Create a chatbot in python

Phase 4: Development part 2

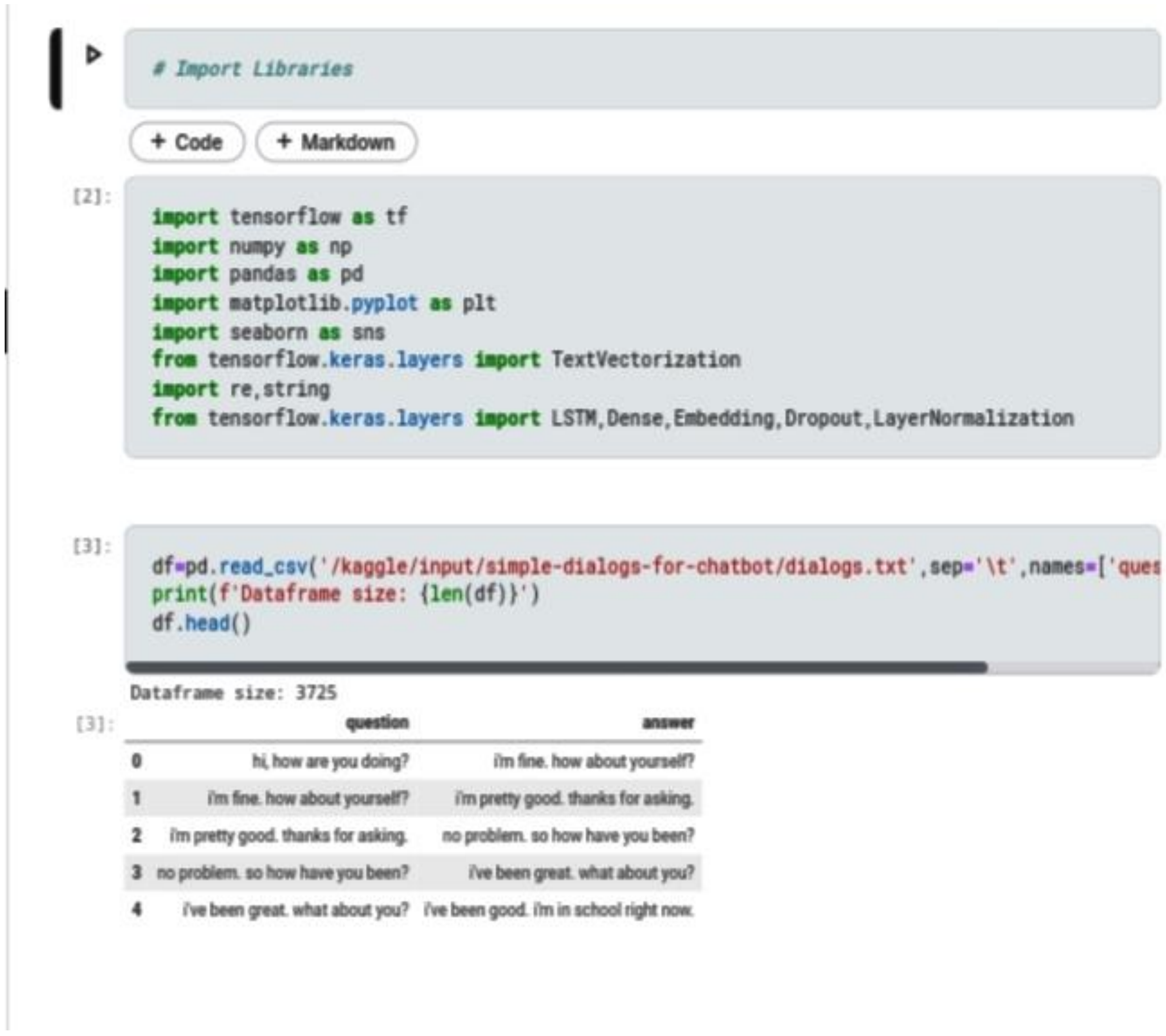
Introduction :

Model training and evaluation in a chatbot using Python involves teaching the chatbot how to understand and generate human-like responses. It's like teaching a virtual assistant to have conversation. Training is the process of teaching the chatbot using a dataset, and evaluation checks how well it performs in conversations.

