IBM PROJECT AI101

ARTIFICIAL INTELLIGENCE-GROUP 3

PROJECT – 3. CHAT BOT

Creating a chatbot for exceptional customer service is a complex task, but I can provide you with a high-level outline of the steps involved in designing and developing such a chatbot in Python.

1. **Functionality: Define the Scope**
    - Determine the specific tasks and questions your chatbot will handle. For example, it could answer frequently asked questions, provide product recommendations, assist with troubleshooting, or handle booking requests.
    - Create a list of intents or user queries that the chatbot should be able to recognize and respond to.

2. **User Interface: Design the Interface**
    - Decide where you want to integrate the chatbot (website, mobile app, or both).
    - Design a user-friendly and intuitive interface for users to interact with the chatbot. This could be a chatbox, a voice-based interface, or a combination of both.
    - Ensure that the interface is responsive and works well on various devices and screen sizes.

3. **Natural Language Processing (NLP): Implement NLP Techniques**
    - Choose an NLP library or framework for Python, such as spaCy, NLTK, or Hugging Face Transformers.
    - Preprocess user input to clean and tokenize the text.
    - Train or fine-tune an NLP model on your dataset to understand user intent and extract relevant information.
    - Implement techniques for entity recognition to identify important details in user queries.

4. **Responses: Plan Bot Responses**
    - Create a database of responses or templates that the chatbot can use to generate answers.
    - Implement a response generation system that selects the most appropriate response based on the user's query and the context.
    - Ensure that responses are concise, accurate, and user-friendly.

5. **Integration: Integrate with Website/App**
    - Depending on your platform, integrate the chatbot using appropriate libraries or frameworks. For example, you can use JavaScript for web integration or mobile app development tools for mobile apps.
    - Set up communication between the chatbot and your backend server if necessary.
    - Implement a user authentication system if personalization is required.

6. **Testing and Improvement: Continuous Enhancement**
    - Conduct thorough testing of the chatbot to identify and fix any issues or limitations.
    - Gather user feedback and monitor user interactions to identify areas for improvement.
    - Continuously update the chatbot's responses and functionality based on user feedback and changing business requirements.

7. **Dataset and Training**
    - Use the provided dataset as a starting point for training your chatbot.
    - Preprocess the dataset, clean the text, and structure it into intents and responses.
    - Train or fine-tune your NLP model using this dataset to improve the chatbot's understanding of user queries.

8. **Deployment and Maintenance**
    - Deploy your chatbot to a production environment and monitor its performance.
    - Implement logging and analytics to track user interactions and diagnose issues.
    - Regularly update and maintain the chatbot to keep it up-to-date with new information, products, or services.

Remember that building an exceptional customer service chatbot is an iterative process. You will likely need to refine and expand its capabilities over time based on user feedback and evolving business needs. Additionally, you may consider integrating machine learning techniques to improve the chatbot's performance and personalization.

