```
import numpy as np
import pandas as pd
import os
```

Attention Class

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


class AttentionLayer(Layer):
    """
    This class implements Bahdanau attention
(https://arxiv.org/pdf/1409.0473.pdf).
    There are three sets of weights introduced W_a, U_a, and V_a
     """
def __init__(self, **kwargs):
        super(AttentionLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        assert isinstance(input_shape, list)
        # Create a trainable weight variable for this layer.

        self.W_a = self.add_weight(name='W_a',

shape=tf.TensorShape((input_shape[0][2], input_shape[0][2])),
                                   initializer='uniform',
                                   trainable=True)
        self.U_a = self.add_weight(name='U_a',

shape=tf.TensorShape((input_shape[1][2], input_shape[0][2])),
                                   initializer='uniform',
                                   trainable=True)
        self.V_a = self.add_weight(name='V_a',

shape=tf.TensorShape((input_shape[0][2], 1)),
                                   initializer='uniform',
                                   trainable=True)
        super(AttentionLayer, self).build(input_shape)  # Be sure to
call this at the end

    def call(self, inputs, verbose=False):
        """
        inputs: [encoder_output_sequence, decoder_output_sequence]
        """
        assert type(inputs) == list
        encoder_out_seq, decoder_out_seq = inputs
        if verbose:
```

```
            print('encoder_out_seq>', encoder_out_seq.shape)
            print('decoder_out_seq>', decoder_out_seq.shape)

        def energy_step(inputs, states):
            """ Step function for computing energy for a single
decoder state
            inputs: (batchsize * 1 * de_in_dim)
            states: (batchsize * 1 * de_latent_dim)
            """

            assert_msg = "States must be an iterable. Got {} of type
{}".format(states, type(states))
            assert isinstance(states, list) or isinstance(states,
tuple), assert_msg

            """ Some parameters required for shaping tensors"""
            en_seq_len, en_hidden = encoder_out_seq.shape[1],
encoder_out_seq.shape[2]
            de_hidden = inputs.shape[-1]

            """ Computing S.Wa where S=[s0, s1, ..., si]"""
            # <= batch size * en_seq_len * latent_dim
            W_a_dot_s = K.dot(encoder_out_seq, self.W_a)

            """ Computing hj.Ua """
            U_a_dot_h = K.expand_dims(K.dot(inputs, self.U_a), 1)  #
<= batch_size, 1, latent_dim
            if verbose:
                print('Ua.h>', U_a_dot_h.shape)
                 print('Ua.h>', U_a_dot_h.shape)

            """ tanh(S.Wa + hj.Ua) """
            # <= batch_size*en_seq_len, latent_dim
            Ws_plus_Uh = K.tanh(W_a_dot_s + U_a_dot_h)
            if verbose:
                print('Ws+Uh>', Ws_plus_Uh.shape)

            """ softmax(va.tanh(S.Wa + hj.Ua)) """
            # <= batch_size, en_seq_len
            e_i = K.squeeze(K.dot(Ws_plus_Uh, self.V_a), axis=-1)
            # <= batch_size, en_seq_len
            e_i = K.softmax(e_i)

            if verbose:
                print('ei>', e_i.shape)

            return e_i, [e_i]
        def context_step(inputs, states):
            """ Step function for computing ci using ei """
```

```
            assert_msg = "States must be an iterable. Got {} of type
{}".format(states, type(states))
            assert isinstance(states, list) or isinstance(states,
tuple), assert_msg

            # <= batch_size, hidden_size
            c_i = K.sum(encoder_out_seq * K.expand_dims(inputs, -1),
axis=1)
            if verbose:
                print('ci>', c_i.shape)
            return c_i, [c_i]

        fake_state_c = K.sum(encoder_out_seq, axis=1)
        fake_state_e = K.sum(encoder_out_seq, axis=2)  # <=
(batch_size, enc_seq_len, latent_dim

        """ Computing energy outputs """
        # e_outputs => (batch_size, de_seq_len, en_seq_len)
        last_out, e_outputs, _ = K.rnn(
            energy_step, decoder_out_seq, [fake_state_e],
        )

        """ Computing context vectors """
        last_out, c_outputs, _ = K.rnn(
            context_step, e_outputs, [fake_state_c],
        )

        return c_outputs, e_outputs

    def compute_output_shape(self, input_shape):
        """ Outputs produced by the layer """
        return [
            tf.TensorShape((input_shape[1][0], input_shape[1][1],
input_shape[1][2])),
            tf.TensorShape((input_shape[1][0], input_shape[1][1],
input_sape[1][2])),
            tf.TensorShape((input_shape[1][0], input_shape[1][1],
input_shape[0][1]))
        ]

import re

lines = open('../input/chatbot-data/cornell movie-dialogs
corpus/movie_lines.txt', encoding='utf-8',
             errors='ignore').read().split('\n')

convers = open('../input/chatbot-data/cornell movie-dialogs
corpus/movie_conversations.txt', encoding='utf-8',
             errors='ignore').read().split('\n')
```