Dataset used:

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

Program:



```
# Import Libraries
```

```
[2]:
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
```

```
[3]:
df=pd.read_csv('/kaggle/input/simple-dialogs-for-chatbot/dialogs.txt',sep='\t',names=['ques
print(f'Dataframe size: {len(df)}')
df.head()
```

```
Dataframe size: 3725
```

[3]:

	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.

Model building:

- Creating a machine learning or deep learning model to predict responses based on input.
- Choose or design a language model. This can be based on rule-based systems, machine learning models (like Recurrent Neural Networks or Transformers), or a combination.

Build Models

Build Encoder

[ ]:

```
```

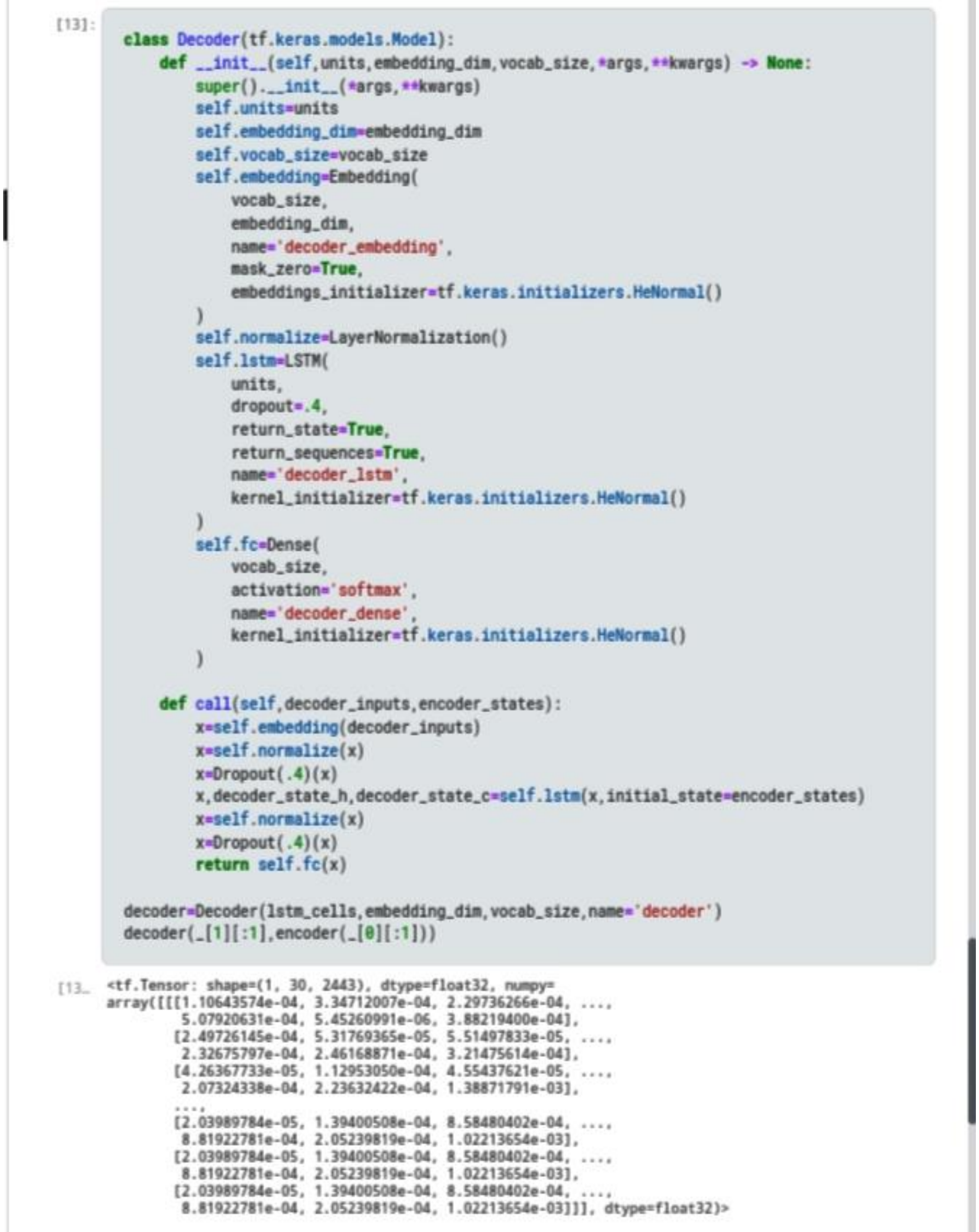
[12]:

```python
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])
```

