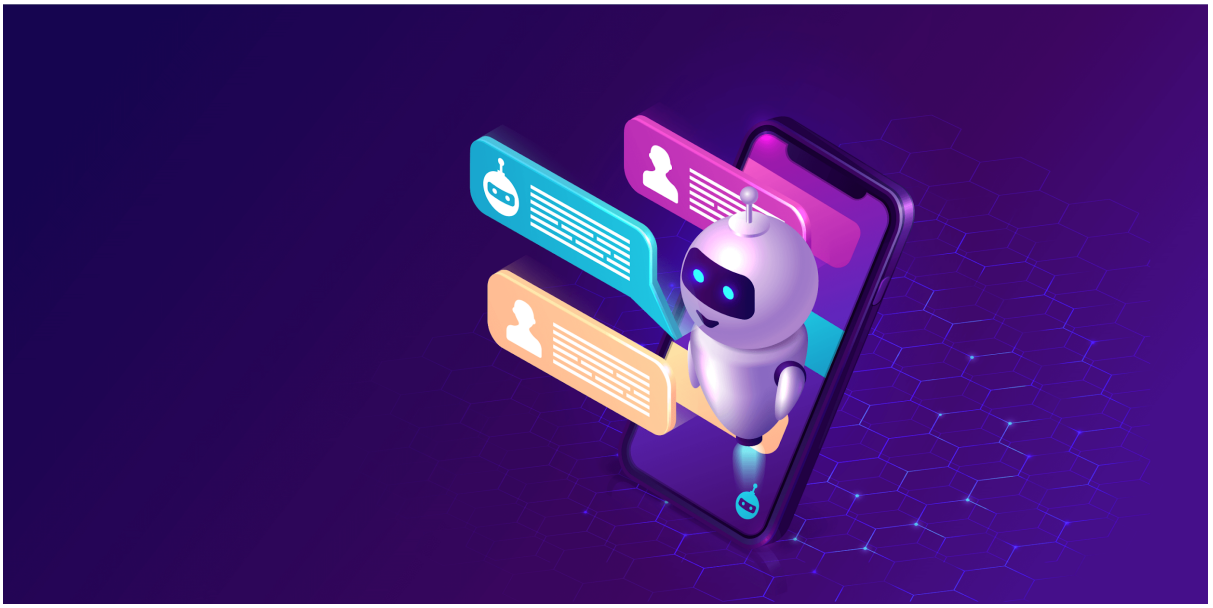
ARTIFICIAL INTELLIGENCE

CREATE A CHATBOT USING PYTHON



Name: Kalavathy K

Reg. No: 513521104019

Department: CSE

Year: III

NM ID: au513521104019

E-Mail: kaviyakalavathy@gmail.com

Phase 3 (Development Part-1)

INTRODUCTION:

"Welcome to the world of Chatbot development with Python! In this project, we embark on a journey to create an intelligent and interactive chatbot powered by Python's Natural Language Processing (NLP) capabilities. Our goal is to design a chatbot that can engage in meaningful conversations, answer questions, and provide assistance across a range of domains. Through data acquisition, model training, and iterative improvement, we aim to craft a chatbot that not only understands user queries but also offers contextually relevant responses. Join us as we explore the fascinating realm of conversational AI and create a chatbot that's ready to assist and engage with users."

In the continuation of the previous documentation AI_Phase2, we will be looking forward on the development part of the project. In this phase, we will start building the model by loading the dataset and processing the data

DATASET LINK: https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot

The following steps are followed in this development part.

1. EMPATHIZE

Understanding user needs is a critical aspect of designing and developing a chatbot that is effective and user-friendly. Developers should consider the following key user needs:

Clear Communication: Users need a chatbot that communicates clearly and in a language they understand. The chatbot should avoid jargon and complex terminology, providing responses that are easy to follow.

Efficiency: Users value efficiency in their interactions with a chatbot. They expect quick and accurate responses to their queries or requests. Developers should prioritize minimizing response times and streamlining interactions.

Relevance: Users want the chatbot to provide information or assistance that is relevant to their needs or context. Developers should ensure that the chatbot's responses and actions are tailored to individual users when possible.

