```
# Character model
```

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", f
```

```
from __future__ import print_function
from keras.models import Model
from keras.layers import Input, LSTM, GRU, Dense, Activation
from keras.activations import softmax as Softmax
from keras.preprocessing.sequence import pad_sequences
from keras.callbacks import ModelCheckpoint
from nltk.translate.bleu_score import corpus_bleu
from keras.utils import to_categorical
from keras import backend as K
import numpy as np
import unicodedata
import re
import tensorflow as tf
```

```
with open("/content/drive/My Drive/DeepLearning/mar.txt", "r") as a:
  print (a.readline())
```

```
Go.	जा.	CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #3138228 (sabretou)
```

```
# clean unwanted data
sep = '\tCC'
with open("/content/drive/My Drive/DeepLearning/mar.txt") as file_in:
    lines = []
    for line in file_in:
        res = line.split(sep,1)[0]
        #print(res)
        lines.append(res + "\n")
f = open('/content/drive/My Drive/DeepLearning/mar_1.txt','w')
for line in lines:
  f.write(line)
f.close()

# modified text file
with open("/content/drive/My Drive/DeepLearning/mar_1.txt", "r") as a:
  print (a.readline())
```

```
Go.	जा.
```

```
BATCH_SIZE = 32  # Batch size for training
NUM_SAMPLES = 10000
EPOCHS = 50
OPTIMIZER = "rmsprop"
EMBED_DIM = 300
HIDDEN_DIM = 256
DATA_PATH = '/content/drive/My Drive/DeepLearning/mar_1.txt'
```

```
def unicode_to_ascii(s):
    return ''.join(c for c in unicodedata.normalize('NFD', s)
                   if unicodedata.category(c) != 'Mn')
```

```
def preprocess_sentence(w):
    w = unicode_to_ascii(w.lower().strip())

    # creating a space between words and punctation following it
```

```
        # eg: "he is a boy." => "he is a boy ."
        w = re.sub(r"([?.!,¿])", r" \1 ", w)
        w = re.sub(r'[" "]+', " ", w)

        # replacing everything with space except (a-z, A-Z, ".", "?", "!", ",")
        #w = re.sub(r"[^a-zA-Z?.!,¿]+", " ", w)

        w = w.rstrip().strip()

        # adding a start and an end token to the sequence
        # so that the model know when to start and stop predicting
        w = "\t " + w + " \n"
        return w


    def loadGloveModel(gloveFile):
        print("Loading Glove Model")
        f = open(gloveFile, 'r', encoding = "utf8")
        model = {}
        for line in f:
            splitLine = line.split()
            word = splitLine[0]
            embedding = np.array([float(val) for val in splitLine[1:]])
            model[word] = embedding
        print("Done.", len(model), " words loaded!")
        return model


    def load_embedding(filename):
        print("Loading Glove Model")
        embedding_model = {}
        f = open(filename,'r', encoding="utf8")
        for line in f:
            values = line.split()
            word = ''.join(values[:-300])
            coefs = np.array(values[-300:], dtype='float32')
            embedding_model[word] = coefs
        print("Done.", len(embedding_model), " words loaded!")
        f.close()
        return embedding_model


    eng_embedding= loadGloveModel("/content/drive/My Drive/DeepLearning/glove.6B.300d.txt")

         Loading Glove Model
         Done. 400000  words loaded!


    input_texts = []
    target_texts = []
    target_chars = set()

    with open(DATA_PATH, 'r', encoding='utf-8') as f:
        lines = f.read().split("\n")
        for line in lines[:NUM_SAMPLES]:
            input_text, target_text = line.split('\t')
            input_text = preprocess_sentence(input_text)
            target_text = preprocess_sentence(target_text)
            input_texts.append(input_text)
            target_texts.append(target_text)
            target_chars.update(list(target_text))

    target_chars = sorted(list(target_chars))
    #print(target_text)


    # get attributes from data
    max_input_seqlen = max([len(txt.split()) for txt in input_texts])
    max_target_seqlen = max([len(txt) for txt in target_texts])
    target_token_size = len(target_chars)
```