[12…

```
(<tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[ 0.09431175, -0.12710264, -0.08852765, ...,  0.12329073,
         -0.17205271, -0.02656413],
        [ 0.0366795 ,  0.10652992,  0.22317867, ...,  0.13967358,
         -0.21897762,  0.08331957],
        [ 0.00859349,  0.06553305,  0.27781972, ..., -0.04496698,
         -0.26204205,  0.18547165],
        ...,
        [ 0.1779747 , -0.0239244 ,  0.0179972 , ..., -0.11767799,
         -0.00430955,  0.07254015],
        [-0.04961572, -0.11756462,  0.05006162, ...,  0.01444003,
         -0.14937747,  0.12971587],
        [-0.21961676, -0.09170805,  0.2791963 , ..., -0.08595876,
         -0.16397955,  0.03669236]], dtype=float32)>,
 <tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[ 0.17323941, -0.1730998 , -0.21646923, ...,  0.21333084,
         -0.2828206 , -0.0437258 ],
        [ 0.07775044,  0.20444411,  0.35847616, ...,  0.40880555,
         -0.39606556,  0.15911165],
        [ 0.01983667,  0.12375534,  0.45666823, ..., -0.13082409,
         -0.49876744,  0.36550373],
        ...,
        [ 0.40776315, -0.04699488,  0.02834211, ..., -0.3351813 ,
         -0.00713065,  0.14568232],
        [-0.10508712, -0.23727265,  0.07724902, ...,  0.03975456,
         -0.25708586,  0.24848795],
        [-0.53576005, -0.17162281,  0.45328242, ..., -0.23305744,
         -0.27900976,  0.07027075]], dtype=float32)>)
```

Build Encoder## Build Decoder



```
[13]:
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]))
```

```
[13_ <tf.Tensor: shape=(1, 30, 2443), dtype=float32, numpy=
array([[[1.10643574e-04, 3.34712007e-04, 2.29736266e-04, ...,
         5.07920631e-04, 5.45260991e-06, 3.88219400e-04],
        [2.49726145e-04, 5.31769365e-05, 5.51497833e-05, ...,
         2.32675797e-04, 2.46168871e-04, 3.21475614e-04],
        [4.26367733e-05, 1.12953050e-04, 4.55437621e-05, ...,
         2.07324338e-04, 2.23632422e-04, 1.38871791e-03],
        ...,
        [2.03989784e-05, 1.39400508e-04, 8.58480402e-04, ...,
         8.81922781e-04, 2.05239819e-04, 1.02213654e-03],
        [2.03989784e-05, 1.39400508e-04, 8.58480402e-04, ...,
         8.81922781e-04, 2.05239819e-04, 1.02213654e-03],
        [2.03989784e-05, 1.39400508e-04, 8.58480402e-04, ...,
         8.81922781e-04, 2.05239819e-04, 1.02213654e-03]]], dtype=float32)>
```

Train the model :

- Gather a dataset of conversations or create your own dataset.
- Preprocess the data by cleaning and tokenizing the text.
- Split the dataset into training and testing sets.
- Choose a suitable machine learning algorithm, such as a sequence-to-sequence model or a transformer model.
- Implement the model using a deep learning framework like TensorFlow or PyTorch.
- Train the model on the training dataset, adjusting hyperparameters as needed.
- Evaluate the model’s performance on the testing dataset to measure its accuracy.
- Fine-tune the model if necessary by iterating on the training process.
- Save the trained model for future use.

