Mounting Google Drive in this notebook and loading the dataset of movie lines from there.

```
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", force_remoun
```

```
import keras
import json
from datetime import datetime
import numpy as np
```

```
dialogues_path = "/content/drive/MyDrive/Datasets/movie_lines.txt"
```

Read Data

Defining the vocab size and the word embedding dimensions

```
VOCAB_SIZE = 5000 # len(keras_tokenizer.word_index) + 1
print(VOCAB_SIZE)
EMBEDDING_DIM = 500
```

```
5000
```

Defining the libraries

```
from keras.preprocessing.text import Tokenizer
from statistics import median
```

Reading the input text file and demarking ~e at the end of every sentence in order to denote end of sentence.

```
EOS_TOKEN = "~e"
dialogue_lines = list()
with open(dialogues_path) as dialogues_file:
    for line in dialogues_file:
        line = line.strip().lower()
        split_line = line.split(' +++$+++ ')
        try:
            dialogue_lines.append(split_line[4] + " " + EOS_TOKEN)
        except IndexError:
            pass
#             print("Skipped line " + line)
```

```
dialogue_lines[:10]
```

```
['they do not! ~e',
 'they do to! ~e',
 'i hope so. ~e',
 'she okay? ~e',
 "let's go. ~e",
 'wow ~e',
 "okay -- you're gonna need to learn how to lie. ~e",
 'no ~e',
 'i\'m kidding.  you know how sometimes you just become this "persona"?  and you don\'t know how to quit? ~e',
 'like my fear of wearing pastels? ~e']
```

Using Keras Tokenizer to create token (Each word individually) from the sentence and removing all the punctuations from the sentences.

```
keras_tokenizer = Tokenizer(num_words=VOCAB_SIZE, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}\t\n')
```

Using Keras Tokenizer to fit on the sentence

```
keras_tokenizer.fit_on_texts(dialogue_lines)
```

```
# keras_tokenizer.word_index
```

Converting sentences into sequence of numbers

```
text_sequences = keras_tokenizer.texts_to_sequences(dialogue_lines)[:2000]


MAX_SEQUENCE_LENGTH = int(median(len(sequence) for sequence in text_sequences))
print(MAX_SEQUENCE_LENGTH)
```

```
8
```

Build Neural Network

```
from keras import backend as K
from keras.engine.topology import Layer
from keras.layers import Input, Dense, RepeatVector, LSTM, Conv1D, Masking, Embedding
from keras.layers.wrappers import TimeDistributed, Bidirectional
from keras.models import Model
from keras.preprocessing.sequence import pad_sequences


x_train = pad_sequences(text_sequences, maxlen=MAX_SEQUENCE_LENGTH, padding='post',
                        truncating='post', value=0)


x_train.shape
```

```
(100, 8)
```

```
x_train_rev = list()
for x_vector in x_train:
    x_rev_vector = list()
    for index in x_vector:
        char_vector = np.zeros(VOCAB_SIZE)
        char_vector[index] = 1
        x_rev_vector.append(char_vector)
    x_train_rev.append(np.asarray(x_rev_vector))
x_train_rev = np.asarray(x_train_rev)


x_train_rev.shape
```

```
(100, 8, 5000)
```

