PHASE 4

BUILD A MODEL IN CHATBOT

Date	25.10.2023
Team ID	Proj_212175_Team-3
Project Name	Create a Chatbot in Python
Maximum Marks	

ABSTRACT :

The primary components of building a chatbot model include data collection, preprocessing, model architecture design, training, and deployment. Natural language understanding and generation techniques play a pivotal role in enabling the chatbot to comprehend and generate human-like responses.

Data is collected from various sources, and preprocessing involves tasks like tokenization, stemming, and removing noise. Model architecture design choices encompass selecting between rule-based, retrieval-based, or generative models, each with its own strengths and weaknesses.

CODE :

```
tf.keras.utils.plot_model(chatbot.decoder,to_file='decoder.png',s
how _shapes=True,show_layer_activations=True)

for idx,i in enumerate(model.layers):

 print('Encoder layers:' if idx==0 else 'Decoder layers: ')
```

for j in i.layers:

 print(j)

 print('---------------------')

class ChatBot(tf.keras.models.Model):

 def _init_(self,base_encoder,base_decoder,*args,**kwargs):

PHASE 4

 super()._init_(*args,**kwargs)

self.encoder,self.decoder=self.build_inference_model(base_enco der,b ase_decoder)

 def

build_inference_model(self,base_encoder,base_decoder):

encoder_inputs=tf.keras.Input(shape=(None,))

 x=base_encoder.layers[0](encoder_inputs)

 x=base_encoder.layers[1](x)

 x,encoder_state_h,encoder_state_c=base_encoder.layers[2](x)

encoder=tf.keras.models.Model(inputs=encoder_inputs,outputs =[en coder_state_h,encoder_state_c],name='chatbot_encoder')

 decoder_input_state_h=tf.keras.Input(shape=(lstm_cells,))

```
decoder_input_state_c=tf.keras.Input(shape=(lstm_cells,))

decoder_inputs=tf.keras.Input(shape=(None,))

 x=base_decoder.layers[0](decoder_inputs)

 x=base_encoder.layers[1](x)


x,decoder_state_h,decoder_state_c=base_decoder.layers[2](x,initi
al_ state=[decoder_input_state_h,decoder_input_state_c])

 decoder_outputs=base_decoder.layers[-1](x)

 decoder=tf.keras.models.Model(
```

PHASE 4

```
inputs=[decoder_inputs,[decoder_input_state_h,decoder_input_st
at e_c]],


outputs=[decoder_outputs,[decoder_state_h,decoder_state_c]],na
me ='chatbot_decoder'

 )

 return encoder,decoder


 def summary(self):

 self.encoder.summary()

 self.decoder.summary()


 def softmax(self,z):
```

```
return np.exp(z)/sum(np.exp(z))


def sample(self,conditional_probability,temperature=0.5):
conditional_probability =
np.asarray(conditional_probability).astype("float64")
conditional_probability = np.log(conditional_probability) /
temperature
reweighted_conditional_probability =
self.softmax(conditional_probability)
probas = np.random.multinomial(1,
reweighted_conditional_probability, 1)
```



```
return np.argmax(probas)


def preprocess(self,text):
text=clean_text(text)
seq=np.zeros((1,max_sequence_length),dtype=np.int32)
for i,word in enumerate(text.split()):
seq[:,i]=sequences2ids(word).numpy()[0]
return seq


def postprocess(self,text):
text=re.sub(' - ','-',text.lower())
```

```
text=re.sub(' [.] ','. ',text)

text=re.sub(' [1] ','1',text)

text=re.sub(' [2] ','2',text)

text=re.sub(' [3] ','3',text)

text=re.sub(' [4] ','4',text)

text=re.sub(' [5] ','5',text)

text=re.sub(' [6] ','6',text)

text=re.sub(' [7] ','7',text)

text=re.sub(' [8] ','8',text)

text=re.sub(' [9] ','9',text)

text=re.sub(' [0] ','0',text)

text=re.sub(' [,] ',', ',text)

text=re.sub(' [?] ','? ',text)

text=re.sub(' [!] ','! ',text)

text=re.sub(' [$] ','$ ',text)

text=re.sub(' [&] ','& ',text)

text=re.sub(' [/] ','/ ',text)

text=re.sub(' [:] ',': ',text)

text=re.sub(' [;] ','; ',text)

text=re.sub(' [] ',' ',text)

text=re.sub(' [\'] ','\'',text)
```

```
 text=re.sub(' [\"] ','\"',text)

 return text


 def call(self,text,config=None):

 input_seq=self.preprocess(text)

 states=self.encoder(input_seq,training=False)

 target_seq=np.zeros((1,1))

 target_seq[:,:]=sequences2ids(['<start>']).numpy()[0][0]

stop_condition=False

 decoded=[]

 while not stop_condition:


decoder_outputs,new_states=self.decoder([target_seq,states],tra
inin g=False)
```

PHASE 4

```
# index=tf.argmax(decoder_outputs[:,-1,:],axis=-
1).numpy().item()

 index=self.sample(decoder_outputs[0,0,:]).item()

word=ids2sequences([index])

 if word=='<end> ' or len(decoded)>=max_sequence_length:

stop_condition=True

 else:
```

```
decoded.append(index)
target_seq=np.zeros((1,1))
target_seq[:,:]=index
states=new_states
return self.postprocess(ids2sequences(decoded))

chatbot=ChatBot(model.encoder,model.decoder,name='chatb
ot') chatbot.summary()

tf.keras.utils.plot_model(chatbot.encoder,to_file='encoder.png',s
how _shapes=True,show_layer_activations=True)

tf.keras.utils.plot_model(chatbot.decoder,to_file='decoder.png',s
how _shapes=True,show_layer_activations=True)
```

EXPLANATION :

- **tf.keras.utils.plot_model(chatbot.decoder, to_file='decoder.png', show_shapes=True, show_layer_activations=True):**

  This code generates a visualization of the chatbot.decoder model and saves it as 'decoder.png'. Here's what each argument does:

- **chatbot.decoder: This is presumably a Keras model (a** neural network) that you want to visualize.

