GRT INSITITUTE OF ENGINEERING ANDTECHNOLOGY-TIRUTTANI-631209

Affiliated to Anna University, Chennai, An ISO9001:2015certifiedinstitution.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

NAAN MUDHALVAN-IBM(AI) PROJECT

IBM AL 101 ARTIFICIAL INTELLIGENCE-GROUP 1(TEAM 5)

PROJECT TITLE:

CREATE A CHATBOT USING PYTHON

PHASE 3:

DEVELOPMENT PART 1

Submitted by:

NAME: DAYANIDHI N
NM ID: au110321106007
Year/sem: III / V
COLLEGE CODE: 1103
Email id: dayanidhi182004@gmail.com

Project Details and Technology:

Project Name	CREATE A CHATBOT USING PYTHON
Abstract	A chatbot is a computer program that simulates human conversation through voice commands or text chats or both.
Language/s Used:	Python (GUI Based)
Python version	3.8 or 3.9

About The Chatbot in Python:

- The Chatbot Project In Python is written in the Python programming language, and it will shows how to code a chatbot in Python.

- A Chatbot Python is a piece of intelligent software that can communicate and conduct tasks in the same way that a human can. Customer interaction, social media marketing, and instant messaging are all common uses for chatbots in Python Project Report.

Prerequisite:

- The project requires we have good knowledge in Python, Keras, and Natural language processing (NLTK). Along with them, we will use some helping modules which we can download using the python-pip command.

How To Make A chatbot In Python?

- Now we are going to build the chatbot using Python but first, let us see the file structure and the type of files we will be creating:

  - **Intents.json** – The data file which has predefined patterns and responses.
  - **train_chatbot.py** – In this Python file, we wrote a script to build the model and train our chatbot.
  - **Words.pkl** – This is a pickle file in which we store the words Python object that contains a list of our vocabulary.

- **Classes.pkl** – The classes pickle file contains the list of categories.
- **Chatbot_model.h5** – This is the trained model that contains information about the model and has weights of the neurons.
- **Chatgui.py** – This is the Python script in which we implemented GUI for our chatbot. Users can easily interact with the bot.

6 steps to create a chatbot in Python from scratch:

1. Import and load the data file
2. Preprocess data
3. Create training and testing data
4. Build the model
5. Predict the response
6. Run the chatbot

Step 1:Import and load the data file:

➢ First, make a file name as train_chatbot.py. We import the necessary packages for our chatbot and initialize the variables we will use in our Python project.

➢ The data file is in JSON format so we used the json package to parse the JSON file into Python.

Code:

```
import nltk
from nltk.stem import WordNetLemmatizer lemmatizer =
WordNetLemmatizer()
import json import
pickle import numpy as
np
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import SGD from
tensorflow.keras.optimizers import SGD
import random
words=[]
classes = [] documents = []
ignore_words = ['?', '!']
data_file = open('intents.json').read() intents =
json.loads(data_file)
```

Step 2:Preprocess data:

- When working with text data, we need to perform various preprocessing on the data before we make a machine learning or a deep learning model. Based on the requirements we need to apply various operations to preprocess the data.

- Tokenizing is the most basic and first thing you can do on text data. Tokenizing is the process of breaking the whole text into small parts like words.

- Here we iterate through the patterns and tokenize the sentence using nltk.word_tokenize() function and append each word in the words list. We also create a list of classes for our tags.

<u>Code:</u>

```
for intent in intents['intents']: for
pattern in intent['patterns']:
#tokenize each word w =
nltk.word_tokenize(pattern)
words.extend(w)
#add documents in the corpus documents.append((w,
intent['tag']))
# add to our classes list if
intent['tag'] not in classes:
classes.append(intent['tag'])
```

- Now we will lemmatize each word and remove duplicate words from the list. Lemmatizing is the process of converting a word into its lemma form and then creating a pickle file to store the Python objects which we will use while predicting.

<u>Code:</u>

```
# lemmatize, lower each word and remove duplicates words =
[lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words] words =
sorted(list(set(words)))
# sort classes
classes = sorted(list(set(classes)))
# documents = combination between patterns and intents print
(len(documents), "documents")
# classes = intents print (len(classes),
"classes", classes) # words = all words,
vocabulary print (len(words), "unique
lemmatized words", words)
pickle.dump(words,open('words.pkl','w
b'))
```

```
pickle.dump(classes,open('classes.pkl','
wb'))
```

Step 3:Create training and testing data:

- Now, we will create the training data in which we will provide the input and the output. Our input will be the pattern and output will be the class our input pattern belongs to. But the computer doesn't understand text so we will convert text into numbers.