```
len(lines)
```

```
304714
```

```
exchn = []
for conver in convers:
    exchn.append(conver.split(' +++$+++ ')[-1][1:-1].replace("'", "
").replace(",","").split())

diag = {}
for line in lines:
    diag[line.split(' +++$+++ ')[0]] = line.split(' +++$+++ ')[-1]



## delete
del(lines, convers, conver, line)



questions = []
answers = []

for conver in exchn:
    for i in range(len(conver) - 1):
        questions.append(diag[conver[i]])
        answers.append(diag[conver[i+1]])




## delete
del(diag, exchn, conver, i)


################################
#        max_len = 13          #
################################

sorted_ques = []
sorted_ans = []
for i in range(len(questions)):
    if len(questions[i]) < 13:
        sorted_ques.append(questions[i])
        sorted_ans.append(answers[i])



################################
#                              #
################################
```

```
def clean_text(txt):
    txt = txt.lower()
    txt = re.sub(r"i'm", "i am", txt)
    txt = re.sub(r"he's", "he is", txt)
    txt = re.sub(r"she's", "she is", txt)
    txt = re.sub(r"that's", "that is", txt)
    txt = re.sub(r"what's", "what is", txt)
    txt = re.sub(r"where's", "where is", txt)
    txt = re.sub(r"\'ll", " will", txt)
    txt = re.sub(r"\'ve", " have", txt)
    txt = re.sub(r"\'re", " are", txt)
    txt = re.sub(r"\'d", " would", txt)
    txt = re.sub(r"won't", "will not", txt)
    txt = re.sub(r"can't", "can not", txt)
    txt = re.sub(r"[^\w\s]", "", txt)
    return txt

clean_ques = []
clean_ans = []

for line in sorted_ques:
    clean_ques.append(clean_text(line))

for line in sorted_ans:
    clean_ans.append(clean_text(line))



## delete
del(answers, questions, line)


################################
#                              #
################################


for i in range(len(clean_ans)):
    clean_ans[i] = ' '.join(clean_ans[i].split()[:11])



################################
#                              #
################################

del(sorted_ans, sorted_ques)
```

```
## trimming
clean_ans=clean_ans[:30000]
clean_ques=clean_ques[:30000]
## delete


###  count occurences ###
word2count = {}

for line in clean_ques:
    for word in line.split():
        if word not in word2count:
            word2count[word] = 1
        else:
            word2count[word] += 1
for line in clean_ans:
    for word in line.split():
        if word not in word2count:
            word2count[word] = 1
        else:
            word2count[word] += 1

## delete
del(word, line)


###  remove less frequent ###
thresh = 5

vocab = {}
word_num = 0
for word, count in word2count.items():
    if count >= thresh:
        vocab[word] = word_num
        word_num += 1

## delete
del(word2count, word, count, thresh)
del(word_num)



for i in range(len(clean_ans)):
    clean_ans[i] = '<SOS> ' + clean_ans[i] + ' <EOS>'



tokens = ['<PAD>', '<EOS>', '<OUT>', '<SOS>']
x = len(vocab)
```

```
for token in tokens:
    vocab[token] = x
    x += 1



vocab['cameron'] = vocab['<PAD>']
vocab['<PAD>'] = 0

## delete
del(token, tokens)
del(x)

### inv answers dict ###
inv_vocab = {w:v for v, w in vocab.items()}



## delete
del(i)



encoder_inp = []
for line in clean_ques:
    lst = []
    for word in line.split():
        if word not in vocab:
            lst.append(vocab['<OUT>'])
        else:
            lst.append(vocab[word])

    encoder_inp.append(lst)

decoder_inp = []
for line in clean_ans:
    lst = []
    for word in line.split():
        if word not in vocab:
            lst.append(vocab['<OUT>'])
        else:
            lst.append(vocab[word])
    decoder_inp.append(lst)

### delete
del(clean_ans, clean_ques, line, lst, word)
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
encoder_inp = pad_sequences(encoder_inp, 13, padding='post',
truncating='post')
decoder_inp = pad_sequences(decoder_inp, 13, padding='post',
truncating='post')
```

```
decoder_final_output = []
for i in decoder_inp:
    decoder_final_output.append(i[1:])

decoder_final_output = pad_sequences(decoder_final_output, 13,
padding='post', truncating='post')
```

```
del(i)
```

```
2024-03-31 17:38:56.758308: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable
to register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
2024-03-31 17:38:56.758474: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
2024-03-31 17:38:56.916588: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable
to register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
```

```
# decoder_final_output, decoder_final_input, encoder_final, vocab,
inv_vocab

VOCAB_SIZE = len(vocab)
MAX_LEN = 13

print(decoder_final_output.shape, decoder_inp.shape,
encoder_inp.shape, len(vocab), len(inv_vocab), inv_vocab[0])
```

```
(30000, 13) (30000, 13) (30000, 13) 3027 3027 <PAD>
```

```
inv_vocab[16]
```

```
'they'
```

```
#print(len(decoder_final_input), MAX_LEN, VOCAB_SIZE)
#decoder_final_input[0]
#decoder_output_data = np.zeros((len(decoder_final_input), MAX_LEN,
VOCAB_SIZE), dtype="float32")
```

