```
from __future__ import print_function

from keras.models import Model
from keras.layers import Input, LSTM, Dense
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', None)


batch_size = 64  # Batch size for training.
epochs = 50  # Number of epochs to train for.
latent_dim = 256  # Latent dimensionality of the encoding space.
num_samples = 3000  # Number of samples to train on.
# Path to the data txt file on disk.
data_path = 'swe.txt'


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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

```
# Vectorize the data.
input_texts = []
target_texts = []
input_characters = set()
target_characters = set()
with open(data_path, 'r', encoding='utf-8') as f:
    lines = f.read().split('\n')


# Vectorize the data.
input_texts = []
target_texts = []
input_characters = set()
target_characters = 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)]:
    input_text, target_text, not_used = line.split('\t')
    # We use "tab" as the "start sequence" character
    # for the targets, and "\n" as "end sequence" character.
    target_text = '\t' + target_text + '\n'
    input_texts.append(input_text)
    target_texts.append(target_text)
    for char in input_text:
        if char not in input_characters:
            input_characters.add(char)
    for char in target_text:
```

```
            if char not in target_characters:
                target_characters.add(char)

    input_characters = sorted(list(input_characters))
    target_characters = sorted(list(target_characters))
    num_encoder_tokens = len(input_characters)
    num_decoder_tokens = len(target_characters)
    max_encoder_seq_length = max([len(txt) for txt in input_texts])
    max_decoder_seq_length = max([len(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)

         Number of samples: 3000
         Number of unique input tokens: 61
         Number of unique output tokens: 66
         Max sequence length for inputs: 16
         Max sequence length for outputs: 41

    input_token_index = dict(
        [(char, i) for i, char in enumerate(input_characters)])
    target_token_index = dict(
        [(char, i) for i, char in enumerate(target_characters)])

    encoder_input_data = np.zeros(
        (len(input_texts), max_encoder_seq_length, num_encoder_tokens),
        dtype='float32')
    decoder_input_data = np.zeros(
        (len(input_texts), max_decoder_seq_length, num_decoder_tokens),
        dtype='float32')
    decoder_target_data = np.zeros(
        (len(input_texts), max_decoder_seq_length, num_decoder_tokens),
        dtype='float32')
    #0: (hola, hi)
      #0:hola
    #basically setuping up the matrix
    for i, (input_text, target_text) in enumerate(zip(input_texts, target_texts)):
        for t, char in enumerate(input_text):
            encoder_input_data[i, t, input_token_index[char]] = 1. #dummy encoding the matrix, so
        for t, char in enumerate(target_text):
            # decoder_target_data is ahead of decoder_input_data by one timestep
            decoder_input_data[i, t, target_token_index[char]] = 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[char]] = 1.


    # 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)



# Run training
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics='accuracy')
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, None, 61)]   0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, None, 66)]   0
__________________________________________________________________________________________________
lstm (LSTM)                     [(None, 256), (None, 325632      input_1[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, None, 256),  330752      input_2[0][0]
                                                                 lstm[0][1]
                                                                 lstm[0][2]
__________________________________________________________________________________________________
dense (Dense)                   (None, None, 66)     16962       lstm_1[0][0]
==================================================================================================
Total params: 673,346
Trainable params: 673,346
Non-trainable params: 0
__________________________________________________________________________________________________
```

```
model.fit([encoder_input_data, decoder_input_data], decoder_target_data,
          batch_size=batch_size,
          epochs=epochs,
          validation_split=0.2)
```