Code:

```
# create our training data
training = []
# create an empty array for our output output_empty =
[0] * len(classes)
# training set, bag of words for each sentence for
doc in documents: # initialize our bag of words
bag = []
# list of tokenized words for the pattern
pattern_words = doc[0]
# lemmatize each word - create base word, in attempt to represent related words
pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words] #
create our bag of words array with 1, if word match found in current pattern for w in
words: bag.append(1) if w in pattern_words else bag.append(0)
# output is a '0' for each tag and '1' for current tag (for each pattern)
output_row = list(output_empty) output_row[classes.index(doc[1])] = 1
training.append([bag, output_row]) # shuffle our features and turn into
np.array random.shuffle(training) training = np.array(training)
# create train and test lists. X - patterns, Y - intents
train_x = list(training[:,0]) train_y = list(training[:,1])
print("Training data created")
```

Step 4:Build the model:

- We have our training data ready, now we will build a deep neural network that has 3 layers. We use the Keras sequential API for this. After training the model for 200 epochs, we achieved 100% accuracy on our model. Let us save the model as 'chatbot_model.h5'.

Code:

```
# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd output layer contains number of
neurons
# equal to number of intents to predict output intent with softmax model
=Sequential()
```

```
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu')) model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))
# Compile model. Stochastic gradient descent with Nesterov accelerated gradient gives good results for this model
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
#fitting and saving the model
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)
model.save('chatbot_model.h5', hist) print("model created")
```

Step 5:Predict the response:

- To predict the sentences and get a response from the user to let us create a new file 'chatapp.py'.
- We will load the trained model and then use a graphical user interface that will predict the response from the bot. The model will only tell us the class it belongs to, so we will implement some functions which will identify the class and then retrieve us a random response from the list of responses.
- Again we import the necessary packages and load the 'words.pkl' and 'classes.pkl' pickle files which we have created when we trained our model:

Code:

```
import nltk
from nltk.stem import WordNetLemmatizer lemmatizer =
WordNetLemmatizer()
import pickle import
numpy as np
from keras.models import load_model
model = load_model('chatbot_model.h5')
import json import random
intents = json.loads(open('intents.json').read())
words = pickle.load(open('words.pkl','rb')) classes =
pickle.load(open('classes.pkl','rb'))
```

- To predict the class, we will need to provide input in the same way as we did while training. So we will create some functions that will perform text preprocessing and then predict the class.

Code:

```
def clean_up_sentence(sentence):
```

```
# tokenize the pattern - split words into array
sentence_words = nltk.word_tokenize(sentence) #
stem each word - create short form for word
sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words] return
sentence_words
# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence def
bow(sentence, words, show_details=True):
# tokenize the pattern sentence_words =
clean_up_sentence(sentence) # bag of words - matrix of
N words, vocabulary matrix bag = [0]*len(words) for s in
sentence_words: for i,w in enumerate(words): if w == s:
# assign 1 if current word is in the vocabulary position
bag[i] = 1 if show_details:
print ("found in bag: %s" % w)
return(np.array(bag)) def predict_class(sentence,
model): # filter out predictions below a threshold
p = bow(sentence, words,show_details=False)
res = model.predict(np.array([p]))[0] ERROR_THRESHOLD =
0.25
results = [[i,r] for i,r in enumerate(res) if r>ERROR_THRESHOLD]
# sort by strength of probability
results.sort(key=lambda x: x[1], reverse=True) return_list = [] for r in
results: return_list.append({"intent": classes[r[0]], "probability":
str(r[1])}) return return_list
```

- After predicting the class, we will get a random response from the list of intents.

<u>Code:</u>

```
 def getResponse(ints, intents_json): tag =
ints[0]['intent'] list_of_intents =
intents_json['intents'] for i in
list_of_intents: if(i['tag']== tag):
result = random.choice(i['responses'])
break return result def
chatbot_response(text): ints =
predict_class(text, model) res =
getResponse(ints, intents) return res
```

- Now we will develop a graphical user interface. Let’s use Tkinter library which is shipped with tons of useful libraries for GUI. We will take the input message from the user and then use the helper functions we have created to get the response from the bot and display it on the GUI. Here is the full source code for the GUI.