Problem Solving: Users often turn to chatbots to solve problems or complete tasks. Developers should design the chatbot to be capable of addressing common user issues and helping users accomplish their goals.

Personalization: Personalization is increasingly important to users. Developers can enhance user experience by allowing chatbots to remember user preferences, past interactions, and other relevant information to deliver a more personalized experience.

User-Friendly Interface: The chatbot's user interface should be intuitive and user-friendly. Users need to be able to navigate the chatbot easily and understand how to use it without confusion.

Privacy and Security: Users are concerned about their data and privacy. Developers must implement robust security measures to protect user information and communicate transparently about data handling practices.

Availability: Users expect chatbots to be available when needed. Ensure the chatbot is accessible 24/7 if possible, or communicate its operating hours clearly to users.

Error Handling: Users appreciate a chatbot that can gracefully handle errors and unexpected inputs. Developers should design the chatbot to respond appropriately when it encounters user mistakes or misunderstandings.

Feedback and Help: Users should have the ability to provide feedback on the chatbot's performance or request assistance when needed. Developers can implement mechanisms for users to report issues or seek human assistance when the chatbot cannot provide a solution.

Accessibility, Consistency, MultilingualSupport, Integration, Simplicity, Empathy, Quality Content etc are some of the user needs in development of chatbot.

By comprehensively understanding these user needs, developers can create chatbots that not only meet user expectations but also provide valuable and satisfying interactions. Regular user feedback and testing are essential to continuously refine the chatbot based on evolving user needs.

2.DEFINE:

"The problem to be solved is that users need an efficient and user-friendly way to schedule appointments with healthcare providers. Existing scheduling systems are often cumbersome, require phone calls, and are prone to errors, leading to patient frustration and missed appointments. The chatbot's objective

is to streamline the appointment scheduling process, provide real-time availability information, and send timely reminders to both patients and healthcare providers, ultimately improving the patient experience and reducing appointment no-shows."This problem statement is clear and specific. It identifies the users' pain points (inefficient scheduling, missed appointments), the limitations of the current system, and the chatbot's role in addressing these issues (streamlining scheduling, real-time information, reminders). It also aligns with the user needs and objectives, making it a robust foundation for the chatbot's design and development.

3.Importing Libraries:

In this step we will import all the required libraries and packages from the pre-installed modules.

```
In [4]: 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
        from flask import Flask
```

4.Loading datasets:

In this step we will be loading the chatbots datasets acquired from Kaggle.

```
In [10]: df=pd.read_csv('dialogs.txt',sep='\t',names=['question','answer'])
         print(f'Dataframe size: {len(df)}')
         df.head()

         Dataframe size: 3725
```

Out[10]:

	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.

---

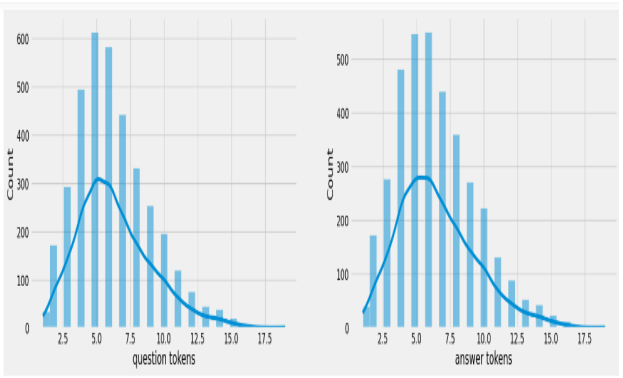
5. Ideate:

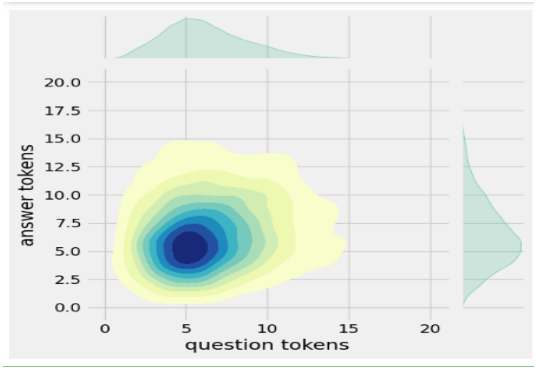
Data Preprocessing: (Data Visualization)

We will be pre-processing the data for the further development of model.

```
In [11]: 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()
```

O/P:





6.Text Cleaning:

Text cleaning, also known as text preprocessing, is the process of preparing and standardizing textual data to make it suitable for analysis or natural language processing (NLP) tasks. This process involves several steps to remove or transform elements in text data that can hinder accurate analysis or modeling.

