CHATBOT USING PYTHON

PHASE 3 – DEVELOPMENT PART 1

Introduction

Bots

Bots are specially built software that interacts with internet users automatically. Bots are made up of algorithms that assist them in completing jobs. By auto-designed, we mean that they run on their own, following instructions, and therefore begin the conservation process without the need for human intervention.

Bots are responsible for the majority of internet traffic. For e-commerce sites, traffic can be significantly higher, accounting for up to 90% of total traffic. They can communicate with people and on social media accounts, as well as on websites.

Type of Bots

1. ChatBot — An Artificial Intelligence (AI) programme that communicates with users through app, message, or phone. It is most commonly utilised by Twitter.

2. Social Media Bot- Created for social media sites to answer automatically all at once.

3. Google Bot is commonly used for indexing and crawling. Spider Bots—Developed for retrieving data from websites, the Google Bot is widely used for indexing and crawling.

4. Spam Bots are programmed that automatically send spam emails to a list of addresses.

5. Transnational Bots are bots that are designed to be used in transactions.

6. Monitoring Bots – Creating bots to keep track of the system’s or website’s health.

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
from tqdm import tqdm
from keras.layers import Dense
import json
import re
import string
from sklearn.feature_extraction.text import TfidfVectorizer
import unicodedata
from sklearn.model_selection import train_test_split
```

```
In [2]:question  =[]
       answer = []
        with open("../input/simple-dialogs-for-chatbot/dialogs.txt",'r') as f :
    for line in f :
        line  =  line.split('\t')
        question.append(line[0])
        answer.append(line[1])
print(len(question) == len(answer))
```

```
True
```

```
In [3]:question[:5]
```

```
Out[3]:
['hi, how are you doing?',
 "i'm fine. how about yourself?",
 "i'm pretty good. thanks for asking.",
 'no problem. so how have you been?',
 "i've been great. what about you?"]
```

```
In [4]:
```

```
answer[:5]Out[4]:
```

```
["i'm fine. how about yourself?\n",
 "i'm pretty good. thanks for asking.\n",
 'no problem. so how have you been?\n',
 "i've been great. what about you?\n",
 "i've been good. i'm in school right now.\n"]
```

```
                                                                    In [5]:
answer = [ i.replace("\n","") for i in answer]
```

```
                                                                    In [6]:
answer[:5]
```

```
                                                                    Out[6]:
["i'm fine. how about yourself?",
 "i'm pretty good. thanks for asking.",
 'no problem. so how have you been?',
 "i've been great. what about you?",
 "i've been good. i'm in school right now."]
```

```
                                                                    In [7]:
```

```
data = pd.DataFrame({"question" : question ,"answer":answer})
data.head()
```

Out[7]:

	question	answer
0	hi, how are you doing?	i'm fine. how about yourself?
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.
2	i'm pretty good. thanks for asking.	no problem. so how have you been?
3	no problem. so how have you been?	i've been great. what about you?
4	i've been great. what about you?	i've been good. i'm in school right now.

In [8]:

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

In [9]:

```
linkcode
def clean_text(text):
    text = unicode_to_ascii(text.lower().strip())
    text = re.sub(r"i'm", "i am", text)
    text = re.sub(r"\r", "", text)
    text = re.sub(r"he's", "he is", text)
    text = re.sub(r"she's", "she is", text)
    text = re.sub(r"it's", "it is", text)
    text = re.sub(r"that's", "that is", text)
    text = re.sub(r"what's", "that is", text)
    text = re.sub(r"where's", "where is", text)
    text = re.sub(r"how's", "how is", text)
    text = re.sub(r"\'ll", " will", text)
    text = re.sub(r"\'ve", " have", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"\'d", " would", text)
    text = re.sub(r"\'re", " are", text)
    text = re.sub(r"won't", "will not", text)
    text = re.sub(r"can't", "cannot", text)
    text = re.sub(r"n't", " not", text)
    text = re.sub(r"n'", "ng", text)
    text = re.sub(r"'bout", "about", text)
    text = re.sub(r"'til", "until", text)
    text = re.sub(r"[-()\"#/@;:<>{}`+=~|.!?,]", "", text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = re.sub("(\\W)"," ",text)
    text = re.sub('\S*\d\S*\s*','', text)
    text =  "<sos> " +  text + " <eos>"
    return text
```

In [10]:
```
data["question"][0]
```

Out[10]:
```
'hi, how are you doing?'
```

In [11]:
```
data["question"] = data.question.apply(clean_text)
```

In [12]:
```
data["question"][0]
```

Out[12]:
```
'<sos> hi how are you doing <eos>'
```

In [13]:
```
data["answer"] = data.answer.apply(clean_text)
```

In [14]:
```
question  = data.question.values.tolist()
answer =  data.answer.values.tolist()
```

In [15]:
```
def tokenize(lang):
    lang_tokenizer = tf.keras.preprocessing.text.Tokenizer(
      filters='')
    lang_tokenizer.fit_on_texts(lang)
    tensor = lang_tokenizer.texts_to_sequences(lang)

    tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor,
                                                           padding='post')

    return tensor, lang_tokenizer
```

In [16]:
```
input_tensor , inp_lang  =  tokenize(question)
```

In [17]:
```
target_tensor , targ_lang  =  tokenize(answer)
```

In [18]:
```
 #len(inp_question) ==  len(inp_answer)
```

In [19]:
```
def remove_tags(sentence):
    return sentence.split("<start>")[-1].split("<end>")[0]
```

In [20]:
```
max_length_targ, max_length_inp = target_tensor.shape[1], input_tensor.shape[1]
```

In [21]:
```
# Creating training and validation sets using an 80-20 split
input_tensor_train, input_tensor_val, target_tensor_train, target_tensor_val = tra
in_test_split(input_tensor, target_tensor, test_size=0.2)
```

In [22]:
```
#print(len(train_inp) , len(val_inp) , len(train_target) , len(val_target))
```

In [23]:

