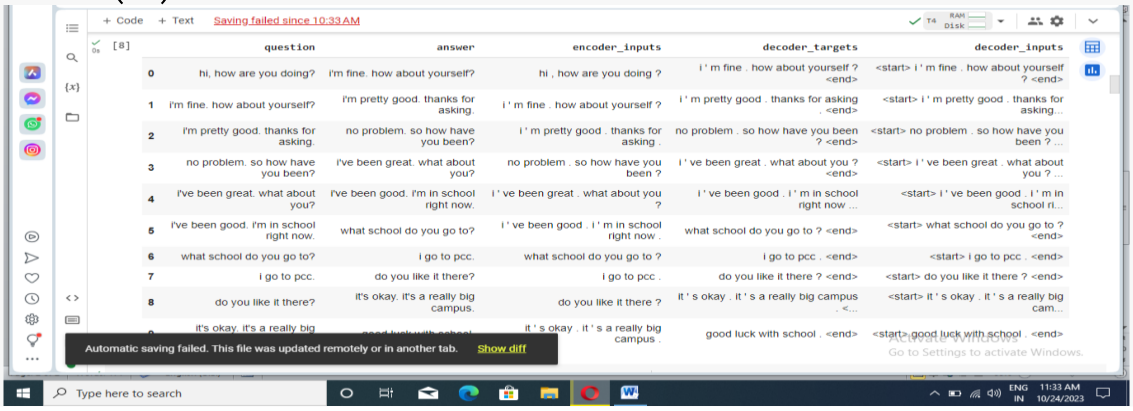
TITLE : CREATE A CHATBOT IN PYTHON

Text Cleaning:

- The code starts by dropping the columns of data that are not needed.
- The code then creates a new column called "encoder_inputs" which is the text from the question and answers it with re.sub() to remove any punctuation, numbers, or symbols. The code drops the columns 'answer tokens','question tokens' and axis=1.
- The decoder input is then created by concatenating the question text with a newline character.

```python
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)
```



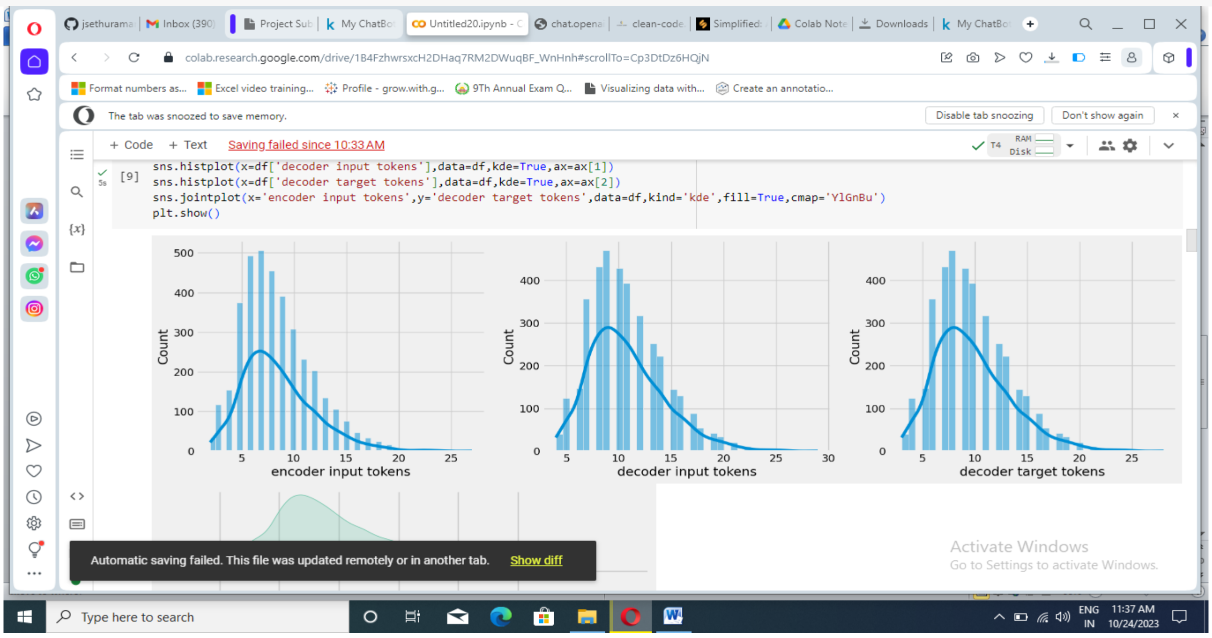
	question	answer	encoder_inputs	decoder_targets	decoder_inputs
0	hi, how are you doing?	i'm fine. how about yourself?	hi , how are you doing ?	i ' m fine . how about yourself ? <end>	<start> i ' m fine . how about yourself ? <end>
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.	i ' 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	good luck with school.	it ' s okay . it ' s a really big campus .	good luck with school . <end>	<start> good luck with school . <end>