- to_file='decoder.png': It specifies the filename where the visualization of the model will be saved as an image (in this case, 'decoder.png').
- show_shapes=True: This flag indicates that you want to display the shapes of the layers in the visualization.
- show_layer_activations=True: This flag indicates that you want to show the layer activations in the visualization. • The code then seems to iterate through the layers of the model (which is not defined in the provided code, but I assume it's a reference to the chatbot.decoder model).

Inside the loop, it prints information about the layers. It differentiates between 'Encoder layers' and 'Decoder layers' based on the idx variable. This suggests that the model might have an encoder-decoder architecture (common in sequence-to-sequence models or chatbots).

It further iterates through the layers within the current i (which is presumably a layer or a model) and prints information about these sub-layers.

PHASE 4

- <u>tf.keras.utils.plot_model(chatbot.encoder, to_file='encoder.png', show_shapes=True, show_layer_activations=True)</u>

It visualizes the architecture of the chatbot.encoder

model. Saves the visualization as 'encoder.png'.

It includes information about the shapes of the layers and shows layer activations in the visualization.

- **tf.keras.utils.plot_model(chatbot.decoder, to_file='decoder.png', show_shapes=True, show_layer_activations=True)**

It visualizes the architecture of the chatbot.decoder

model. Saves the visualization as 'decoder.png'.

It also includes information about the shapes of the layers and shows layer activations in the visualization.

OUTPUT :

Encoder layers:
<keras.layers.core.embedding.Embedding object at 0x782084b9d19 0>
<keras.layers.normalization.layer_normalization.LayerNormaliza ti on object at 0x7820e56f1b90>
<keras.layers.rnn.lstm.LSTM object at 0x7820841bd650> ---------------------
Decoder layers:

<keras.layers.core.embedding.Embedding object at 0x78207c258590 >
<keras.layers.normalization.layer_normalization.LayerNormaliza

```
ti on object at 0x78207c78bd10>
<keras.layers.rnn.lstm.LSTM object at
0x78207c258a10> <keras.layers.core.dense.Dense
object at 0x78207c2636d0> --------------------



Model: "chatbot_encoder"
_________________________________________________________________ _______
 Layer (type) Output Shape Param #
==========================================================
===============
 input_1 (InputLayer) [(None, None)] 0

 encoder_embedding (Embeddin (None, None, 256) 625408
 g)

 layer_normalization (LayerN (None, None, 256) 512
 ormalization)

 encoder_lstm (LSTM) [(None, None, 256), 525312
(None, 256),
 (None, 256)]

==========================================================
===============
Total params: 1,151,232
Trainable params: 1,151,232
Non-trainable params: 0
_________________________________________________________________ _______
```

Model: "chatbot_decoder"
____
____

Layer (type) Output Shape Param # Connected to
==
==

input_4 (InputLayer) [(None, None)] 0 []

decoder_embedding (Embedding) (None, None, 256) 625408 ['i
nput_4[0][0]']

layer_normalization (LayerNorm (None, None, 256) 512 ['dec
oder_embedding[0][0]']
alization)

input_2 (InputLayer) [(None, 256)] 0 []

input_3 (InputLayer) [(None, 256)] 0 []

decoder_lstm (LSTM) [(None, None, 256), 525312 ['layer_
normalization[1][0]',
(None, 256), 'input_2[0][0]', (None, 256)] 'input_3[0][0]']
decoder_dense (Dense) (None, None, 2443) 627851 ['decod
er_lstm[0][0]']

==
== Total
params: 1,779,083
Trainable params: 1,779,083

PHASE 4

Non-trainable params: 0

____
____

CONCLUSION :

Building a chatbot model involves several key steps:

Data Collection: Gather relevant training data, including conversations, questions, and answers.

Data Preprocessing: Clean and prepare the data, which often includes tokenization, stemming, and removing noise.

Model Selection: Choose an appropriate architecture, such as a sequence-to-sequence model or transformer model.

PHASE 4

Training: Train the model on your preprocessed data using appropriate loss functions and optimizers.

Evaluation: Assess the model's performance using metrics like accuracy, BLEU score, or human evaluation.

Fine-Tuning: Refine the model based on evaluation results, adjusting hyperparameters and data.

Deployment: Deploy the chatbot, whether it's in a web application, messaging platform, or other interfaces.

Monitoring and Maintenance: Continuously monitor and update the chatbot to improve its responses and keep it up to-date.

Building a chatbot model requires a combination of data handling, machine learning expertise, and domain knowledge to create an effective and conversational AI system.