Build Training Model

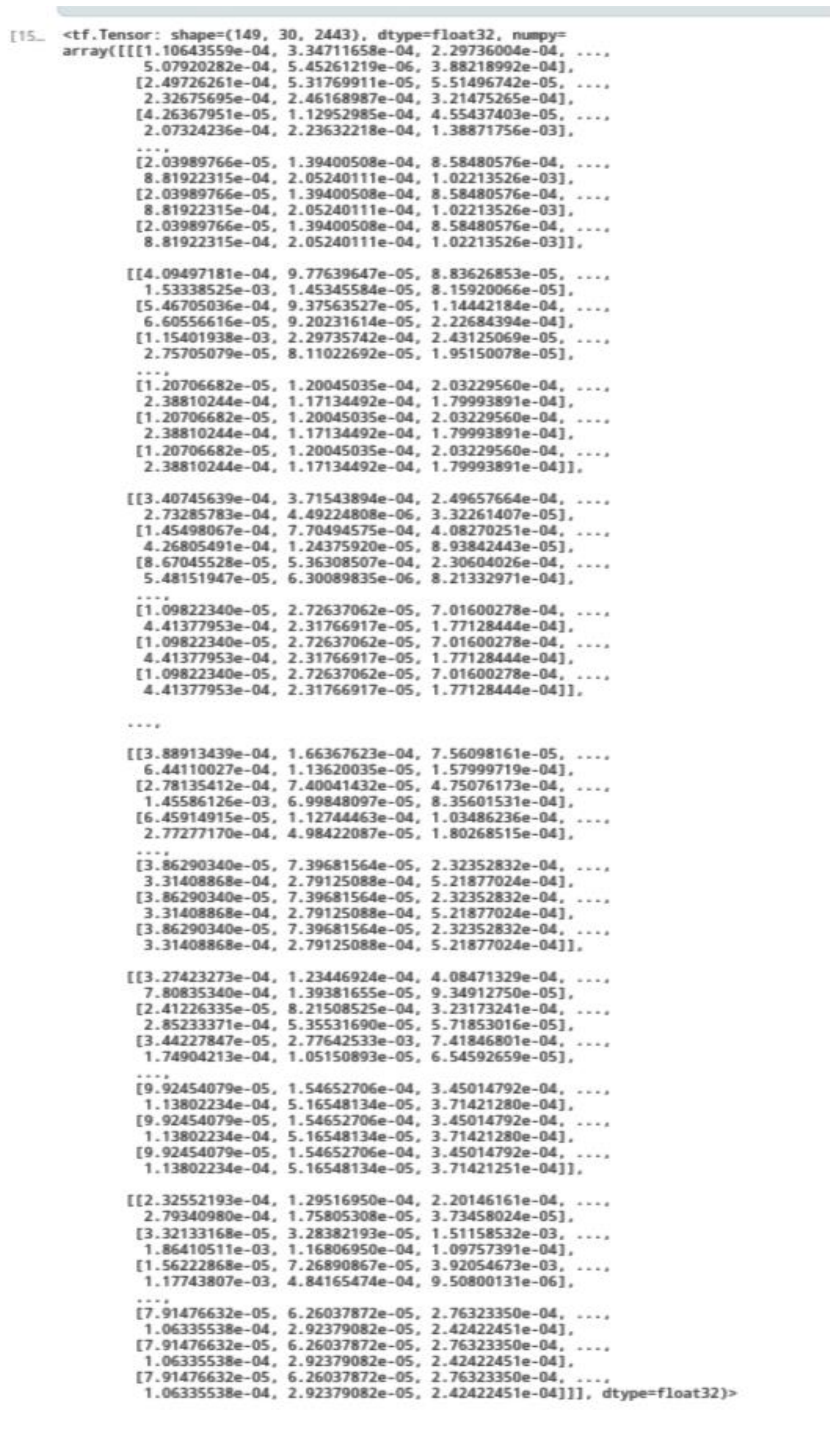
[14]:

```
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
```