```
#print(decoder_output_data.shape)
#decoder_final_input[80]

from tensorflow.keras.utils import to_categorical
decoder_final_output = to_categorical(decoder_final_output,
len(vocab))

decoder_final_output.shape
```

```
(30000, 13)
```

GLOVE EMBEDDING

```
import numpy as np

embeddings_index = {}

with open('../input/glove6b50d/glove.6B.50d.txt', encoding='utf-8') as
f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

print("Glove Loaded!")
```

```
Glove Loaded!
```

```
embedding_dimention = 50
def embedding_matrix_creater(embedding_dimention, word_index):
    embedding_matrix = np.zeros((len(word_index)+1,
embedding_dimention))
    for word, i in word_index.items():
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
          # words not found in embedding index will be all-zeros.
            embedding_matrix[i] = embedding_vector
    return embedding_matrix
embedding_matrix = embedding_matrix_creater(50, word_index=vocab)
del(embeddings_index)
embedding_matrix.shape
```

```
(3028, 50)
```

```
embedding_matrix[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0.,
```

```
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0.])

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Embedding, LSTM, Input,
Bidirectional, Concatenate, Dropout, Attention

import keras
import tensorflow as tf

!pip install tensorflow

Requirement already satisfied: tensorflow in
/usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.25.2)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (24.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!
=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.20.3)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (4.10.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.36.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.62.1)
Requirement already satisfied: tensorboard<2.16,>=2.15 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.2)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: keras<2.16,>=2.15.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0-
>tensorflow) (0.43.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (1.2.0)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (3.6)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in /usr/local/lib/python3.10/dist-packages (from
tensorboard<2.16,>=2.15->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
>tensorflow) (3.0.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.16,>=2.15->tensorflow) (5.3.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.16,>=2.15->tensorflow) (0.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in
/usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.16,>=2.15->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth-
oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (1.4.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.16,>=2.15->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.16,>=2.15->tensorflow) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.16,>=2.15->tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorboard<2.16,>=2.15->tensorflow) (2024.2.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1-
>tensorboard<2.16,>=2.15->tensorflow) (2.1.5)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in
/usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1-
>google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (0.6.0)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests-
oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.5-
>tensorboard<2.16,>=2.15->tensorflow) (3.2.2)

import tensorflow as tf

enc_inp = tf.keras.layers.Input(shape=(13, ))

enc_inp = Input(shape=(13, ))
enc_model = tf.keras.models.Model(enc_inp, [encoder_outputs,
enc_states])


decoder_state_input_h = tf.keras.layers.Input(shape=( 400 * 2,))
decoder_state_input_c = tf.keras.layers.Input(shape=( 400 * 2,))

decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]


decoder_outputs, state_h, state_c = dec_lstm(dec_embed ,
initial_state=decoder_states_inputs)
decoder_states = [state_h, state_c]

#decoder_output = dec_dense(decoder_outputs)

dec_model = tf.keras.models.Model([dec_inp, decoder_states_inputs],
                                  [decoder_outputs] +
decoder_states)

from tensorflow.keras.layers import Input, LSTM, Embedding, Dense,
TimeDistributed

from tensorflow.keras.layers import Input

from keras.preprocessing.sequence import pad_sequences
print("#########################################")
print("#       start chatting ver. 1.0          #")
print("#########################################")
```

```
prepro1 = ""
while prepro1 != 'q':

    prepro1 = input("you : ")
    try:
        prepro1 = clean_text(prepro1)
        prepro = [prepro1]

        txt = []
        for x in prepro:
            lst = []
            for y in x.split():
                try:
                    lst.append(vocab[y])
                except:
                    lst.append(vocab['<OUT>'])
            txt.append(lst)
        txt = pad_sequences(txt, 13, padding='post')


        ###
        enc_op, stat = enc_model.predict( txt )

        empty_target_seq = np.zeros( ( 1 , 1) )
        empty_target_seq[0, 0] = vocab['<SOS>']
        stop_condition = False
        decoded_translation = ''


        while not stop_condition :

            dec_outputs , h , c = dec_model.predict([ empty_target_seq
] + stat )

            ###
            ############################
            attn_op, attn_state = attn_layer([enc_op, dec_outputs])
            decoder_concat_input = Concatenate(axis=-1)([dec_outputs,
attn_op])
            decoder_concat_input = dec_dense(decoder_concat_input)
            ############################

            sampled_word_index = np.argmax( decoder_concat_input[0, -
1, :] )

            sampled_word = inv_vocab[sampled_word_index] + ' '

            if sampled_word != '<EOS> ':
                decoded_translation += sampled_word
```

```
            if sampled_word == '<EOS> ' or
len(decoded_translation.split()) > 13:
                stop_condition = True

            empty_target_seq = np.zeros( ( 1 , 1 ) )
            empty_target_seq[ 0 , 0 ] = sampled_word_index
            stat = [ h , c ]

        print("chatbot attention : ", decoded_translation )
        print("==============================================")

    except:
        print("sorry didn't got you , please type again :( ")

##########################################
#       start chatting ver. 1.0          #
##########################################
sorry didn't got you , please type again :(
you : hi
sorry didn't got you , please type again :(
```