```
# Save model
model.save('swetoeng')
```

```
Epoch 1/50
38/38 [==============================] - 14s 273ms/step - loss: 1.2835 - accuracy: 0.0
Epoch 2/50
38/38 [==============================] - 10s 255ms/step - loss: 1.1784 - accuracy: 0.0
Epoch 3/50
38/38 [==============================] - 10s 252ms/step - loss: 1.1344 - accuracy: 0.0
Epoch 4/50
38/38 [==============================] - 10s 252ms/step - loss: 1.1008 - accuracy: 0.
Epoch 5/50
38/38 [==============================] - 10s 251ms/step - loss: 1.0725 - accuracy: 0.
Epoch 6/50
38/38 [==============================] - 10s 252ms/step - loss: 1.0463 - accuracy: 0.
Epoch 7/50
38/38 [==============================] - 10s 255ms/step - loss: 1.0259 - accuracy: 0.
Epoch 8/50
38/38 [==============================] - 10s 252ms/step - loss: 1.0100 - accuracy: 0.
Epoch 9/50
38/38 [==============================] - 9s 250ms/step - loss: 0.9927 - accuracy: 0.1
Epoch 10/50
38/38 [==============================] - 10s 253ms/step - loss: 0.9771 - accuracy: 0.
Epoch 11/50
38/38 [==============================] - 10s 251ms/step - loss: 0.9619 - accuracy: 0.
Epoch 12/50
38/38 [==============================] - 10s 252ms/step - loss: 0.9492 - accuracy: 0.
Epoch 13/50
38/38 [==============================] - 10s 255ms/step - loss: 0.9353 - accuracy: 0.
Epoch 14/50
38/38 [==============================] - 10s 252ms/step - loss: 0.9186 - accuracy: 0.
Epoch 15/50
38/38 [==============================] - 10s 257ms/step - loss: 0.9140 - accuracy: 0.
Epoch 16/50
38/38 [==============================] - 10s 254ms/step - loss: 0.9297 - accuracy: 0.
Epoch 17/50
38/38 [==============================] - 10s 259ms/step - loss: 0.9488 - accuracy: 0.
Epoch 18/50
38/38 [==============================] - 10s 252ms/step - loss: 0.9135 - accuracy: 0.
Epoch 19/50
38/38 [==============================] - 10s 254ms/step - loss: 0.8971 - accuracy: 0.
Epoch 20/50
38/38 [==============================] - 10s 254ms/step - loss: 0.8844 - accuracy: 0.
Epoch 21/50
38/38 [==============================] - 10s 254ms/step - loss: 0.8735 - accuracy: 0.
Epoch 22/50
38/38 [==============================] - 10s 256ms/step - loss: 0.8632 - accuracy: 0.
Epoch 23/50
38/38 [==============================] - 10s 255ms/step - loss: 0.8577 - accuracy: 0.
Epoch 24/50
38/38 [==============================] - 10s 256ms/step - loss: 0.8489 - accuracy: 0.
Epoch 25/50
38/38 [==============================] - 10s 258ms/step - loss: 0.8440 - accuracy: 0.
Epoch 26/50
38/38 [==============================] - 10s 258ms/step - loss: 0.8328 - accuracy: 0.
Epoch 27/50
38/38 [==============================] - 10s 255ms/step - loss: 0.8241 - accuracy: 0.
```

```
Epoch 28/50
38/38 [==============================] - 10s 254ms/step - loss: 0.8182 - accuracy: 0.
Epoch 29/50
38/38 [==============================] - 10s 256ms/step - loss: 0.8118 - accuracy: 0.
```

Double-click (or enter) to edit

This model is not very effecient as the val_loss does not decline while loss keeps declining. This model is overfit. 23.25% is the Best model.

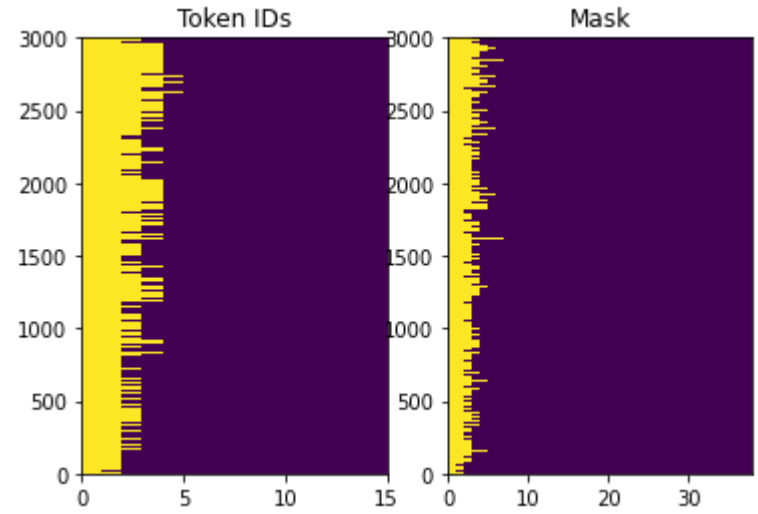
```
#now need to create model for character level model


from __future__ import print_function
import os
from keras.models import Model
from keras.layers import Input, LSTM, Dense
import numpy as np
import pandas as pd
from matplotlib import pyplot
!pip install contractions
pd.set_option('display.max_columns', None)


import contractions
input_texts2 = []
target_texts2 = []
with open('swe.txt', 'r', encoding='utf-8') as f:
 lines = f.read().split('\n')
for line in lines[: min(num_samples, len(lines) - 1)]:
 input_text2, target_text2, useless2 = line.split('\t')
 #target_text2 = '\t' + target_text2 + '\n'
 input_texts2.append(input_text2)
 target_texts2.append(target_text2)




def remove_punc(string):
    punc = '''!()-[]{};:"\,<>./?@#$%^&*_~'''
    for ele in string:
        if ele in punc:
            string = string.replace(ele, "")
    return string