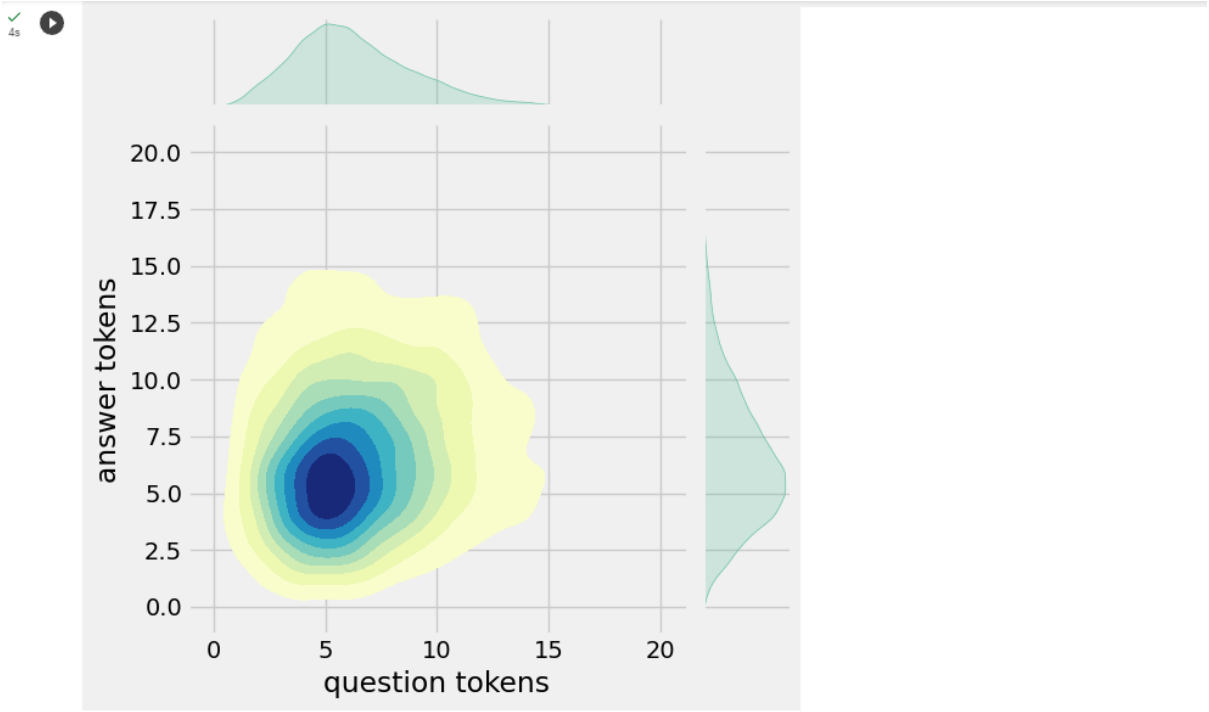
PROJECT CODE:

```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.layers import TextVectorization
import re,string
from tensorflow.keras.layers import LSTM,Dense,Embedding,Dropout,LayerNormalization
```

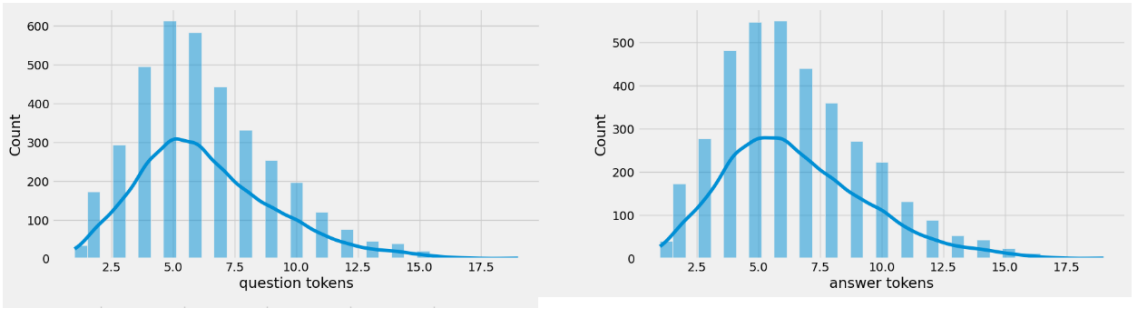
```
df=pd.read_csv('dialogs.txt',sep='\t',names=['question','answer'])
print(f'Dataframe size: {len(df)}')
df.head()
```

```
Dataframe size: 3725
```

	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.