Automatic saving failed. This file was updated remotely or in another tab. Show diff

Activate Windows
Go to Settings to activate Windows.

Type here to search
ENG IN 11:33 AM 10/24/2023

```
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()
```



```
[9] 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()
```

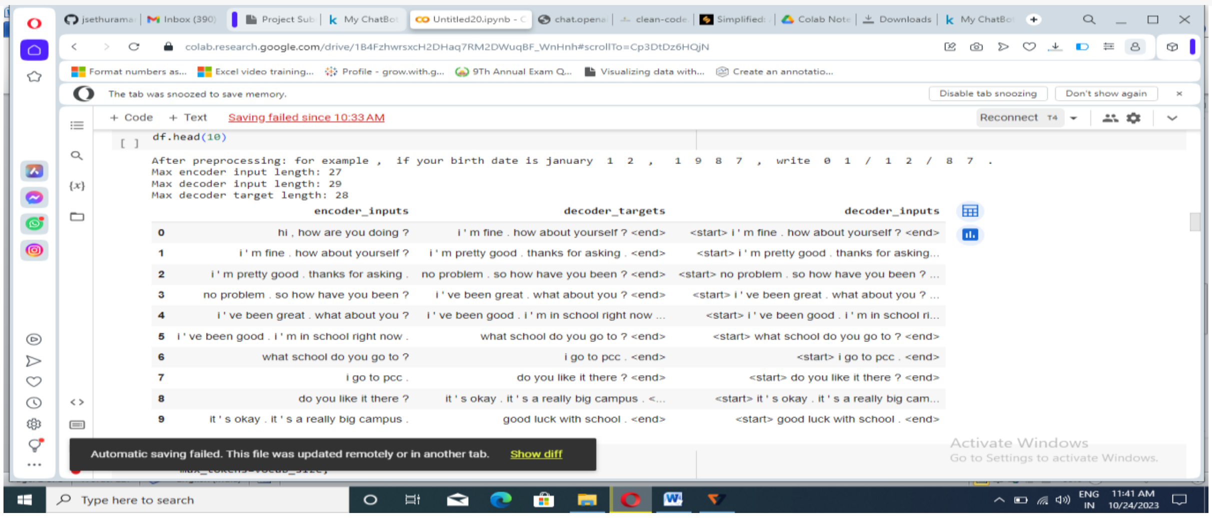
colab.research.google.com/drive/1B4FzhwrsxcH2DHaq7RM2DWuqBF_WnHnh#scrollTo=Cp3DtDz6HQjN
The tab was snoozed to save memory.
Disable tab snoozing
Don't show again
+ Code
+ Text
Saving failed since 10:33 AM
Count
encoder input tokens
decoder input tokens
decoder target tokens
Automatic saving failed. This file was updated remotely or in another tab.
Show diff
Activate Windows
Go to Settings to activate Windows.
Type here to search
11:37 AM
10/24/2023

Encode & decode:

- It then creates a dictionary with all of the parameters that will be used in training and testing.
- The code then defines a function to create an encoder-decoder pair for each question, which is done by creating two lists: one list containing the input tokens from the encoder side and another list containing the output tokens from the decoder side.
- The next step is to define how many sequences are allowed per batch (149).
- Next, it sets up some variables for learning rate, batch size, LSTM cells, vocabulary size, embedding dimensionality and buffer size.

```
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 tok
ens','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)
```



```
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>

Automatic saving failed. This file was updated remotely or in another tab.

Tokenization:

- This is done with the following line: vectorize_layer=TextVectorization(max_tokens=vocab_size, The next step is to create an adaptor that will take in the encoder input and decoder target as well as the start and end of the sentence.
- The adaptor then creates a vocabulary size of 12 words for this sentence. The code first creates a TextVectorization layer that will be used to vectorize the input and output sequences.
- The max_tokens parameter specifies the maximum number of tokens in each sequence, which is set to 12 in this case.