Creating the seq2seq neural network with simple LSTM cells and Categorical crossentropy as the loss function and adam as the optimizer.

```
def get_seq2seq_model():
    main_input = Input(shape=x_train[0].shape, dtype='float32', name='main_input')
    print(main_input)

    embed_1 = Embedding(input_dim=VOCAB_SIZE, output_dim=EMBEDDING_DIM,
                        mask_zero=True, input_length=MAX_SEQUENCE_LENGTH) (main_input)
    print(embed_1)

    lstm_1 = Bidirectional(LSTM(2048, name='lstm_1'))(embed_1)
    print(lstm_1)

    repeat_1 = RepeatVector(MAX_SEQUENCE_LENGTH, name='repeat_1')(lstm_1)
    print(repeat_1)

    lstm_3 = Bidirectional(LSTM(2048, return_sequences=True, name='lstm_3'))(repeat_1)
    print(lstm_3)

    softmax_1 = TimeDistributed(Dense(VOCAB_SIZE, activation='softmax'))(lstm_3)
    print(softmax_1)

    model = Model(main_input, softmax_1)
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model


seq2seq_model = get_seq2seq_model()
```

```
Tensor("main_input:0", shape=(None, 8), dtype=float32)
Tensor("embedding/embedding_lookup/Identity_1:0", shape=(None, 8, 500), dtype=float32)
Tensor("bidirectional/concat:0", shape=(None, 4096), dtype=float32)
Tensor("repeat_1/Tile:0", shape=(None, 8, 4096), dtype=float32)
Tensor("bidirectional_1/concat:0", shape=(None, 8, 4096), dtype=float32)
Tensor("time_distributed/Reshape_1:0", shape=(None, 8, 5000), dtype=float32)
```

```
seq2seq_model.fit(x_train, x_train_rev, batch_size=128, epochs=20)
```

```
Epoch 1/20
1/1 [==============================] - 0s 2ms/step - loss: 8.5168 - accuracy: 0.0000e+00
Epoch 2/20
1/1 [==============================] - 0s 2ms/step - loss: 8.4257 - accuracy: 0.2688
Epoch 3/20
1/1 [==============================] - 0s 2ms/step - loss: 7.8848 - accuracy: 0.2650
Epoch 4/20
1/1 [==============================] - 0s 2ms/step - loss: 7.6355 - accuracy: 0.2650
Epoch 5/20
1/1 [==============================] - 0s 2ms/step - loss: 6.4500 - accuracy: 0.2650
Epoch 6/20
1/1 [==============================] - 0s 3ms/step - loss: 6.1519 - accuracy: 0.2988
Epoch 7/20
1/1 [==============================] - 0s 1ms/step - loss: 5.3593 - accuracy: 0.2688
Epoch 8/20
1/1 [==============================] - 0s 2ms/step - loss: 4.7754 - accuracy: 0.2075
Epoch 9/20
1/1 [==============================] - 0s 2ms/step - loss: 4.1648 - accuracy: 0.2800
Epoch 10/20
1/1 [==============================] - 0s 3ms/step - loss: 4.1554 - accuracy: 0.2850
Epoch 11/20
1/1 [==============================] - 0s 2ms/step - loss: 4.0323 - accuracy: 0.2875
Epoch 12/20
1/1 [==============================] - 0s 2ms/step - loss: 3.9976 - accuracy: 0.2875
Epoch 13/20
1/1 [==============================] - 0s 1ms/step - loss: 3.9486 - accuracy: 0.2912
Epoch 14/20
1/1 [==============================] - 0s 2ms/step - loss: 3.8591 - accuracy: 0.2800
Epoch 15/20
1/1 [==============================] - 0s 2ms/step - loss: 3.7775 - accuracy: 0.2788
Epoch 16/20
1/1 [==============================] - 0s 1ms/step - loss: 3.7330 - accuracy: 0.2837
Epoch 17/20
1/1 [==============================] - 0s 1ms/step - loss: 3.7060 - accuracy: 0.2875
Epoch 18/20
1/1 [==============================] - 0s 1ms/step - loss: 3.6878 - accuracy: 0.2875
Epoch 19/20
1/1 [==============================] - 0s 2ms/step - loss: 3.6424 - accuracy: 0.2812
Epoch 20/20
1/1 [==============================] - 0s 2ms/step - loss: 3.5646 - accuracy: 0.2825
<tensorflow.python.keras.callbacks.History at 0x7f88011bbb00>
```

Predicting the training dataset on the trained model.

```
predictions = seq2seq_model.predict(x_train)
```

Finding out the dictionary for number to word mapping

```
index2word_map = inv_map = {v: k for k, v in keras_tokenizer.word_index.items()}
```

Converting the numbers back to sequences.

```
def sequence_to_str(sequence):
    word_list = list()
    for element in sequence:
#         if amax(element) < max_prob:
#             continue
        index = np.argmax(element) + 1
        word = index2word_map[index]
        word_list.append(word)

    return word_list


predictions_file_path = \
    "/content/" + datetime.now().strftime('%Y-%m-%d-%H-%M-%S') + ".txt"
```

Predicting the test sentences and checking the performance of the model by comparing them with the actual test results.

```
with open(predictions_file_path, 'w') as predictions_file:
    for i in range(len(predictions)):
        predicted_word_list = sequence_to_str(predictions[i])
        actual_len = len(dialogue_lines[i])

        actual_sentence = "Actual: " + dialogue_lines[i][:len(dialogue_lines[i])-3]

        generated_sentence = ""
        for word in predicted_word_list:
            if word == EOS_TOKEN:
```

```
                if word == EOS_TOKEN:
                    predictions_file.write('\n')
                    break
                generated_sentence += word + " "

            sent_dict = dict()
            sent_dict["actual"] = actual_sentence.strip()
            sent_dict["generated"] = generated_sentence.strip()

            predictions_file.write(json.dumps(sent_dict, sort_keys=True, indent=2, separators=(',', ': ')))
            predictions_file.write("\n")
```

Start coding or generate with AI.