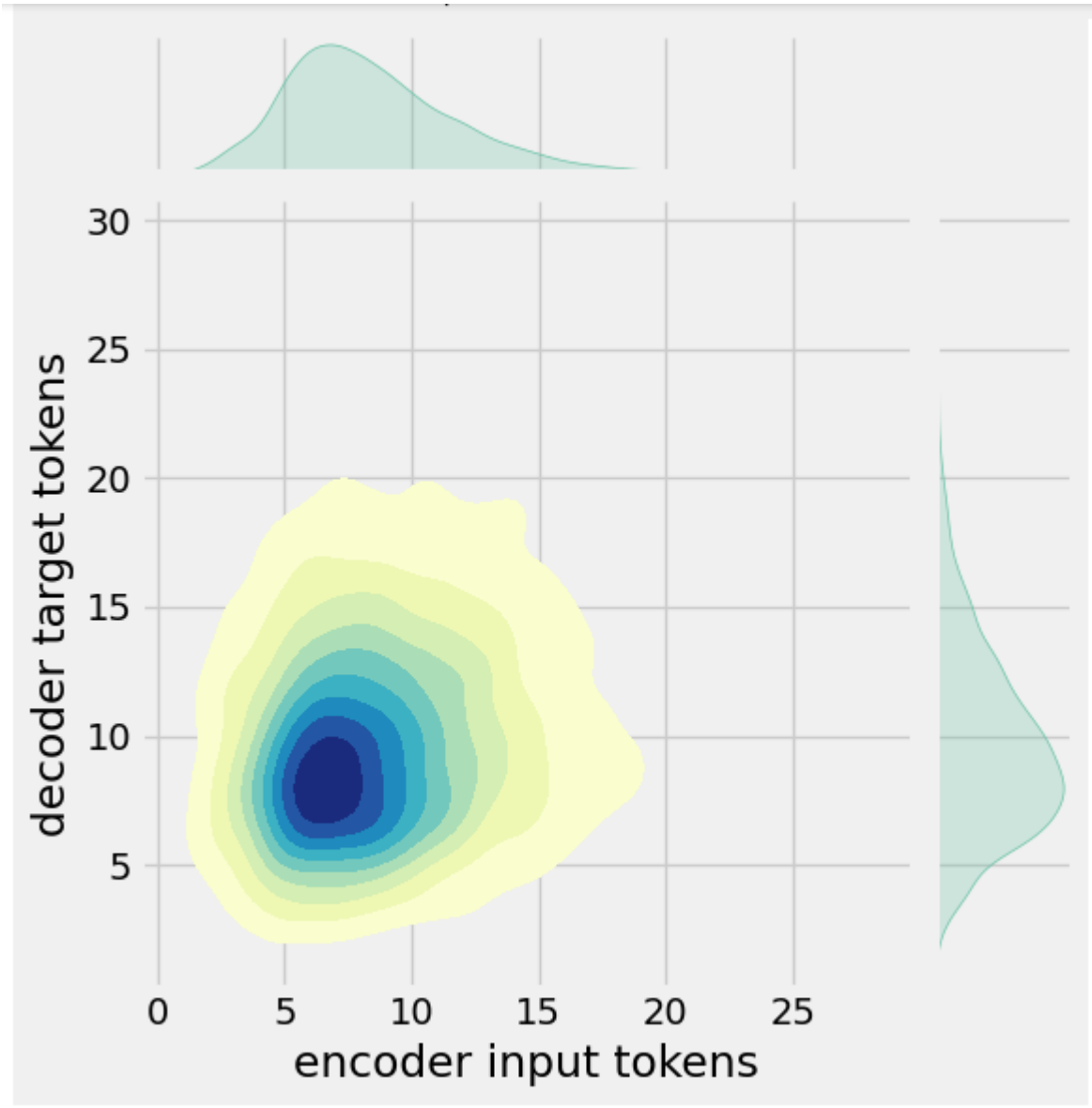
```
df['question tokens']=df['question'].apply(lambda x:len(x.split()))
df['answer tokens']=df['answer'].apply(lambda x:len(x.split()))
plt.style.use('fivethirtyeight')
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))
sns.set_palette('Set2')
sns.histplot(x=df['question tokens'],data=df,kde=True,ax=ax[0])
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()
```



