Neural Machine Translation

- Translate a given sentence in one language to another desired language.

In this notebook, we aim to build a model which can translate German sentences to English.

Dataset

Dataset is taken from http://www.manythings.org/anki/.

We are considering German – English deu-eng.zip file from the above mentioned website.

In the above zip file there is a file with name `deu.txt` that contains **152,820** pairs of English to German phras
separating the phrases.

For example,

The first 5 lines in deu.txt are as given below.

```
Hi.    Hallo!
Hi.    Grüß Gott!
Run!    Lauf!
Wow!    Potzdonner!
Wow!    Donnerwetter!
```

Problem

Given a sequence of words in German as input, predict the sequence of words in Englis

1. Prepare Data

The preprocessing of the data involves:

1. Removing punctuation marks from the data.
2. Converting text corpus into lower case characters.
3. Split into Train and Test sets.
4. Shuffling the sentences.

The above tasks are done and full dataset is given as `english-german-both.pkl` respectively.

Download dataset files from here: https://drive.google.com/open?id=1gWVk7SuuE93Cf_nT9Lb7GBCiwfAgdE

Character level Machine Translation

Initialize parameters

Run the below code to initialize the variables required for the model.

```
batch_size = 64  # Batch size for training.
epochs = 10  # Number of epochs to train for.
latent_dim = 256  # Latent dimensionality of the encoding space.
num_samples = 10000  # Number of samples to train on.
# Path to the data txt file on disk.
data_path = 'fra.txt'
```

Connect to google drive

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

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

Give the path for the folder in which the dataset is present in google drive

```
project_path = "/content/drive/My Drive/"
```

Change present working directory

```
import os
os.chdir(project_path)
```

Load the pickle file (`english-german-both.pkl`) into a variable with name `da`

Run the below code to load the .pkl file.

```
import pickle

with open(project_path + 'english-german-both.pkl', 'rb') as f:
  dataset = pickle.load(f)
```

Check the `dataset` variable at this step. It should be as given below

```
dataset
```

```
array([['stay with us', 'bleib bei uns'],
       ['she wants him', 'sie will ihn'],
       ['youre strong', 'du bist stark'],
       ...,
       ['i thought so', 'das dachte ich mir'],
       ['keep warm', 'haltet euch warm'],
       ['im sick', 'ich bin krank']], dtype='<U291')
```

Feature set and target set division from the **dataset**

Run the below code to divide the dataset into feature set(input) and target set(output).

1. We are creating two lists for storing input sentences and output sentences separately.
2. We are storing each character in a list from both input and target sets separately.
3. Print and check `input_texts` and `target_texts`.
4. Print and check `input_characters` and `target_characters`.

```
# Vectorize the data.
input_texts = []
target_texts = []
input_characters = set()
target_characters = set()

for line in dataset[: min(num_samples, len(dataset) - 1)]:
    # This conversion is from English[0] to German[1]
    # input_text, target_text = line[0], line[1]

    # This conversion is from German[1] to English[0]
    input_text, target_text = line[1], line[0]

    # 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)
```

Print input text

```
print(input_text)
```

```
haltet euch warm
```

Print target text

```
print(target_text)
```

```
	keep warm
```

Print input character

```
print(input_characters)
```

```
{'e', 'p', 'g', 'z', 'j', 'y', 'f', 'u', 'k', 'v', 'h', 'm', 'r', 'i', 'c', 'o', 't', 's', 'w',
```

Print target character

```
print(target_characters)
```

```
{'e', 'g', 'p', 'z', 'j', 'y', 'f', 'u', 'k', 'v', 'h', 'm', 'r', 'i', 'c', '\t', 'o', 't', 's'
```

Stats from the dataset

Run the below code to check the stats from the dataset

```
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: 9999
Number of unique input tokens: 27
Number of unique output tokens: 29
Max sequence length for inputs: 49
Max sequence length for outputs: 17
```

Build character to index dictionary names `input_token_index` and `target_t` and target sets respectively.

```
import tensorflow as tf
```

```
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)])
```

```
#Tokenizer for source language
#encoder_t = tf.keras.preprocessing.text.Tokenizer()
#encoder_t.fit_on_texts(input_text) #Fit it on Source sentences
#input_token_index = encoder_t.texts_to_sequences(input_text) #Convert sentences to numbers
#input_token_index[1:10] #Display some converted sentences
```

```
#Tokenizer for target language, filters should not <start> and <end>
#remove < and > used in Target language sequences
#decoder_t = tf.keras.preprocessing.text.Tokenizer()
#decoder_t.fit_on_texts(target_text) #Fit it on target sentences
#target_token_index = decoder_t.texts_to_sequences(target_text) #Convert sentences to numbers
```