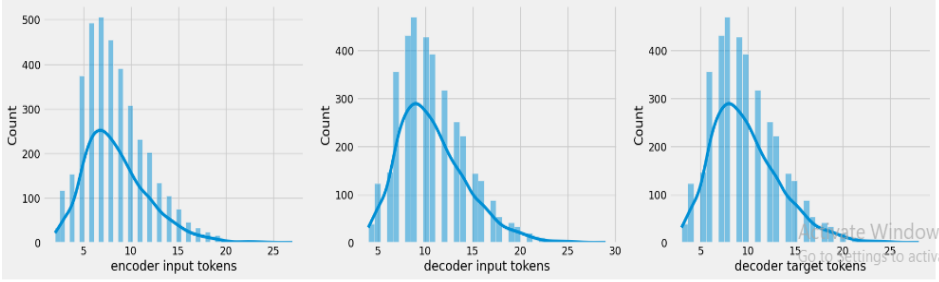
```
def clean_text(text):
    text=re.sub('-',' ',text.lower())
    text=re.sub('[.]',' . ',text)
    text=re.sub('[1]',' 1 ',text)
    text=re.sub('[2]',' 2 ',text)
    text=re.sub('[3]',' 3 ',text)
    text=re.sub('[4]',' 4 ',text)
    text=re.sub('[5]',' 5 ',text)
    text=re.sub('[6]',' 6 ',text)
    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)
```

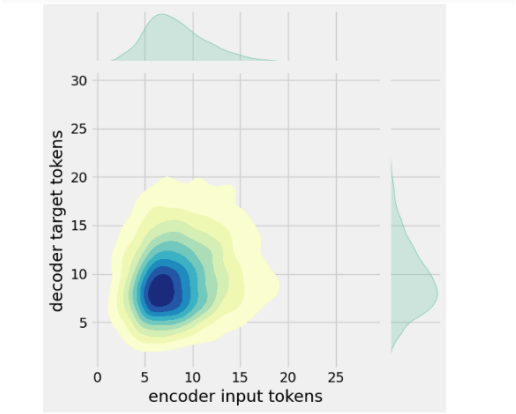
```
df.drop(columns=['answer tokens','question tokens'],axis=1,inplace=True)
df['encoder_inputs']=df['question'].apply(clean_text)
df['decoder_targets']=df['answer'].apply(clean_text)+' <end>'
df['decoder_inputs']='<start> '+df['answer'].apply(clean_text)+' <end>'

df.head(10)
```

Out[12]:

	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>

In [13]:
```
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()
```





After text preprocessing in the context of chatbot development, the text data is in a cleaner and more structured format, ready for further analysis, understanding, and use in natural language processing (NLP) tasks.

```
print(f"After preprocessing: {' '.join(df[df['encoder input tokens'].max()==df['encoder input tokens']]['encoder_inputs'].values.
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>

7. Tokenization:

Tokenization is the process of splitting a text or a sequence of characters into individual units, known as tokens. These tokens are typically words,

phrases, or other meaningful elements. Tokenization is a fundamental step in natural language processing (NLP) and is crucial for various text analysis tasks.

In [15]:
```
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']
```

In [16]:
```
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}')
```

```
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)
```

In [17]:
```
print(f'Encoder input: {x[0][:12]} ...')
print(f'Decoder input: {yd[0][:12]} ...')
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] ...
```

In [18]:
```
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}')
```

O\P:

```
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)
```

8. Prototype:

Build encoder,

To build an encoder, you'll typically be working in the context of deep learning, and this encoder is often associated with autoencoders, recurrent neural networks (RNNs), or other neural network architectures.

In [19]:
```
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])
```

Out[19]:
```
(<tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[ 0.19900371, -0.1929147 ,  0.07616628, ...,  0.06260844,
         -0.05182864, -0.06066753],
        [ 0.26634306,  0.03015221, -0.10834284, ...,  0.12511149,
          0.09928918,  0.09458925],
        [ 0.04378179,  0.34261167, -0.23351805, ...,  0.10806578,
         -0.03977764, -0.2159342 ],
        ...,
        [ 0.15017916, -0.35100752,  0.12575859, ...,  0.18343027,
          0.04804143, -0.0435859 ],
        [ 0.11577409, -0.06436783,  0.0907531 , ...,  0.08779578,
          0.09389018,  0.06551306],
        [ 0.22780733,  0.10320853, -0.08747951, ...,  0.12326789,
          0.05119178, -0.00499345]], dtype=float32)>,
 <tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[ 0.3750309 , -0.2936198 ,  0.19917853, ...,  0.11117338,
         -0.169982  , -0.12423997],
        [ 0.40158898,  0.0662723 , -0.2980452 , ...,  0.2906554 ,
          0.25374115,  0.2768807 ],
        [ 0.08081909,  0.5704955 , -0.6084406 , ...,  0.1829527 ,
         -0.13145694, -0.47369635],
        ...,
        [ 0.29247916, -0.5301531 ,  0.3527859 , ...,  0.328242  ,
          0.15890412, -0.09224524],
        [ 0.20818451, -0.12069319,  0.16402757, ...,  0.18189447,
          0.15012874,  0.10780704],
        [ 0.42850024,  0.14844967, -0.25023955, ...,  0.21185929,
          0.16683699, -0.01101565]], dtype=float32)>)
```

Build Decoder,

In [20]:
```
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]))
```

Out[20]:
```
<tf.Tensor: shape=(1, 30, 2443), dtype=float32, numpy=
array([[[9.3978917e-04, 5.8102928e-04, 3.4918604e-04, ...,
         9.6560674e-05, 3.4844521e-03, 1.9126637e-03],
        [1.5957150e-04, 7.5159995e-05, 4.0091670e-04, ...,
         9.1877293e-05, 2.4869712e-04, 2.4281540e-03],
        [1.7921842e-04, 1.2850739e-03, 6.6422741e-04, ...,
         8.5191889e-05, 6.6548015e-04, 2.1413603e-04],
        ...,
        [1.0080441e-04, 2.4488752e-03, 2.6977875e-03, ...,
         8.5539563e-05, 6.7219567e-03, 2.0856218e-04],
        [1.0080446e-04, 2.4488750e-03, 2.6977872e-03, ...,
         8.5539556e-05, 6.7219539e-03, 2.0856215e-04],
        [1.0080446e-04, 2.4488750e-03, 2.6977872e-03, ...,
         8.5539556e-05, 6.7219539e-03, 2.0856215e-04]]], dtype=float32)>
```


Build Training Model,

In [21]:
```
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])
```


Out[21]:
```
<tf.Tensor: shape=(149, 30, 2443), dtype=float32, numpy=
array([[[9.39789286e-04, 5.81029046e-04, 3.49186215e-04, ...,
         9.65607323e-05, 3.48445144e-03, 1.91266334e-03],
        [1.59571457e-04, 7.51599000e-05, 4.00916470e-04, ...,
         9.18773512e-05, 2.48697062e-04, 2.42815516e-03],
        [1.79218347e-04, 1.28507416e-03, 6.64227584e-04, ...,
         8.51919322e-05, 6.65480271e-04, 2.14136118e-04],
        ...,
        [1.00804398e-04, 2.44887429e-03, 2.69778934e-03, ...,
         8.55395920e-05, 6.72195712e-03, 2.08562240e-04],
        [1.00804398e-04, 2.44887429e-03, 2.69778934e-03, ...,
         8.55395920e-05, 6.72195712e-03, 2.08562240e-04],
        [1.00804398e-04, 2.44887429e-03, 2.69778934e-03, ...,
         8.55395920e-05, 6.72195712e-03, 2.08562240e-04]],