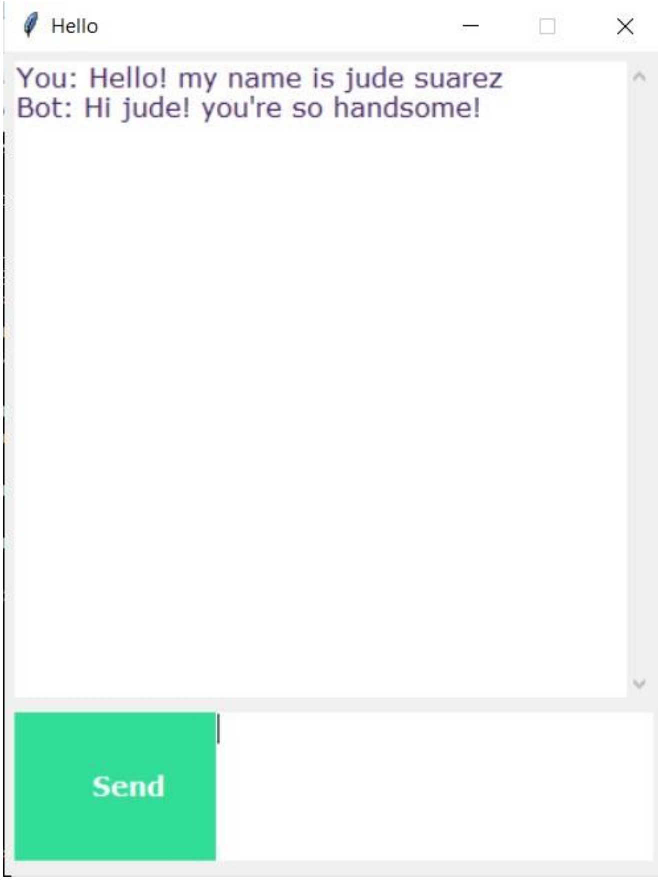
<u>Code:</u>

```
#Creating GUI with tkinter import tkinter
from tkinter import * def send(): msg =
```

```
EntryBox.get("1.0",'end-1c').strip()
EntryBox.delete("0.0",END) if msg != '':
ChatLog.config(state=NORMAL)
ChatLog.insert(END, "You: " + msg + '\n\n')
ChatLog.config(foreground="#442265", font=("Verdana", 12 )) res =
chatbot_response(msg)
ChatLog.insert(END, "Bot: " + res + '\n\n')
ChatLog.config(state=DISABLED)
ChatLog.yview(END) base =
Tk()
base.title("Hello") base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)
#Create Chat window
ChatLog = Text(base, bd=0, bg="white", height="8", width="50", font="Arial",)
ChatLog.config(state=DISABLED) #Bind
scrollbar to Chat window
scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")
ChatLog['yscrollcommand'] = scrollbar.set
#Create Button to send message
SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12", height=5,
bd=0, bg="#32de97", activebackground="#3c9d9b",fg='#ffffff', command= send )
#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")
#EntryBox.bind("<Return>", send) #Place all
components on the screen
scrollbar.place(x=376,y=6, height=386)
ChatLog.place(x=6,y=6, height=386, width=370)
EntryBox.place(x=128, y=401, height=90, width=265)
SendButton.place(x=6, y=401, height=90) base.mainloop()
```

Step 6:Run the chatbot:

- To run the chatbot, we have two main files;
  1.train_chatbot.py
  2.chatapp.py.
- First, we train the model using the command in the terminal: python train_chatbot.py
- If we don’t see any error during training, we have successfully created the model. Then to run the app, we run the second file. python chatgui.py
- The program will open up a GUI window within a few seconds. With the GUI you can easily chat with the bot.