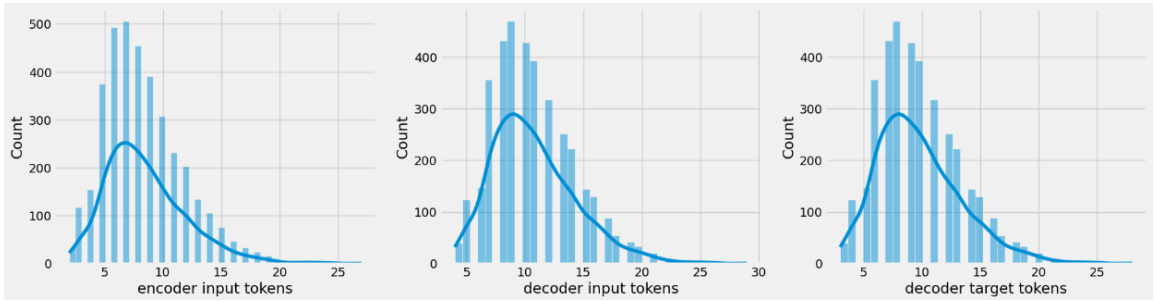

```
df.head(10)
```

	question	answer	encoder_inputs	decoder_targets	decoder_inputs
0	hi, how are you doing?	i'm fine. how about yourself?	hi , how are you doing ?	i ' m fine . how about yourself ? <end>	<start> i ' m fine . how about yourself ? <end>
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.	i ' m fine . how about yourself ?	i ' m pretty good . thanks for asking . <end>	<start> i ' m pretty good . thanks for asking...
2	i'm pretty good. thanks for asking.	no problem. so how have you been?	i ' m pretty good . thanks for asking .	no problem . so how have you been ? <end>	<start> no problem . so how have you been ? ...
3	no problem. so how have you been?	i've been great. what about you?	no problem . so how have you been ?	i ' ve been great . what about you ? <end>	<start> 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.	i ' ve been great . what about you ?	i ' ve been good . i ' m in school right now ...	<start> i ' ve been good . i ' m in school ri...
5	i've been good. i'm in school right now.	what school do you go to?	i ' ve been good . i ' m in school right now .	what school do you go to ? <end>	<start> what school do you go to ? <end>
6	what school do you go to?	i go to pcc.	what school do you go to ?	i go to pcc . <end>	<start> i go to pcc . <end>
7	i go to pcc.	do you like it there?	i go to pcc .	do you like it there ? <end>	<start> do you like it there ? <end>
8	do you like it there?	it's okay. it's a really big campus.	do you like it there ?	it ' s okay . it ' s a really big campus . <...	<start> it ' s okay . it ' s a really big cam...
9	it's okay. it's a really big campus.	good luck with school.	it ' s okay . it ' s a really big campus .	good luck with school . <end>	<start> good luck with school . <end>

```
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['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.tol
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']
buffer_size=params['buffer_size']
max_sequence_length=params['max_sequence_length']
df.head(10)
```

```
After preprocessing: for example ,  if your birth date is january  1  2  ,  1  9  8  7  ,  write  0  1  /  1  2  /  8  7  .
Max encoder input length: 27
Max decoder input length: 29
Max decoder target length: 28
```