[15]:

```
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, 3443), dtype=float32, numpy=
```



```
[15_ <tf.Tensor: shape=(149, 30, 2443), dtype=float32, numpy=
     array([[[1.10643559e-04, 3.34711658e-04, 2.29736004e-04, ...,
              5.07920282e-04, 5.45261219e-06, 3.88218992e-04],
             [2.49726261e-04, 5.31769911e-05, 5.51496742e-05, ...,
              2.32675695e-04, 2.46168987e-04, 3.21475265e-04],
             [4.26367951e-05, 1.12952985e-04, 4.55437403e-05, ...,
              2.07324236e-04, 2.23632218e-04, 1.38871756e-03],
             ...,
             [2.03989766e-05, 1.39400508e-04, 8.58480576e-04, ...,
              8.81922315e-04, 2.05240111e-04, 1.02213526e-03],
             [2.03989766e-05, 1.39400508e-04, 8.58480576e-04, ...,
              8.81922315e-04, 2.05240111e-04, 1.02213526e-03],
             [2.03989766e-05, 1.39400508e-04, 8.58480576e-04, ...,
              8.81922315e-04, 2.05240111e-04, 1.02213526e-03]],

            [[4.09497181e-04, 9.77639647e-05, 8.83626853e-05, ...,
              1.53338525e-03, 1.45345584e-05, 8.15920066e-05],
             [5.46705036e-04, 9.37563527e-05, 1.14442184e-04, ...,
              6.60556616e-05, 9.20231614e-05, 2.22684394e-04],
             [1.15401938e-03, 2.29735742e-04, 2.43125069e-05, ...,
              2.75705079e-05, 8.11022692e-05, 1.95150078e-05],
             ...,
             [1.20706682e-05, 1.20045035e-04, 2.03229560e-04, ...,
              2.38810244e-04, 1.17134492e-04, 1.79993891e-04],
             [1.20706682e-05, 1.20045035e-04, 2.03229560e-04, ...,
              2.38810244e-04, 1.17134492e-04, 1.79993891e-04],
             [1.20706682e-05, 1.20045035e-04, 2.03229560e-04, ...,
              2.38810244e-04, 1.17134492e-04, 1.79993891e-04]],

            [[3.40745639e-04, 3.71543894e-04, 2.49657664e-04, ...,
              2.73285783e-04, 4.49224808e-06, 3.32261407e-05],
             [1.45498067e-04, 7.70494575e-04, 4.08270251e-04, ...,
              4.26805491e-04, 1.24375920e-05, 8.93842443e-05],
             [8.67045528e-05, 5.36308507e-04, 2.30604026e-04, ...,
              5.48151947e-05, 6.30089835e-06, 8.21332971e-04],
             ...,
             [1.09822340e-05, 2.72637062e-05, 7.01600278e-04, ...,
              4.41377953e-04, 2.31766917e-05, 1.77128444e-04],
             [1.09822340e-05, 2.72637062e-05, 7.01600278e-04, ...,
              4.41377953e-04, 2.31766917e-05, 1.77128444e-04],
             [1.09822340e-05, 2.72637062e-05, 7.01600278e-04, ...,
              4.41377953e-04, 2.31766917e-05, 1.77128444e-04]],

            ...,

            [[3.88913439e-04, 1.66367623e-04, 7.56098161e-05, ...,
              6.44110027e-04, 1.13620035e-05, 1.57999719e-04],
             [2.78135412e-04, 7.40041432e-05, 4.75076173e-04, ...,
              1.45586126e-03, 6.99848097e-05, 8.35601531e-04],
             [6.45914915e-05, 1.12744463e-04, 1.03486236e-04, ...,
              2.77277170e-04, 4.98422087e-05, 1.80268515e-04],
             ...,
             [3.86290340e-05, 7.39681564e-05, 2.32352832e-04, ...,
              3.31408868e-04, 2.79125088e-04, 5.21877024e-04],
             [3.86290340e-05, 7.39681564e-05, 2.32352832e-04, ...,
              3.31408868e-04, 2.79125088e-04, 5.21877024e-04],
             [3.86290340e-05, 7.39681564e-05, 2.32352832e-04, ...,