Hello
You: Hello! my name is jude suarez
Bot: Hi jude! you're so handsome!
Send

FULL SOURCE CODE FOR THIS PROJECT:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import re,string
data = {
    'question': [
        "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?"
    ],
    'answer': [
        "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."
    ]
}
df = pd.DataFrame(data)
print(df)
df['question tokens']=df['question'].apply(lambda x:len(x.split()))
```

```
df['answer tokens']=df['answer'].apply(lambda x:len(x.split()))

plt.style.use('fivethirtyeight')

fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))

sns.set_palette('Set2')

sns.histplot(x=df['question tokens'],data=df,kde=True,ax=ax[0])

sns.histplot(x=df['answer tokens'],data=df,kde=True,ax=ax[1])

sns.jointplot(x='question tokens',y='answer tokens',data=df,kind='kde',fill=True,cmap='YlGnBu')

plt.show()

def clean_text(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)
    text=re.sub('\t',' ',text)
    return text
df.drop(columns=['answer tokens','question tokens'],axis=1,inplace=True)
df['encoder_inputs']=df['question'].apply(clean_text)
df['decoder_targets']=df['answer'].apply(clean_text)+' <end>'
df['decoder_inputs']='<start> '+df['answer'].apply(clean_text)+' <end>'
df.head(10)
df['encoder input tokens']=df['encoder_inputs'].apply(lambda x:len(x.split()))
df['decoder input tokens']=df['decoder_inputs'].apply(lambda x:len(x.split()))
df['decoder target tokens']=df['decoder_targets'].apply(lambda x:len(x.split()))
plt.style.use('fivethirtyeight')
fig,ax=plt.subplots(nrows=1,ncols=3,figsize=(20,5))
sns.set_palette('Set2')
sns.histplot(x=df['encoder input tokens'],data=df,kde=True,ax=ax[0])
sns.histplot(x=df['decoder input tokens'],data=df,kde=True,ax=ax[1])
sns.histplot(x=df['decoder target tokens'],data=df,kde=True,ax=ax[2])
sns.jointplot(x='encoder input tokens',y='decoder target
tokens',data=df,kind='kde',fill=True,cmap='YlGnBu')
plt.show()
print(f"After preprocessing: {' '.join(df[df['encoder input tokens'].max()==df['encoder input
tokens']]['encoder_inputs'].values.tolist())}")
print(f"Max encoder input length: {df['encoder input tokens'].max()}")
print(f"Max decoder input length: {df['decoder input tokens'].max()}")
print(f"Max decoder target length: {df['decoder target tokens'].max()}")
```

```
df.drop(columns=['question','answer','encoder input tokens','decoder input tokens','decoder target tokens'],axis=1,inplace=True)

params={

    "vocab_size":2500,

    "max_sequence_length":30,

    "learning_rate":0.008,

    "batch_size":149,

    "lstm_cells":256,

    "embedding_dim":256,

    "buffer_size":10000

}

learning_rate=params['learning_rate']

batch_size=params['batch_size']

embedding_dim=params['embedding_dim']

lstm_cells=params['lstm_cells']

vocab_size=params['vocab_size']

max_sequence_length=params['max_sequence_length']

df.head(10)

vectorizelayer=TextVectorization(

max_tokens=vocab_size,

standardize=None,

output_mode='int',

output_sequence_length=max_sequence_length

)

vectorize_layer.adapt(df['encoder_inputs']+' '+df['decoder_targets']+' <start> <end>')

vocab_size=len(vectorize_layer.get_vocabulary())

print(f'Vocab size: {len(vectorize_layer.get_vocabulary())}')

print(f'{vectorize_layer.get_vocabulary()[:12]}')

def sequences2ids(sequence):
```

```
    return vectorize_layer(sequence)


def ids2sequences(ids):
    decode=''
    if type(ids)==int:
        ids=[ids]
    for id in ids:
        decode+=vectorize_layer.get_vocabulary()[id]+' '
    return decode


x=sequences2ids(df['encoder_inputs'])
yd=sequences2ids(df['decoder_inputs'])
y=sequences2ids(df['decoder_targets'])


print(f'Question sentence: hi , how are you ?')
print(f'Question to tokens: {sequences2ids("hi , how are you ?")[:10]}')
print(f'Encoder input shape: {x.shape}')
print(f'Decoder input shape: {yd.shape}')
print(f'Decoder target shape: {y.shape}')
print(f'Encoder input: {x[0][:12]} ...')
print(f'Decoder input: {yd[0][:12]} ...')    # shifted by one time step of the target as input to decoder is the
output of the previous timestep
print(f'Decoder target: {y[0][:12]} ...')
data=tf.data.Dataset.from_tensor_slices((x,yd,y))
data=data.shuffle(buffer_size)
class Encoder(tf.keras.models.Model):
    def __init__(self,units,embedding_dim,vocab_size,*args,**kwargs) -> None:
        super().__init__(*args,**kwargs)
        self.units=units
```