	encoder_inputs	decoder_targets	decoder_inputs
0	hi , how are you doing ?	i ' m fine . how about yourself ? <end>	<start> i ' m fine . how about yourself ? <end>
1	i ' m fine . how about yourself ?	i ' m pretty good . thanks for asking . <end>	<start> i ' m pretty good . thanks for asking...
2	i ' m pretty good . thanks for asking .	no problem . so how have you been ? <end>	<start> no problem . so how have you been ? ...
3	no problem . so how have you been ?	i ' ve been great . what about you ? <end>	<start> 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 ...	<start> i ' ve been good . i ' m in school ri...
5	i ' ve been good . i ' m in school right now .	what school do you go to ? <end>	<start> what school do you go to ? <end>
6	what school do you go to ?	i go to pcc . <end>	<start> i go to pcc . <end>
7	i go to pcc .	do you like it there ? <end>	<start> do you like it there ? <end>
8	do you like it there ?	it ' s okay . it ' s a really big campus . <...	<start> it ' s okay . it ' s a really big cam...
9	it ' s okay . it ' s a really big campus .	good luck with school . <end>	<start> good luck with school . <end>

```
print(f'Decoder input shape: {yd.shape}')
print(f'Decoder target shape: {y.shape}')
```

```
Question sentence: hi , how are you ?
Question to tokens: [1971    9   45   24    8    7    0    0    0    0]
Encoder input shape: (3725, 30)
Decoder input shape: (3725, 30)
Decoder target shape: (3725, 30)
```

```
[15] 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]} ...')
```

```
Encoder input: [1971    9   45   24    8  194    7    0    0    0    0    0] ...
Decoder input: [   4    6    5   38  646    3   45   41  563    7    2    0] ...
Decoder target: [   6    5   38  646    3   45   41  563    7    2    0    0] ...
```

```
vectorize_layer=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]}')
```

```
Vocab size: 2443
['', '[UNK]', '<end>', '.', '<start>', "'", 'i', '?', 'you', ',', 'the', 'to']
```

```
[14] 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}')
```

```
data=tf.data.Dataset.from_tensor_slices((x,yd,y))
data=data.shuffle(buffer_size)

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}')
```

```
Number of train batches: 23
Number of training data: 3427
Number of validation batches: 3
Number of validation data: 447
Encoder Input shape (with batches): (149, 30)
Decoder Input shape (with batches): (149, 30)
Target Output shape (with batches): (149, 30)
```

```
        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])
```

```
(<tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[-0.17145002, -0.04260545,  0.2256314 , ..., -0.0145164 ,
         -0.17359242, -0.01855551],
        [-0.10152829,  0.13051443, -0.01529688, ..., -0.07839621,
         -0.11302923, -0.09111515],
        [-0.13723563,  0.06314871, -0.35969943, ..., -0.07937612,
          0.03776905,  0.18839118],
        ...,
        [-0.00948302,  0.01294993,  0.07565455, ..., -0.0822028 ,
         -0.07200203, -0.05146787],
        [-0.07005285,  0.02555089,  0.00188984, ...,  0.02081227,
         -0.22570358, -0.03289159],
        [-0.27591974,  0.09821565, -0.15053254, ...,  0.13982196,
         -0.07864141,  0.00545281]], dtype=float32)>,
 <tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[-0.52501184, -0.09229817,  0.46547413, ..., -0.03284347,
         -0.4565962 , -0.04035625],
        [-0.30694658,  0.32048392, -0.02922243, ..., -0.18096462,
         -0.30194065, -0.19842044],
        [-0.21641225,  0.18601088, -0.68878156, ..., -0.18863262,
          0.09230713,  0.31972426],
        ...,
        [-0.02837378,  0.02808335,  0.15522261, ..., -0.18543348,
         -0.18222567, -0.10704239],
        [-0.21262512,  0.05439565,  0.00377202, ...,  0.0460677 ,
         -0.65940535, -0.07160754],
        [-0.4611279 ,  0.31099635, -0.280626  , ...,  0.3438197 ,
         -0.19100055,  0.00925176]], dtype=float32)>)
```

Builder Encoder

```
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)
```

```
        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]))
```

```
<tf.Tensor: shape=(1, 30, 2443), dtype=float32, numpy=
array([[[2.85351271e-04, 2.16305649e-04, 6.01786014e-05, ...,
         4.10375971e-04, 3.27060916e-05, 2.30062884e-04],
        [3.85508145e-04, 7.79589682e-05, 1.02411585e-04, ...,
         2.87848059e-04, 3.14770114e-05, 1.94452950e-05],
        [5.27507982e-05, 4.91588580e-05, 3.35450168e-04, ...,
         1.82789561e-04, 1.70131272e-04, 2.90362605e-05],
        ...,
        [3.05304420e-05, 3.24474444e-04, 1.06263353e-04, ...,
         6.16623947e-05, 4.36500908e-04, 5.61305460e-05],
        [3.05304420e-05, 3.24474444e-04, 1.06263353e-04, ...,
         6.16623947e-05, 4.36500908e-04, 5.61305460e-05],
        [3.05304420e-05, 3.24474444e-04, 1.06263353e-04, ...,
         6.16623947e-05, 4.36500908e-04, 5.61305460e-05]]], dtype=float32)>
```

```
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)
```

```
    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
```