```
# get decoder data
targchars2idx = dict([(char, i) for i, char in enumerate(target_chars)])
idx2targchars = dict((i, char) for char, i in targchars2idx.items())
decoder_data = np.zeros(
    shape=(NUM_SAMPLES, max_target_seqlen, target_token_size))
decoder_target_data = np.zeros(
    shape=(NUM_SAMPLES, max_target_seqlen, target_token_size))

for textIdx, text in enumerate(target_texts):
    for idx, char in enumerate(text):
        c2idx = targchars2idx[char]
        decoder_data[textIdx, idx, c2idx] = 1
        if idx > 0:
            decoder_target_data[textIdx, idx - 1, c2idx] = 1
#print(targchars2idx["\t"])


# get encoder data
encoder_data = []
for text in input_texts:
    tmp = []
    for word in text.split():
        embed = np.random.randn(EMBED_DIM)
        if word in eng_embedding:
            embed = eng_embedding[word]
        tmp.append(embed)
    encoder_data.append(tmp)
encoder_data = pad_sequences(encoder_data, max_input_seqlen, padding="post")


decoder_data.shape
```

```
(10000, 40, 74)
```

```
decoder_target_data.shape
```

```
(10000, 40, 74)
```

```
encoder_data.shape
```

```
(10000, 7, 300)
```

```
# construct model
encoder_inputs = Input(shape=(max_input_seqlen, EMBED_DIM))
encoder_lstm = LSTM(HIDDEN_DIM, return_state=True, name="encoder_lstm")
_, state_h, state_c = encoder_lstm(encoder_inputs)
encoder_states = [state_h, state_c]

decoder_inputs = Input(shape=(None, target_token_size))
decoder_lstm = LSTM(HIDDEN_DIM, return_sequences=True,
                    return_state=True, name="decoder_lstm")
decoder_outputs, _, _ = decoder_lstm(
    decoder_inputs, initial_state=encoder_states)
decoder_dense = Dense(
    target_token_size, activation="softmax", name="decoder_dense")
decoder_outputs = decoder_dense(decoder_outputs)


# define training model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

model.compile(optimizer=OPTIMIZER,
              loss='categorical_crossentropy', metrics=["acc"])
print(model.summary())
filename = 'seq2seq_keras.h5'
checkpoint = ModelCheckpoint(
    filename, verbose=1, save_best_only=True, mode='min')
# checkpoint = ModelCheckpoint(filename, verbose=1, mode='min')
```

```
# checkpoint = ModelCheckpoint(filename, verbose=1, mode='min')
model = model.fit([encoder_data, decoder_data], decoder_target_data,
          batch_size=BATCH_SIZE,
          epochs=EPOCHS, validation_split=0.2)
```

```
250/250 [==============================] - 2s 8ms/step - loss: 0.2033 - acc: 0.4056 - val_loss: 0.8710 - val_
Epoch 22/50
250/250 [==============================] - 2s 8ms/step - loss: 0.1903 - acc: 0.4093 - val_loss: 0.8671 - val_
Epoch 23/50
250/250 [==============================] - 2s 8ms/step - loss: 0.1780 - acc: 0.4133 - val_loss: 0.8916 - val_
Epoch 24/50
250/250 [==============================] - 2s 8ms/step - loss: 0.1674 - acc: 0.4159 - val_loss: 0.9094 - val_
Epoch 25/50
250/250 [==============================] - 2s 8ms/step - loss: 0.1567 - acc: 0.4190 - val_loss: 0.9231 - val_
Epoch 26/50
250/250 [==============================] - 2s 9ms/step - loss: 0.1477 - acc: 0.4216 - val_loss: 0.9408 - val_
Epoch 27/50
250/250 [==============================] - 2s 9ms/step - loss: 0.1392 - acc: 0.4239 - val_loss: 0.9585 - val_
Epoch 28/50
250/250 [==============================] - 2s 9ms/step - loss: 0.1313 - acc: 0.4260 - val_loss: 0.9814 - val_
Epoch 29/50
250/250 [==============================] - 2s 9ms/step - loss: 0.1238 - acc: 0.4284 - val_loss: 0.9966 - val_
Epoch 30/50
250/250 [==============================] - 2s 8ms/step - loss: 0.1172 - acc: 0.4298 - val_loss: 1.0180 - val_
Epoch 31/50
250/250 [==============================] - 2s 8ms/step - loss: 0.1113 - acc: 0.4315 - val_loss: 1.0118 - val_
Epoch 32/50
250/250 [==============================] - 2s 8ms/step - loss: 0.1046 - acc: 0.4335 - val_loss: 1.0536 - val_
Epoch 33/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0999 - acc: 0.4344 - val_loss: 1.0475 - val_
Epoch 34/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0946 - acc: 0.4359 - val_loss: 1.0734 - val_
Epoch 35/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0903 - acc: 0.4372 - val_loss: 1.1059 - val_
Epoch 36/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0863 - acc: 0.4381 - val_loss: 1.0825 - val_
Epoch 37/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0825 - acc: 0.4392 - val_loss: 1.1110 - val_
Epoch 38/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0792 - acc: 0.4400 - val_loss: 1.1317 - val_
Epoch 39/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0757 - acc: 0.4411 - val_loss: 1.1399 - val_
Epoch 40/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0732 - acc: 0.4415 - val_loss: 1.1585 - val_
Epoch 41/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0706 - acc: 0.4421 - val_loss: 1.1515 - val_
Epoch 42/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0673 - acc: 0.4432 - val_loss: 1.1768 - val_
Epoch 43/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0648 - acc: 0.4439 - val_loss: 1.1696 - val_
Epoch 44/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0628 - acc: 0.4445 - val_loss: 1.1689 - val_
Epoch 45/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0615 - acc: 0.4445 - val_loss: 1.1982 - val_
Epoch 46/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0597 - acc: 0.4452 - val_loss: 1.2099 - val_
Epoch 47/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0578 - acc: 0.4455 - val_loss: 1.2239 - val_
Epoch 48/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0563 - acc: 0.4460 - val_loss: 1.2319 - val_
Epoch 49/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0550 - acc: 0.4462 - val_loss: 1.2295 - val_
Epoch 50/50
250/250 [==============================] - 2s 8ms/step - loss: 0.0535 - acc: 0.4466 - val_loss: 1.2456 - val_
```

