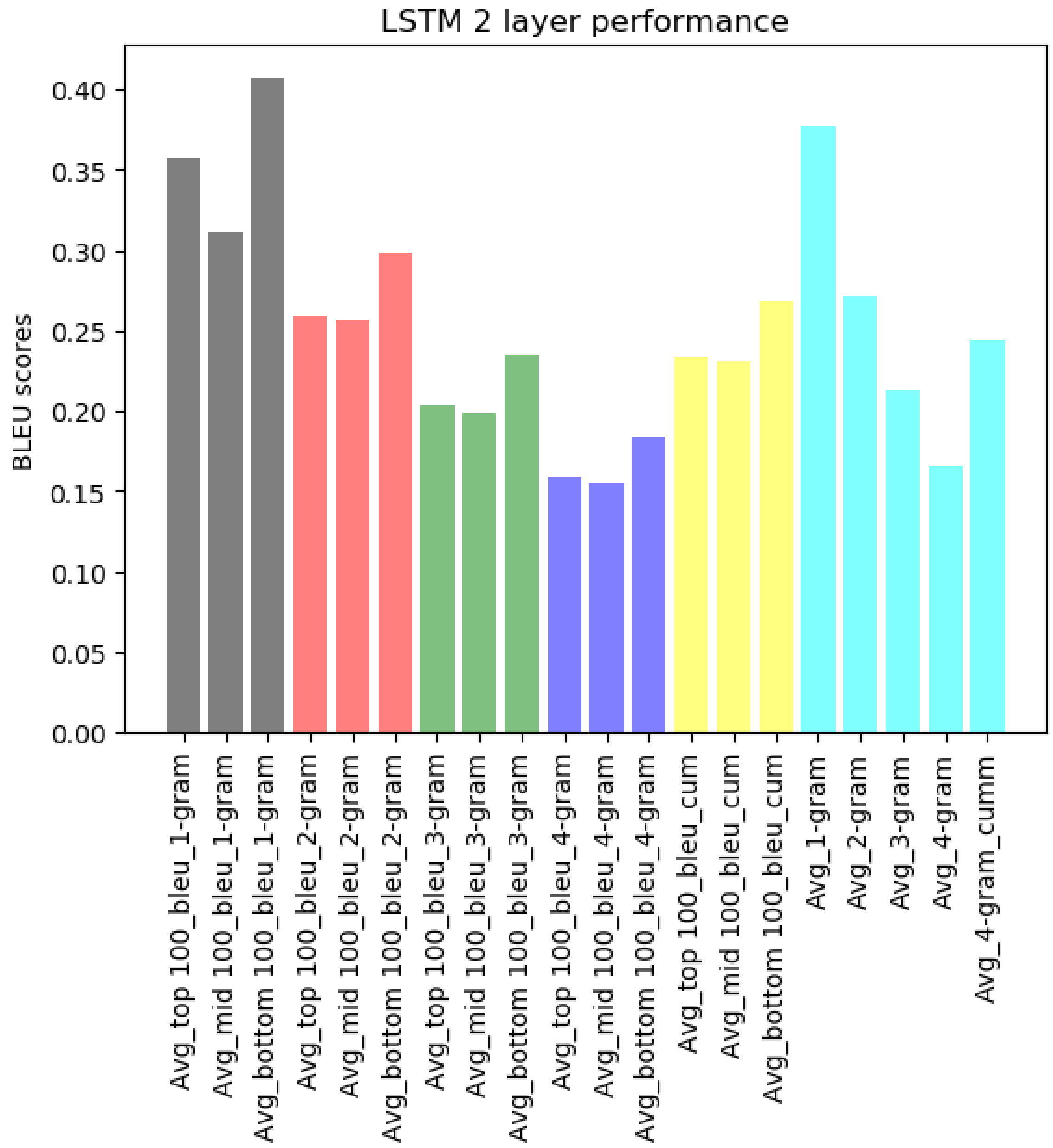
After editing the csv to reflect average values

In [3]:

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

In [4]:

```
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', 'red
plt.xticks(y_pos, objects,rotation=90)
plt.ylabel('BLEU scores')
plt.title('LSTM 2 layer performance')

plt.show()
```



In [ ]:

```

```