Build Training Model

```
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)
```

```
          [[4.84186137e-04, 5.55964420e-04, 6.26785550e-05, ...,
            1.24011049e-03, 3.20533836e-05, 2.00664828e-04],
           [1.80135656e-04, 2.12483268e-04, 2.18024550e-04, ...,
            2.93179788e-03, 1.59664254e-04, 7.73084612e-05],
           [1.38315227e-04, 5.59945984e-05, 2.96784128e-04, ...,
            7.95073109e-04, 3.48751666e-04, 1.44463062e-04],
           ...,
           [2.90083699e-05, 1.66829777e-04, 4.75352645e-05, ...,
            1.42469173e-04, 4.96200868e-04, 1.87878890e-04],
           [2.90083699e-05, 1.66829777e-04, 4.75352645e-05, ...,
            1.42469173e-04, 4.96200868e-04, 1.87878890e-04],
           [2.90083699e-05, 1.66829777e-04, 4.75352645e-05, ...,
            1.42469173e-04, 4.96200868e-04, 1.87878890e-04]],

          [[6.11300929e-04, 1.33299339e-03, 1.48072024e-04, ...,
            1.62014307e-03, 1.24733924e-04, 7.29546009e-05],
           [3.77469732e-05, 1.01345417e-04, 3.52506795e-05, ...,
            2.13585285e-04, 6.59531070e-05, 5.86589566e-04],
           [2.40089230e-05, 5.54721082e-05, 3.97601943e-05, ...,
            1.60320415e-04, 5.47426404e-04, 1.03108716e-04],
           ...,
           [2.57122792e-05, 8.62471643e-04, 2.60748930e-04, ...,
            8.11950886e-05, 2.73739366e-04, 4.75429370e-05],
           [2.57122792e-05, 8.62471643e-04, 2.60748930e-04, ...,
            8.11950886e-05, 2.73739366e-04, 4.75429370e-05],
           [2.57122792e-05, 8.62471643e-04, 2.60748930e-04, ...,
            8.11950886e-05, 2.73739366e-04, 4.75429370e-05]]], dtype=float32)>
```

```
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])
```

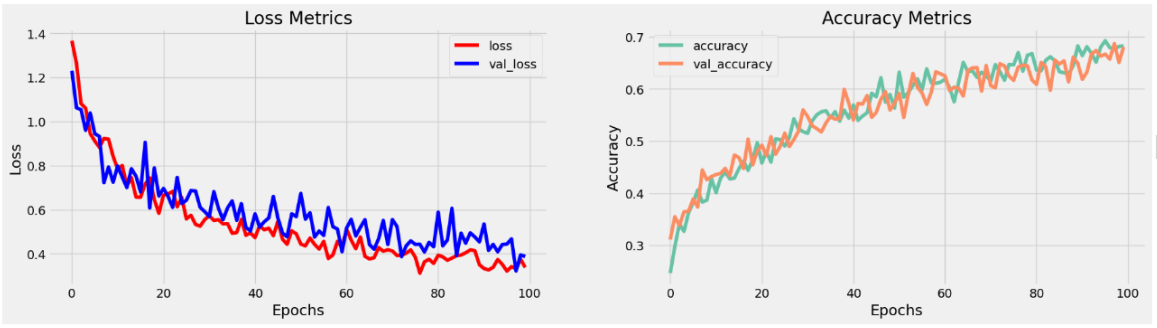
```
<tf.Tensor: shape=(149, 30, 2443), dtype=float32, numpy=
array([[[2.85351125e-04, 2.16305736e-04, 6.01786305e-05, ...,
         4.10375971e-04, 3.27060916e-05, 2.30063102e-04],
        [3.85507854e-04, 7.79589900e-05, 1.02411555e-04, ...,
         2.87848117e-04, 3.14770041e-05, 1.94452932e-05],
        [5.27508091e-05, 4.91588435e-05, 3.35450546e-04, ...,
         1.82789721e-04, 1.70131461e-04, 2.90362659e-05],
        ...,
        [3.05304202e-05, 3.24474502e-04, 1.06263324e-04, ...,
         6.16623802e-05, 4.36500733e-04, 5.61306078e-05],
        [3.05304202e-05, 3.24474502e-04, 1.06263324e-04, ...,
         6.16623802e-05, 4.36500733e-04, 5.61306078e-05],
        [3.05304202e-05, 3.24474502e-04, 1.06263324e-04, ...,
         6.16623802e-05, 4.36500733e-04, 5.61306078e-05]],

       [[2.06110228e-04, 7.12070556e-04, 9.69701141e-05, ...,
         5.72449004e-04, 6.11645810e-05, 4.02734353e-04],
        [6.22326392e-04, 1.78978400e-04, 2.81426241e-03, ...,
         2.68322590e-04, 2.70739420e-05, 2.92494544e-04],
        [1.31035427e-04, 6.53479830e-04, 1.29260798e-03, ...,
         5.88871539e-04, 1.56488532e-04, 1.56494061e-04],
        ...,
        [4.90486491e-05, 8.17499764e-04, 1.39667580e-04, ...,
         3.97730437e-05, 5.09029487e-04, 8.68067582e-05],
        [4.90486491e-05, 8.17499764e-04, 1.39667580e-04, ...,
         3.97730437e-05, 5.09029487e-04, 8.68067582e-05],
        [4.90486491e-05, 8.17499764e-04, 1.39667580e-04, ...,
```