```
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]}')
```

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

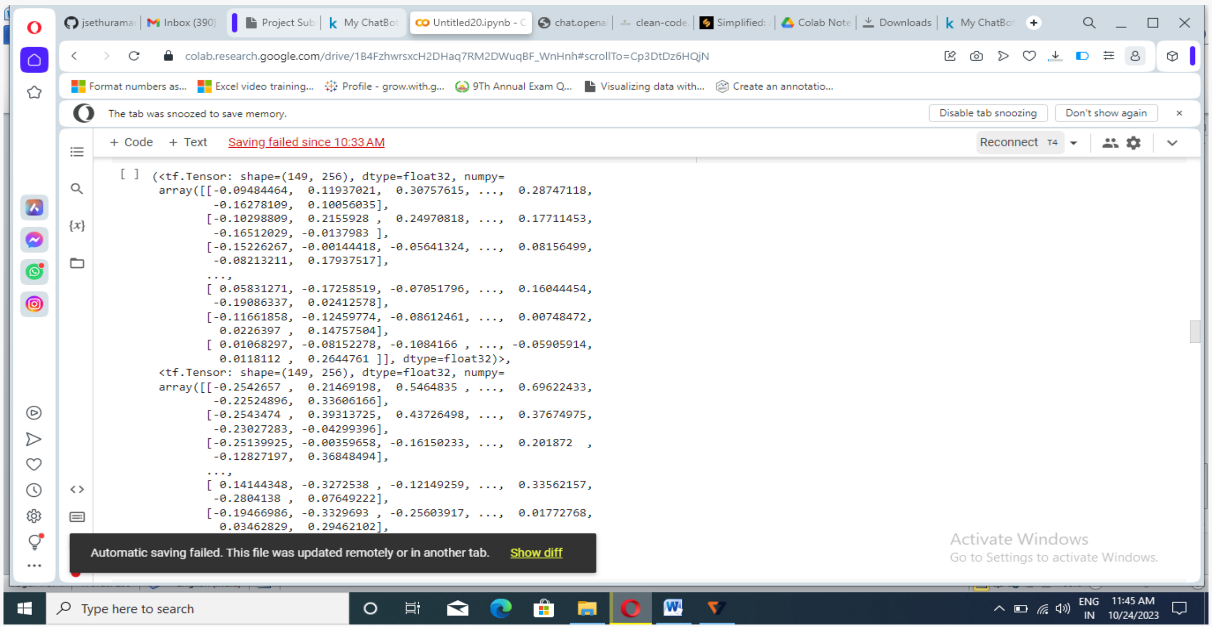
Build Encoder:

- The code starts by creating a new class called Encoder.
- The constructor for the class takes in three arguments: units, embedding_dim, and vocab_size.
- These are all integers that represent the number of input neurons to use in an LSTM layer, the dimensionality of the embedding space (in this case 128), and how many words there are in our vocabulary respectively. We then create another function called call which accepts one argument - encoder inputs - which is what was passed
- The code is an example of how to create an encoder model in Keras.

```
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])
```



```
[ ]  (<tf.Tensor: shape=(149, 256), dtype=float32, numpy=
      array([[-0.09484464,  0.11937021,  0.30757615, ...,  0.28747118,
              -0.16278109,  0.10056035],
             [-0.10298809,  0.2155928 ,  0.24970818, ...,  0.17711453,
              -0.16512029, -0.0137983 ],
             [-0.15226267, -0.00144418, -0.05641324, ...,  0.08156499,
              -0.08213211,  0.17937517],
             ...,
             [ 0.05831271, -0.17258519, -0.07051796, ...,  0.16044454,
              -0.19086337,  0.02412578],
             [-0.11661858, -0.12459774, -0.08612461, ...,  0.00748472,
               0.0226397 ,  0.14757504],
             [ 0.01068297, -0.08152278, -0.1084166 , ..., -0.05905914,
               0.0118112 ,  0.2644761 ]], dtype=float32)>,
      <tf.Tensor: shape=(149, 256), dtype=float32, numpy=
      array([[-0.2542657 ,  0.21469198,  0.5464835 , ...,  0.69622433,
              -0.22524896,  0.33606166],
             [-0.2543474 ,  0.39313725,  0.43726498, ...,  0.37674975,
              -0.23027283, -0.04299396],
             [-0.25139925, -0.00359658, -0.16150233, ...,  0.201872  ,
              -0.12827197,  0.36848494],
             ...,
             [ 0.14144348, -0.3272538 , -0.12149259, ...,  0.33562157,
              -0.2804138 ,  0.07649222],
             [-0.19466986, -0.3329693 , -0.25603917, ...,  0.01772768,
               0.03462829,  0.29462102],
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