       [[1.48899248e-03, 1.11791574e-04, 1.69336592e-04, ...,
         2.72725611e-05, 1.22156390e-03, 1.01141294e-03],
        [7.83129595e-04, 2.81261309e-04, 5.02188013e-05, ...,
         1.78441359e-03, 9.25112690e-05, 1.86895180e-04],
        [1.20824232e-04, 5.75737329e-04, 4.34654357e-05, ...,
         1.15328957e-03, 6.45271575e-05, 5.51116944e-04],
        ...,
        [3.07689370e-05, 6.83894672e-04, 1.33376860e-03, ...,
         1.96581837e-04, 1.09349994e-03, 6.80164376e-05],
        [3.07689370e-05, 6.83894672e-04, 1.33376860e-03, ...,
         1.96581837e-04, 1.09349994e-03, 6.80164376e-05],
        [3.07689370e-05, 6.83894672e-04, 1.33376860e-03, ...,
         1.96581837e-04, 1.09349994e-03, 6.80164376e-05]],


         8.04460724e-05, 2.84239650e-03, 6.34894473e-03],
        [1.06537039e-03, 1.79962313e-04, 7.52430497e-05, ...,
         3.37697216e-04, 8.45795556e-04, 2.65551789e-04],
        [1.22962883e-04, 6.93331094e-05, 6.00464518e-05, ...,
         3.87452892e-05, 9.59527737e-04, 1.02134619e-03],
        ...,
        [1.17018724e-04, 4.17125382e-04, 2.49348185e-03, ...,
         8.36507315e-05, 8.44088104e-03, 9.44107742e-05],
        [1.17018724e-04, 4.17125382e-04, 2.49348185e-03, ...,
         8.36507315e-05, 8.44088104e-03, 9.44107742e-05],
        [1.17018724e-04, 4.17125382e-04, 2.49348185e-03, ...,
         8.36507315e-05, 8.44088104e-03, 9.44107742e-05]],

       ...,

       [[8.92622746e-04, 8.24281364e-04, 6.51491515e-04, ...,
         2.69386510e-04, 3.47320316e-03, 2.96128052e-03],
        [2.58860382e-04, 9.77864023e-04, 5.19359484e-04, ...,
         4.62538999e-04, 1.51072303e-03, 1.34733960e-03],
        [9.86914034e-04, 1.78585819e-04, 1.52172684e-03, ...,
         4.31042863e-04, 2.96844955e-04, 6.79351564e-04],
        ...,
        [1.02885824e-04, 1.63706238e-04, 2.08607782e-03, ...,
         4.12734407e-05, 2.38581165e-03, 1.92870823e-04],
        [1.02885824e-04, 1.63706238e-04, 2.08607782e-03, ...,
         4.12734407e-05, 2.38581165e-03, 1.92870837e-04],
        [1.02885824e-04, 1.63706238e-04, 2.08607782e-03, ...,
         4.12734407e-05, 2.38581165e-03, 1.92870823e-04]],
```

```
       ...,
       [2.33066210e-04, 1.59448746e-03, 1.13014027e-03, ...,
        2.15326436e-05, 2.81096133e-03, 1.51375134e-04],
       [2.33066210e-04, 1.59448746e-03, 1.13014027e-03, ...,
        2.15326436e-05, 2.81096133e-03, 1.51375134e-04],
       [2.33066210e-04, 1.59448746e-03, 1.13014027e-03, ...,
        2.15326436e-05, 2.81096133e-03, 1.51375134e-04]],

      [[1.10331795e-03, 5.95025660e-04, 3.74417345e-04, ...,
        6.13579541e-05, 3.35158408e-03, 7.89283891e-04],
       [1.57069473e-04, 4.53210610e-04, 1.73766151e-04, ...,
        1.18845041e-04, 1.98629030e-04, 1.43995113e-03],
       [7.53673958e-04, 3.93289747e-03, 1.64684563e-04, ...,
        1.18359640e-04, 1.91663008e-03, 2.92276818e-04],
       ...,
       [1.15308554e-04, 5.29505778e-04, 1.03027304e-03, ...,
        4.52578888e-06, 7.39003485e-03, 6.57932178e-05],
       [1.15308561e-04, 5.29505836e-04, 1.03027222e-03, ...,
        4.52579161e-06, 7.39003392e-03, 6.57932251e-05],
       [1.15308561e-04, 5.29505836e-04, 1.03027222e-03, ...,
        4.52579161e-06, 7.39003392e-03, 6.57932251e-05]]], dtype=float32)>
```