Train Model

```
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)
    ]
)
```

```
23/23 [==============================] - 38s 2s/step - loss: 0.3605 - accuracy: 0.6601 - val_loss: 0.4476 - val_accuracy: 0.6538
Epoch 88/100
23/23 [==============================] - ETA: 0s - loss: 0.3585 - accuracy: 0.6644
Epoch 88: val_loss did not improve from 0.38672
23/23 [==============================] - 38s 2s/step - loss: 0.3609 - accuracy: 0.6629 - val_loss: 0.4946 - val_accuracy: 0.6134
Epoch 89/100
23/23 [==============================] - ETA: 0s - loss: 0.3559 - accuracy: 0.6634
Epoch 89: val_loss did not improve from 0.38672
23/23 [==============================] - 38s 2s/step - loss: 0.3583 - accuracy: 0.6626 - val_loss: 0.4761 - val_accuracy: 0.6609
Epoch 90/100
23/23 [==============================] - ETA: 0s - loss: 0.3540 - accuracy: 0.6649
Epoch 90: val_loss did not improve from 0.38672
23/23 [==============================] - 39s 2s/step - loss: 0.3539 - accuracy: 0.6656 - val_loss: 0.4536 - val_accuracy: 0.6490
Epoch 91/100
23/23 [==============================] - ETA: 0s - loss: 0.3512 - accuracy: 0.6679
Epoch 91: val_loss did not improve from 0.38672
23/23 [==============================] - 38s 2s/step - loss: 0.3504 - accuracy: 0.6677 - val_loss: 0.5347 - val_accuracy: 0.6182
Epoch 92/100
23/23 [==============================] - ETA: 0s - loss: 0.3526 - accuracy: 0.6681
Epoch 92: val_loss did not improve from 0.38672
23/23 [==============================] - 37s 2s/step - loss: 0.3515 - accuracy: 0.6686 - val_loss: 0.4149 - val_accuracy: 0.6321
Epoch 93/100
```

```
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()
```



Save Model

```
model.load_weights('ckpt')
model.save('models',save_format='tf')
```

```
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_config()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_config()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_config()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
WARNING:tensorflow:Model's `__init__()` arguments contain non-serializable objects. Please implement a `get_config()` method in the subclassed Model for proper saving and loading. Defaulting to empty config.
```

```
[25] for idx,i in enumerate(model.layers):
        print('Encoder layers:' if idx==0 else 'Decoder layers: ')
        for j in i.layers:
            print(j)
        print('---------------------')
```

```
Encoder layers:
<keras.src.layers.core.embedding.Embedding object at 0x792cb59dece0>
<keras.src.layers.normalization.layer_normalization.LayerNormalization object at 0x792cb59df880>
<keras.src.layers.rnn.lstm.LSTM object at 0x792cb376eb80>
---------------------
Decoder layers:
<keras.src.layers.core.embedding.Embedding object at 0x792cb35e3910>
<keras.src.layers.normalization.layer_normalization.LayerNormalization object at 0x792cb35e1900>
<keras.src.layers.rnn.lstm.LSTM object at 0x792cb367d810>
<keras.src.layers.core.dense.Dense object at 0x792cb35ac460>
---------------------
```

```
    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==
```