```
BUFFER_SIZE = len(input_tensor_train)
BATCH_SIZE = 64
steps_per_epoch = len(input_tensor_train)//BATCH_SIZE
embedding_dim = 256
units = 1024
vocab_inp_size = len(inp_lang.word_index)+1
vocab_tar_size = len(targ_lang.word_index)+1

dataset = tf.data.Dataset.from_tensor_slices((input_tensor_train, target_tensor_tr
ain)).shuffle(BUFFER_SIZE)
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)

example_input_batch, example_target_batch = next(iter(dataset))
example_input_batch.shape, example_target_batch.shape
```

Out[23]:

```
(TensorShape([64, 22]), TensorShape([64, 22]))
```

In [24]:

```
class Encoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
        super(Encoder, self).__init__()
        self.batch_sz = batch_sz
        self.enc_units = enc_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.enc_units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')

    def call(self, x,hidden):
        x = self.embedding(x)
        output, state = self.gru(x, initial_state = hidden)
        return output, state

    def initialize_hidden_state(self):
        return tf.zeros((self.batch_sz, self.enc_units))
```

In [25]:

```
encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE)

# sample input
sample_hidden = encoder.initialize_hidden_state()
sample_output, sample_hidden = encoder(example_input_batch, sample_hidden)
print ('Encoder output shape: (batch size, sequence length, units) {}'.format(samp
le_output.shape))
print ('Encoder Hidden state shape: (batch size, units) {}'.format(sample_hidden.s
hape))
```

```
Encoder output shape: (batch size, sequence length, units) (64, 22, 1024)
Encoder Hidden state shape: (batch size, units) (64, 1024)
```

In [26]:

```
class BahdanauAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)
    def call(self, query, values):
        # query hidden state shape == (batch_size, hidden size)
        # query_with_time_axis shape == (batch_size, 1, hidden size)
        # values shape == (batch_size, max_len, hidden size)
        # we are doing this to broadcast addition along the time axis to calculate
the score
        query_with_time_axis = tf.expand_dims(query, 1)

        # score shape == (batch_size, max_length, 1)
        # we get 1 at the last axis because we are applying score to self.V
        # the shape of the tensor before applying self.V is (batch_size, max_lengt
h, units)
        score = self.V(tf.nn.tanh(
            self.W1(query_with_time_axis) + self.W2(values)))

        # attention_weights shape == (batch_size, max_length, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * values
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights
```

In [27]:

```
attention_layer = BahdanauAttention(10)
attention_result, attention_weights = attention_layer(sample_hidden, sample_output
)

print("Attention result shape: (batch size, units) {}".format(attention_result.sha
pe))
print("Attention weights shape: (batch_size, sequence_length, 1) {}".format(attent
ion_weights.shape))
```

```
Attention result shape: (batch size, units) (64, 1024)
Attention weights shape: (batch_size, sequence_length, 1) (64, 22, 1)
```

In [28]:

```
class Decoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
        super(Decoder, self).__init__()
        self.batch_sz = batch_sz
        self.dec_units = dec_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.dec_units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')
        self.fc = tf.keras.layers.Dense(vocab_size)

        # used for attention
        self.attention = BahdanauAttention(self.dec_units)

    def call(self, x, hidden, enc_output):
        # enc_output shape == (batch_size, max_length, hidden_size)
        context_vector, attention_weights = self.attention(hidden, enc_output)

        # x shape after passing through embedding == (batch_size, 1, embedding_dim
)
        x = self.embedding(x)
        # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_si
ze)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
        output, state = self.gru(x)

        # output shape == (batch_size * 1, hidden_size)
        output = tf.reshape(output, (-1, output.shape[2]))

        # output shape == (batch_size, vocab)
        x = self.fc(output)

        return x, state, attention_weights
```

In [29]:

```
decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH_SIZE)

sample_decoder_output, _, _ = decoder(tf.random.uniform((BATCH_SIZE, 1)),
                                      sample_hidden, sample_output)

print ('Decoder output shape: (batch_size, vocab size) {}'.format(sample_decoder_o
utput.shape))
```

```
Decoder output shape: (batch_size, vocab size) (64, 2347)
```

In [30]:

```
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')

def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)
```

In [31]:

```
@tf.function
def train_step(inp, targ, enc_hidden):
    loss = 0

    with tf.GradientTape() as tape:
        enc_output, enc_hidden = encoder(inp, enc_hidden)

        dec_hidden = enc_hidden

        dec_input = tf.expand_dims([targ_lang.word_index['<sos>']] * BATCH_SIZE, 1
)

        # Teacher forcing - feeding the target as the next input
        for t in range(1, targ.shape[1]):
            # passing enc_output to the decoder
            predictions, dec_hidden, _ = decoder(dec_input, dec_hidden, enc_output
)

            loss += loss_function(targ[:, t], predictions)
            # using teacher forcing
            dec_input = tf.expand_dims(targ[:, t], 1)

    batch_loss = (loss / int(targ.shape[1]))

    variables = encoder.trainable_variables + decoder.trainable_variables

    gradients = tape.gradient(loss, variables)

    optimizer.apply_gradients(zip(gradients, variables))

    return batch_loss
```

Conclusion

This article is the base of knowledge of the definition of ChatBot, its importance in the Business, and how we can build a simple Chatbot by using Python and Library Chatterbot.