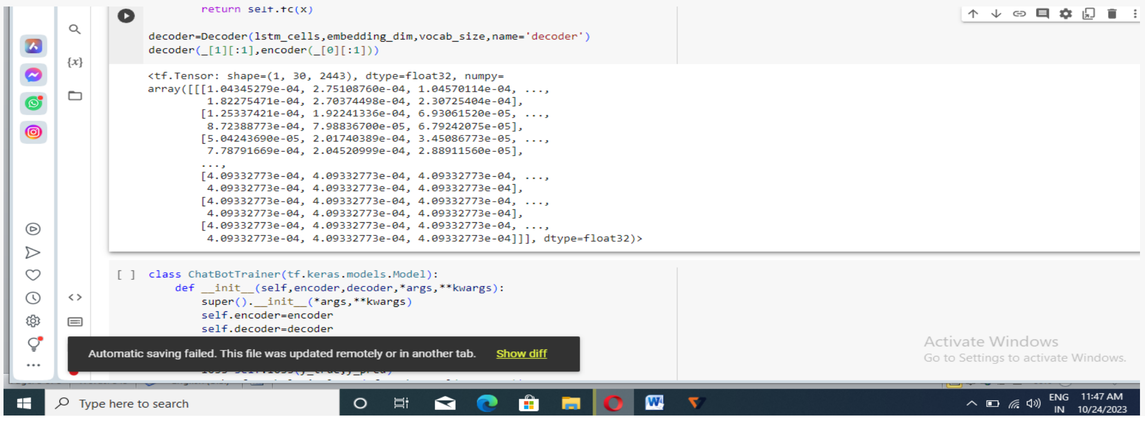
Build Decoder:

- Next, it creates a LayerNormalization layer which is used to normalize the output of the Embedding layer so that all values are between 0 and 1.
- Then it creates an LSTM layer which has units as its number of layers and dropout set at .4 for each hidden state
- The code is a simple example of how to create a Keras model.
- The first line creates an instance of the Decoder class with units, embedding_dim, and vocabulary_size.
- The next line defines the initializer for the layer that will be used in our network.

```
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_st
ates)
        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]))
```



```
        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([[[1.04345279e-04, 2.75108760e-04, 1.04570114e-04, ...,
         1.82275471e-04, 2.70374498e-04, 2.30725404e-04],
        [1.25337421e-04, 1.92241336e-04, 6.93061520e-05, ...,
         8.72388773e-04, 7.98836700e-05, 6.79242075e-05],
        [5.04243690e-05, 2.01740389e-04, 3.45086773e-05, ...,
         7.78791669e-04, 2.04520999e-04, 2.88911560e-05],
        ...,
        [4.09332773e-04, 4.09332773e-04, 4.09332773e-04, ...,
         4.09332773e-04, 4.09332773e-04, 4.09332773e-04],
        [4.09332773e-04, 4.09332773e-04, 4.09332773e-04, ...,
         4.09332773e-04, 4.09332773e-04, 4.09332773e-04],
        [4.09332773e-04, 4.09332773e-04, 4.09332773e-04, ...,
         4.09332773e-04, 4.09332773e-04, 4.09332773e-04]]], dtype=float32)>
```

```
[ ] class ChatBotTrainer(tf.keras.models.Model):
        def __init__(self,encoder,decoder,*args,**kwargs):
            super().__init__(*args,**kwargs)
            self.encoder=encoder
            self.decoder=decoder
```

Automatic saving failed. This file was updated remotely or in another tab. Show diff

Train Model:

- It then defines a function to fit the model on the training data, and another function to save it in a checkpoint file.
- The code then creates an instance of tf.keras.Model with train_data and epochs=100 .
- Then it calls fit() , which takes two arguments: train_data : A list of tensors that represent input features for each observation in the dataset epochs
- We also create callbacks for TensorBoard and ModelCheckpoint .
- The code is a code for training and validating the model.
- This code trains the model with 100 epochs and saves the best performing model at each epoch.

```
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=Tru
e)
    ]
)
```

```
Epoch 1/100
23/23 [==============================] - ETA: 0s - loss: 1.6485 - accuracy: 0.
2192
Epoch 1: val_loss improved from inf to 1.34765, saving model to ckpt
23/23 [==============================] - 43s 1s/step - loss: 1.6394 - accuracy
: 0.2214 - val_loss: 1.3477 - val_accuracy: 0.2801
Epoch 2/100
23/23 [==============================] - ETA: 0s - loss: 1.2380 - accuracy: 0.
3101
Epoch 2: val_loss improved from 1.34765 to 1.08449, saving model to ckpt
23/23 [==============================] - 28s 1s/step - loss: 1.2367 - accuracy
: 0.3099 - val_loss: 1.0845 - val_accuracy: 0.3406
Epoch 3/100
23/23 [==============================] - ETA: 0s - loss: 1.1060 - accuracy: 0.
3365
Epoch 3: val_loss improved from 1.08449 to 1.03598, saving model to ckpt
23/23 [==============================] - 28s 1s/step - loss: 1.1007 - accuracy
: 0.3377 - val_loss: 1.0360 - val_accuracy: 0.3656
Epoch 100: val_loss did not improve from 0.35408
23/23 [==============================] - 1s 30ms/step - loss: 0.3441 - accurac
y: 0.6755 - val_loss: 0.4577 - val_accuracy: 0.6582
```

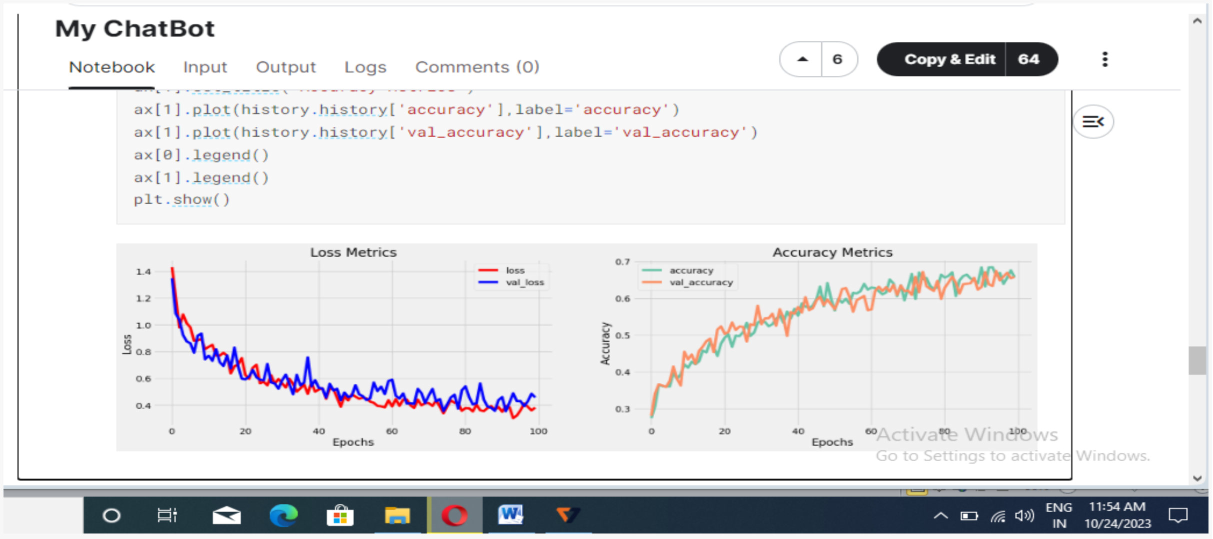
Visualize Metrics:

- The code starts by creating a list of the history.history objects in order to plot them on the same graph.
- The code then creates two subplots, one for loss and one for accuracy, with each plotted over an epoch.

- The x-axis is labeled "Epochs" and the y-axis is labeled "Loss"
- Then it plots the history objects' values on those axes using ax[0].plot(history.history['loss'],label='loss',c='red') and ax[1].plot(history.history['accuracy'],label='accuracy').
- Finally, it shows both graphs with plt.show().

```
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()
```



```
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()
```