Create Inference Model

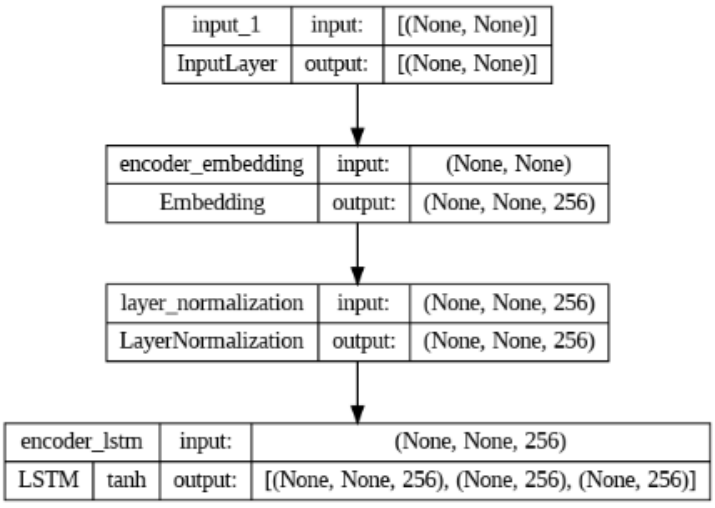
```
class ChatBot(tf.keras.models.Model):
    def __init__(self,base_encoder,base_decoder,*args,**kwargs):
        super().__init__(*args,**kwargs)
        self.encoder,self.decoder=self.build_inference_model(base_encoder,base_decoder)

    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],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,initial_state=[decoder_input_state_h,decoder_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)
```

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

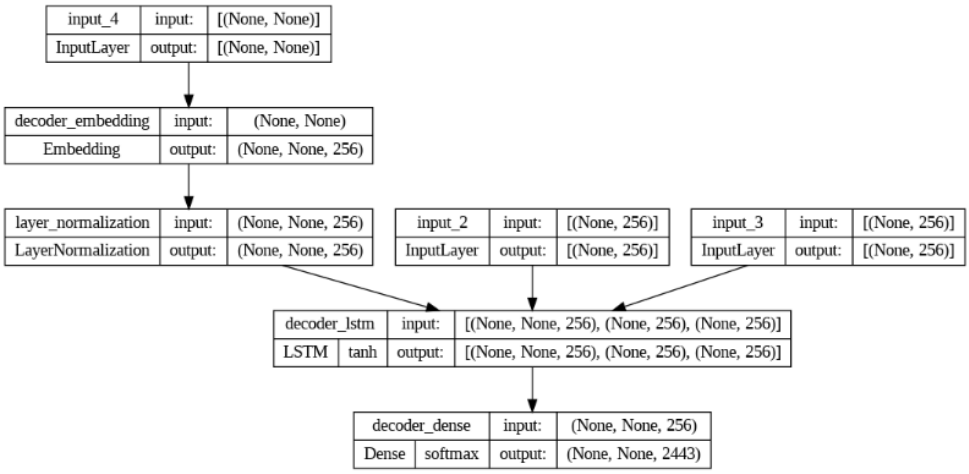
```
Model: "chatbot_encoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 encoder_embedding (Embeddi  (None, None, 256)         625408
 ng)

 layer_normalization (Layer  (None, None, 256)         512
 Normalization)

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

=================================================================
Total params: 1151232 (4.39 MB)
Trainable params: 1151232 (4.39 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Model: "chatbot_decoder"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_4 (InputLayer)        [(None, None)]            0         []

 decoder_embedding (Embeddi  (None, None, 256)         625408    ['input_4[0][0]']
 ng)

 layer_normalization (Layer  (None, None, 256)         512       ['decoder_embedding[0][0]']
 Normalization)

 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    ['decoder_lstm[0][0]']

==================================================================================================
Total params: 1779083 (6.79 MB)
```



```
tf.keras.utils.plot_model(chatbot.encoder,to_file='encoder.png',show_shapes=True,show_layer_activations=True)
```



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




```
[32] 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.'''
])
```

```
You: hi
Bot: what's the matter?
=======================
You: do yo know me?
Bot: the forecast says that it will be warm the weekend.
=======================
You: what is your name?
Bot: it's about $10.
=======================
You: you are bot?
Bot: i'm not sure.
=======================
You: hi, how are you doing?
Bot: i'm going to be a teacher.
=======================
You: i'm pretty good. thanks for asking.
Bot: no problem. that's a good idea.
=======================
You: Don't ever be in a hurry
Bot: it's the best job.
=======================
You: I'm gonna put some dirt in your eye
Bot: that's a good idea.
=======================
You: You're trash
Bot: no. it's not too.
```