              3.31408868e-04, 2.79125088e-04, 5.21877024e-04]],

            [[3.27423273e-04, 1.23446924e-04, 4.08471329e-04, ...,
              7.80835340e-04, 1.39381655e-05, 9.34912750e-05],
             [2.41226335e-05, 8.21508525e-04, 3.23173241e-04, ...,
              2.85233371e-04, 5.35531690e-05, 5.71853016e-05],
             [3.44227847e-05, 2.77642533e-03, 7.41846801e-04, ...,
              1.74904213e-04, 1.05150893e-05, 6.54592659e-05],
             ...,
             [9.92454079e-05, 1.54652706e-04, 3.45014792e-04, ...,
              1.13802234e-04, 5.16548134e-05, 3.71421280e-04],
             [9.92454079e-05, 1.54652706e-04, 3.45014792e-04, ...,
              1.13802234e-04, 5.16548134e-05, 3.71421280e-04],
             [9.92454079e-05, 1.54652706e-04, 3.45014792e-04, ...,
              1.13802234e-04, 5.16548134e-05, 3.71421251e-04]],

            [[2.32552193e-04, 1.29516950e-04, 2.20146161e-04, ...,
              2.79340980e-04, 1.75805308e-05, 3.73458024e-05],
             [3.32133168e-05, 3.28382193e-05, 1.51158532e-03, ...,
              1.86410511e-03, 1.16806950e-04, 1.09757391e-04],
             [1.56222868e-05, 7.26890867e-05, 3.92054673e-03, ...,
              1.17743807e-03, 4.84165474e-04, 9.50800131e-06],
             ...,
             [7.91476632e-05, 6.26037872e-05, 2.76323350e-04, ...,
              1.06335538e-04, 2.92379082e-05, 2.42422451e-04],
             [7.91476632e-05, 6.26037872e-05, 2.76323350e-04, ...,
              1.06335538e-04, 2.92379082e-05, 2.42422451e-04],
             [7.91476632e-05, 6.26037872e-05, 2.76323350e-04, ...,
              1.06335538e-04, 2.92379082e-05, 2.42422451e-04]]], dtype=float32)>
```

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)
    ]
)
```

```
Epoch 1/100
23/23 [==============================] - ETA: 0s - loss: 1.6623 - accuracy: 0.2185
Epoch 1: val_loss improved from inf to 1.21886, saving model to ckpt
23/23 [==============================] - 80s 3s/step - loss: 1.6524 - accuracy: 0.2206 - val_loss:
1.2189 - val_accuracy: 0.2890
Epoch 2/100
 4/23 [====>.........................] - ETA: 20s - loss: 1.3146 - accuracy: 0.2886
```

Visualize Metrics

```
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')
```

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

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,encode

        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_i
        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_decod
        )
        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]
    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
```

[ ]:

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

Time to Chat

[ ]:

```
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.'''
])
```

Model evaluation :

- To evaluate a chatbot model, we can use various metrics like accuracy, precision, recall, and F1 score. Additionally, we can perform qualitative analysis by manually reviewing the chatbot’s responses.
- Another important aspect of model evaluation for a chatbot in Python is conducting user testing. This involves getting feedback from real users to assess the chatbot’s performance, usability, and overall user satisfaction.
- User testing can provide valuable insights into areas of improvement and help refine the chatbot’s responses.