#lis = ["Th@!is", "i#s" , "*&a", "list!", "For%", "#Pyt#$hon.?^pool"]
input_texts2 = [x.lower() for x in input_texts2]
input_texts2 = [remove_punc(i) for i in input_texts2]
```

```
target_texts2 = [x.lower() for x in target_texts2]
target_texts2 = [remove_punc(i) for i in target_texts2]



input_words2 = []
target_words2 = []
for i in input_texts2:
 i=i.split()
 for j in i:
    if j not in input_words2:
      input_words2.append(j)

input_words2=set(sorted(input_words2))
for i in target_texts2:
  i=i.split()
  for j in i:
    if j not in target_words2:
      target_words2.append(j)

target_words2=set(sorted(target_words2))
num_encoder_tokens2 = len(input_words2)+1
num_decoder_tokens2 = len(target_words2)+1


max_encoder_seq_length2 = max(len(txt) for txt in input_texts2)
max_decoder_seq_length2 = max(len(txt) for txt in target_texts2)

#max_encoder_seq_length2 = 12
#max_decoder_seq_length2 = 86
print('Number of samples:', len(input_texts2))
print('Number of unique input tokens:', num_encoder_tokens2)
print('Number of unique output tokens:', num_decoder_tokens2)
print('Max sequence length for inputs:', max_encoder_seq_length2)
print('Max sequence length for outputs:', max_decoder_seq_length2)

input_token_index2 = dict(
 [(char, i) for i, char in enumerate(input_words2)])
target_token_index2 = dict(
 [(char, i) for i, char in enumerate(target_words2)])
encoder_input_data2 = np.zeros(
 (len(input_texts2), max_encoder_seq_length2, num_encoder_tokens2),
 dtype='float32')
decoder_input_data2 = np.zeros(
 (len(input_texts2), max_decoder_seq_length2, num_decoder_tokens2),
 dtype='float32')
decoder_target_data2 = np.zeros(
 (len(input_texts2), max_decoder_seq_length2, num_decoder_tokens2),
 dtype='float32')
for i, (x, y) in enumerate(zip(input_texts2, target_texts2)):
  for t, word in enumerate(x.split()):
```

```
        encoder_input_data2[i, t, input_token_index2[word]] = 1. #dummy encoding the matrix, so
    for t, word in enumerate(y.split()):
    # decoder_target_data is ahead of decoder_input_data by one timestep  decoder_input_data[i,
       decoder_input_data2[i, t, target_token_index2[word]] = 1.
       if t > 0:
    # decoder_target_data will be ahead by one timestep
    # and will not include the start character.
        decoder_target_data2[i, t - 1, target_token_index2[word]] = 1.

        Number of samples: 3000
        Number of unique input tokens: 1251
        Number of unique output tokens: 1688
        Max sequence length for inputs: 15
        Max sequence length for outputs: 38


########START FROM HERE####
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession

config = ConfigProto()
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)

from keras.preprocessing.text import Tokenizer
t=Tokenizer()
t.fit_on_texts(input_texts2)
eng_matrix=t.texts_to_sequences(input_texts2)
t.fit_on_texts(target_texts2)
swe_matrix=t.texts_to_sequences(target_texts2)


from keras.preprocessing.sequence import pad_sequences
eng_padded = pad_sequences(eng_matrix, maxlen=max_encoder_seq_length2, padding='post')
swe_padded = pad_sequences(swe_matrix, maxlen=max_decoder_seq_length2, padding='post')


eng_padded= np.array(eng_padded)
swe_padded= np.array(swe_padded)


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(eng_padded, swe_padded, test_size=0.20, r



import os
from tensorflow.python.keras.layers import Layer
from tensorflow.python.keras import backend as K