```
        self.vocab_size=vocab_size
        self.embedding_dim=embedding_dim
        self.embedding=Embedding(
            vocab_size,
            embedding_dim,
            name='encoder_embedding',
            mask_zero=True,
            embeddings_initializer=tf.keras.initializers.GlorotNormal()
        )
        self.normalize=LayerNormalization()
        self.lstm=LSTM(
            units,
            dropout=.4,
            return_state=True,
            return_sequences=True,
            name='encoder_lstm',
            kernel_initializer=tf.keras.initializers.GlorotNormal()
        )

    def call(self,encoder_inputs):
        self.inputs=encoder_inputs
        x=self.embedding(encoder_inputs)
        x=self.normalize(x)
        x=Dropout(.4)(x)
        encoder_outputs,encoder_state_h,encoder_state_c=self.lstm(x)
        self.outputs=[encoder_state_h,encoder_state_c]
        return encoder_state_h,encoder_state_c
```

```
encoder=Encoder(lstm_cells,embedding_dim,vocab_size,name='encoder')
encoder.call(_[0])

train_data=data.take(int(.9*len(data)))
train_data=train_data.cache()
train_data=train_data.shuffle(buffer_size)
train_data=train_data.batch(batch_size)
train_data=train_data.prefetch(tf.data.AUTOTUNE)
train_data_iterator=train_data.as_numpy_iterator()

val_data=data.skip(int(.9*len(data))).take(int(.1*len(data)))
val_data=val_data.batch(batch_size)
val_data=val_data.prefetch(tf.data.AUTOTUNE)

_=train_data_iterator.next()
print(f'Number of train batches: {len(train_data)}')
print(f'Number of training data: {len(train_data)*batch_size}')
print(f'Number of validation batches: {len(val_data)}')
print(f'Number of validation data: {len(val_data)*batch_size}')
print(f'Encoder Input shape (with batches): {_[0].shape}')
print(f'Decoder Input shape (with batches): {_[1].shape}')
print(f'Target Output shape (with batches): {_[2].shape}')
class Decoder(tf.keras.models.Model):
    def __init__(self,units,embedding_dim,vocab_size,*args,**kwargs) -> None:
        super().__init__(*args,**kwargs)
        self.units=units
        self.embedding_dim=embedding_dim
        self.vocab_size=vocab_size
```

```
    self.embedding=Embedding(
      vocab_size,
      embedding_dim,
      name='decoder_embedding',
      mask_zero=True,
      embeddings_initializer=tf.keras.initializers.HeNormal()
    )
    self.normalize=LayerNormalization()
    self.lstm=LSTM(
      units,
      dropout=.4,
      return_state=True,
      return_sequences=True,
      name='decoder_lstm',
      kernel_initializer=tf.keras.initializers.HeNormal()
    )
    self.fc=Dense(
      vocab_size,
      activation='softmax',
      name='decoder_dense',
      kernel_initializer=tf.keras.initializers.HeNormal()
    )

  def call(self,decoder_inputs,encoder_states):
    x=self.embedding(decoder_inputs)
    x=self.normalize(x)
    x=Dropout(.4)(x)
    x,decoder_state_h,decoder_state_c=self.lstm(x,initial_state=encoder_states)
```

```
        x=self.normalize(x)
        x=Dropout(.4)(x)
        return self.fc(x)

decoder=Decoder(lstm_cells,embedding_dim,vocab_size,name='decoder')
decoder(_[1][:1],encoder(_[0][:1]))
class ChatBotTrainer(tf.keras.models.Model):
    def __init__(self,encoder,decoder,*args,**kwargs):
        super().__init__(*args,**kwargs)
        self.encoder=encoder
        self.decoder=decoder

    def loss_fn(self,y_true,y_pred):
        loss=self.loss(y_true,y_pred)
        mask=tf.math.logical_not(tf.math.equal(y_true,0))
        mask=tf.cast(mask,dtype=loss.dtype)
        loss*=mask
        return tf.reduce_mean(loss)

    def accuracy_fn(self,y_true,y_pred):
        pred_values = tf.cast(tf.argmax(y_pred, axis=-1), dtype='int64')
        correct = tf.cast(tf.equal(y_true, pred_values), dtype='float64')
        mask = tf.cast(tf.greater(y_true, 0), dtype='float64')
        n_correct = tf.keras.backend.sum(mask * correct)
        n_total = tf.keras.backend.sum(mask)
        return n_correct / n_total