Print input_index_token

```
print(input_token_index)
```

```
{' ': 0, 'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k':
```

Print target_token_index

```
print(target_token_index)
```

```
{'\t': 0, '\n': 1, ' ': 2, 'a': 3, 'b': 4, 'c': 5, 'd': 6, 'e': 7, 'f': 8, 'g': 9, 'h': 10, 'i'
```

Build Model

Initialize the required layers from keras

Import libraries

```
from __future__ import print_function

from keras.models import Model
from keras.layers import Input, LSTM, Dense
import numpy as np
```

```
Using TensorFlow backend.
```

Run the below code to build one-hot vectors for the 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')
```

```
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.
    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.
```

Build the encoder Model

Define an input sequence and process it.

Discard `encoder_outputs` and only keep the states.

```
encoder_inputs = Input(shape=(None, num_encoder_tokens))
encoder = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_inputs)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend
```

```
encoder_states = [state_h, state_c]
```

Build the decoder Model

Set up the decoder, using `encoder_states` as initial state.

We set up our decoder to return full output sequences, and to return internal states as well. We don't use the
but we will use them in inference.

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

Define the model that will turn `encoder_input_data` & `decoder_input_data` into `decoder_target_data`

```
#Defining Model
```

```
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

Compile and fit the model

```
# Compiling the model
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: The nam

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend
```

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

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/math_
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend

Train on 7999 samples, validate on 2000 samples
Epoch 1/10
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend

7999/7999 [==============================] - 11s 1ms/step - loss: 2.1338 - acc: 0.1905 - val_lo
Epoch 2/10
7999/7999 [==============================] - 9s 1ms/step - loss: 1.7803 - acc: 0.2906 - val_los
Epoch 3/10
7999/7999 [==============================] - 9s 1ms/step - loss: 1.5919 - acc: 0.3282 - val_los
Epoch 4/10
7999/7999 [==============================] - 9s 1ms/step - loss: 1.4840 - acc: 0.3488 - val_los
Epoch 5/10
7999/7999 [==============================] - 9s 1ms/step - loss: 1.4087 - acc: 0.3683 - val_los
Epoch 6/10
7999/7999 [==============================] - 9s 1ms/step - loss: 1.3615 - acc: 0.3811 - val_los
Epoch 7/10
7999/7999 [==============================] - 9s 1ms/step - loss: 1.2913 - acc: 0.4037 - val_los
Epoch 8/10
7999/7999 [==============================] - 9s 1ms/step - loss: 1.2342 - acc: 0.4213 - val_los
Epoch 9/10
7999/7999 [==============================] - 9s 1ms/step - loss: 1.1867 - acc: 0.4365 - val_los
Epoch 10/10
7999/7999 [==============================] - 9s 1ms/step - loss: 1.1466 - acc: 0.4442 - val_los
<keras.callbacks.History at 0x7fb5bcb12dd8>
```

Save the model

```
# Save model
model.save('s2s.h5')
```

Run the below code for inferencing the model

```
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)
```

Reverse-lookup token index to decode sequences back to something readab

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

```
print(reverse_input_char_index)
print(reverse_target_char_index)
```

```
{0: ' ', 1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f', 7: 'g', 8: 'h', 9: 'i', 10: 'j', 11: '
 {0: '\t', 1: '\n', 2: ' ', 3: 'a', 4: 'b', 5: 'c', 6: 'd', 7: 'e', 8: 'f', 9: 'g', 10: 'h', 11:
```

```
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['\t']] = 1.

    # Sampling loop for a batch of sequences
    # (to simplify, here we assume a batch of size 1).
    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict(
            [target_seq] + states_value)

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]
        decoded_sentence += sampled_char

        # Exit condition: either hit max length
        # or find stop character.
        if (sampled_char == '\n' 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
```

Run the below code for checking some outputs from the model.

```
for seq_index in range(10):
    # 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('Decoded sentence:', decoded_sentence)
```

```
-
Input sentence: bleib bei uns
Decoded sentence: its too sack

-
Input sentence: sie will ihn
Decoded sentence: its too sack

-
Input sentence: du bist stark
Decoded sentence: its too sack

-
Input sentence: untersuchen sie das
Decoded sentence: its too sack

-
Input sentence: hier ist meine karte
Decoded sentence: its too sack

-
Input sentence: tom stie auf
Decoded sentence: tom is that

-
Input sentence: das ist kein witz
Decoded sentence: its too sack

-
Input sentence: tom ist ein spion
Decoded sentence: tom is that

-
Input sentence: ich bin ein teenager
Decoded sentence: im not a cat

-
Input sentence: ich bin nicht verruckt
Decoded sentence: im not a cat
```