!pip install tensorflow_text
import numpy as np
```

```
import typing
from typing import Any, Tuple

import tensorflow as tf
from tensorflow.keras.layers.experimental import preprocessing

import tensorflow_text as tf_text

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
use_builtins = True


plt.subplot(1, 2, 1)
plt.pcolormesh(eng_padded !=0)
plt.title('Token IDs')

plt.subplot(1, 2, 2)
plt.pcolormesh(swe_padded !=0)
plt.title('Mask')
```

```
Text(0.5, 1.0, 'Mask')
```




```
encoder_inputs_att = Input(shape=(None, num_encoder_tokens2))
encoder_att = LSTM(latent_dim, return_state=True)
encoder_outputs_att, state_h, state_c = encoder_att(encoder_inputs_att)
# We discard `encoder_outputs` and only keep the states.
encoder_states_att = [state_h, state_c]

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs_att = Input(shape=(None, num_decoder_tokens2))
# 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_att = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs_att, _, _ = decoder_lstm_att(decoder_inputs_att,
                                     initial_state=encoder_states_att)
```

```
decoder_dense_att = Dense(num_decoder_tokens2, activation='softmax')
decoder_outputs_att = decoder_dense_att(decoder_outputs_att)

# Define the model that will turn
# `encoder_input_data` & `decoder_input_data` into `decoder_target_data`
model3 = Model([encoder_inputs_att, decoder_inputs_att], decoder_outputs_att)

y_train.shape
```

```
(2400, 38)
```

```
#Define an input sequence and process it.
encoder_inputs = Input(shape=(None, num_encoder_tokens2))
encoder = LSTM(128, 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_tokens2))
decoder_lstm = LSTM(128, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_inputs,
 initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens2, 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)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',metrics=['accuracy'])
model.summary()
Model: "model_1"
```

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_5 (InputLayer)            [(None, None, 1251)] 0
__________________________________________________________________________________________________
input_6 (InputLayer)            [(None, None, 1688)] 0
__________________________________________________________________________________________________
lstm_5 (LSTM)                   [(None, 128), (None, 706560      input_5[0][0]
__________________________________________________________________________________________________
lstm_6 (LSTM)                   [(None, None, 128),  930304      input_6[0][0]
                                                                 lstm_5[0][1]
                                                                 lstm_5[0][2]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, None, 1688)   217752      lstm_6[0][0]
==================================================================================================
Total params: 1,854,616
Trainable params: 1,854,616
Non-trainable params: 0
__________________________________________________________________________________________________
```

```
history = model.fit([encoder_input_data2, decoder_input_data2], decoder_target_data2,  batch_
 epochs=50,
 validation_split=0.2)
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```

```
Epoch 1/50
75/75 [==============================] - 23s 272ms/step - loss: 0.3832 - accuracy: 0.006
Epoch 2/50
75/75 [==============================] - 19s 258ms/step - loss: 0.3662 - accuracy: 0.006
Epoch 3/50
75/75 [==============================] - 19s 258ms/step - loss: 0.3616 - accuracy: 0.006
Epoch 4/50
75/75 [==============================] - 20s 262ms/step - loss: 0.3586 - accuracy: 0.006
Epoch 5/50
75/75 [==============================] - 19s 260ms/step - loss: 0.3557 - accuracy: 0.006
Epoch 6/50
75/75 [==============================] - 20s 263ms/step - loss: 0.3529 - accuracy: 0.006
```

The model is not training at all and randomly stops mid training.

```
75/75 [==============================] - 20s 264ms/step - loss: 0.3487 - accuracy: 0.006
Epoch 9/50
75/75 [==============================] - 20s 263ms/step - loss: 0.3463 - accuracy: 0.006
Epoch 10/50
75/75 [==============================] - 20s 265ms/step - loss: 0.3468 - accuracy: 0.006
Epoch 11/50
75/75 [==============================] - 20s 268ms/step - loss: 0.3445 - accuracy: 0.006
Epoch 12/50
75/75 [==============================] - 20s 266ms/step - loss: 0.3435 - accuracy: 0.006
Epoch 13/50
75/75 [==============================] - 20s 265ms/step - loss: 0.3429 - accuracy: 0.006
Epoch 14/50
75/75 [==============================] - 20s 267ms/step - loss: 0.3425 - accuracy: 0.006
Epoch 15/50
75/75 [==============================] - 20s 263ms/step - loss: 0.3433 - accuracy: 0.006
Epoch 16/50
75/75 [==============================] - 20s 263ms/step - loss: 0.3420 - accuracy: 0.006
Epoch 17/50
75/75 [==============================] - 20s 265ms/step - loss: 0.3413 - accuracy: 0.006
Epoch 18/50
75/75 [==============================] - 20s 266ms/step - loss: 0.3412 - accuracy: 0.006
Epoch 19/50
75/75 [==============================] - 20s 263ms/step - loss: 0.3417 - accuracy: 0.006
Epoch 20/50
```