    def call(self,inputs):
```

```
        encoder_inputs,decoder_inputs=inputs
        encoder_states=self.encoder(encoder_inputs)
        return self.decoder(decoder_inputs,encoder_states)

    def train_step(self,batch):
        encoder_inputs,decoder_inputs,y=batch
        with tf.GradientTape() as tape:
            encoder_states=self.encoder(encoder_inputs,training=True)
            y_pred=self.decoder(decoder_inputs,encoder_states,training=True)
            loss=self.loss_fn(y,y_pred)
            acc=self.accuracy_fn(y,y_pred)

        variables=self.encoder.trainable_variables+self.decoder.trainable_variables
        grads=tape.gradient(loss,variables)
        self.optimizer.apply_gradients(zip(grads,variables))
        metrics={'loss':loss,'accuracy':acc}
        return metrics

    def test_step(self,batch):
        encoder_inputs,decoder_inputs,y=batch
        encoder_states=self.encoder(encoder_inputs,training=True)
        y_pred=self.decoder(decoder_inputs,encoder_states,training=True)
        loss=self.loss_fn(y,y_pred)
        acc=self.accuracy_fn(y,y_pred)
        metrics={'loss':loss,'accuracy':acc}
        return metrics
    model=ChatBotTrainer(encoder,decoder,name='chatbot_trainer')
model.compile(
```

```
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
    weighted_metrics=['loss','accuracy']
)
model(_[:2])
history=model.fit(
    train_data,
    epochs=100,
    validation_data=val_data,
    callbacks=[
        tf.keras.callbacks.TensorBoard(log_dir='logs'),
        tf.keras.callbacks.ModelCheckpoint('ckpt',verbose=1,save_best_only=True)
    ]
)
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))
ax[0].plot(history.history['loss'],label='loss',c='red')
ax[0].plot(history.history['val_loss'],label='val_loss',c = 'blue')
ax[0].set_xlabel('Epochs')
ax[1].set_xlabel('Epochs')
ax[0].set_ylabel('Loss')
ax[1].set_ylabel('Accuracy')
ax[0].set_title('Loss Metrics')
ax[1].set_title('Accuracy Metrics')
ax[1].plot(history.history['accuracy'],label='accuracy')
ax[1].plot(history.history['val_accuracy'],label='val_accuracy')
ax[0].legend()
ax[1].legend()
plt.show()
```

```
model.load_weights('ckpt')
model.save('models',save_format='tf')
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):
        super().__init__(*args,**kwargs)
        self.encoder,self.decoder=self.build_inference_model(base_encoder,base_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=[encoder_state_h,encoder_state_c],nam
e='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,initial_state=[decoder_input_state_h,decod
er_input_state_c])
        decoder_outputs=base_decoder.layers[-1](x)
```

```
    decoder=tf.keras.models.Model(
        inputs=[decoder_inputs,[decoder_input_state_h,decoder_input_state_c]],
        outputs=[decoder_outputs,[decoder_state_h,decoder_state_c]],name='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],training=False)
#            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='chatbot')
chatbot.summary()
tf.keras.utils.plot_model(chatbot.encoder,to_file='encoder.png',show_shapes=True,show_layer_activation
s=True)
tf.keras.utils.plot_model(chatbot.decoder,to_file='decoder.png',show_shapes=True,show_layer_activation
s=True)
def print_conversation(texts):
    for text in texts:
        print(f'You: {text}')
        print(f'Bot: {chatbot(text)}')
        print('========================')
```

```
print_conversation([
    'hi',
    'do yo know me?',
    'what is your name?',
    'you are bot?',
    'hi, how are you doing?',
    "i'm pretty good. thanks for asking.",
    "Don't ever be in a hurry",
    '''I'm gonna put some dirt in your eye ''',
    '''You're trash ''',
    '''I've read all your research on nano-technology ''',
    '''You want forgiveness? Get religion''',
    '''While you're using the bathroom, i'll order some food.''',
    '''Wow! that's terrible.''',
    '''We'll be here forever.''',
    '''I need something that's reliable.''',
    '''A speeding car ran a red light, killing the girl.''',
    '''Tomorrow we'll have rice and fish for lunch.''',
    '''I like this restaurant because they give you free bread.'''])
```