```
encoder_model = Model(encoder_inputs, encoder_states)

decoder_state_input_h = Input(shape=(HIDDEN_DIM,))
decoder_state_input_c = Input(shape=(HIDDEN_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_outputs = (BATCH_SIZE, seqlen, HIDDEN_DIM)
decoder_states = [state_h, state_c]
decoder_outputs = decoder_dense(decoder_outputs)
```

```
# decoder_outputs = (BATCH_SIZE, seqlen, target_token_size)
decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs, [decoder_outputs] + decoder_states)


def decode(input_seq):
    states = encoder_model.predict(input_seq)
    target_seq = np.zeros((1, 1, target_token_size))
    target_seq[0, 0, targchars2idx['\t']] = 1.0
    stop_condition = False
    prediction = ''

    while not stop_condition:
        output_tokens, h, c = decoder_model.predict(
            [target_seq] + states)
        sampled_token_idx = np.argmax(output_tokens[0, -1, :])
        sampled_char = idx2targchars[sampled_token_idx]
        prediction += sampled_char

        if (sampled_char == '\n' or len(prediction) > max_target_seqlen):
            stop_condition = True

        target_seq = np.zeros((1, 1, target_token_size))
        target_seq[0, 0, sampled_token_idx] = 1.0
        states = [h, c]

    return prediction


actual, predicted = list(), list()

for index in [1900, 5534, 7467, 1258, 4500, 1345, 7863, 7688, 6782]: # considered random index
    input_seq = encoder_data[index]
    input_seq = np.expand_dims(input_seq, axis=0)
    actual.append(target_texts[index].split())
    prediction = decode(input_seq)
    predicted.append(prediction.split())
    print('-')
    print("Input sentence:",input_texts[index])
    print("Translation: ", prediction)
```

```
-
Input sentence:          you will die .
Translation:   मराल तमही .
-
Input sentence:          give me my money .
Translation:   मला माझ पस द .
-
Input sentence:          i tried to forget .
Translation:   मी पिर झाला .
-
Input sentence:          what's that ?
Translation:   त काय आह ?
-
Input sentence:          i'm your friend .
Translation:   मी तझी मतरिण आह .
-
Input sentence:          come help me .
Translation:   या माझी मदत करा .
```

```
-
Input sentence:          she is her friend .

Translation:   ती तिची मतरिण आह .

-
Input sentence:          it was quite cold .

Translation:   बरयापकी थड होत .

-
Input sentence:          why didn't i die ?

Translation:   मी का नाही मलो ?
```