```
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


In [ ]:
df=pd.read_csv('/content/convert.csv',sep='\t',names=['question','answer'])
print(f'Dataframe size: {len(df)}')
df.head()



Dataframe size: 3725

Out[ ]:

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.

Data Preprocessing

Data Visualization

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

Text Cleaning

In [ ]:
```
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)
```

Out[ ]:

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 [ ]:
```
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()
```

In [ ]:

```
print(f"After preprocessing: {' '.join(df[df['encoder input tokens'].max()==df['encoder input tokens']]['encoder_inputs'].values.tolist())}")
print(f"Max encoder input length: {df['encoder input tokens'].max()}")
print(f"Max decoder input length: {df['decoder input tokens'].max()}")
print(f"Max decoder target length: {df['decoder target tokens'].max()}")

df.drop(columns=['question','answer','encoder input tokens','decoder input tokens','decoder target tokens'],axis=1,inplace=True)
params={
    "vocab_size":2500,
    "max_sequence_length":30,
    "learning_rate":0.008,
    "batch_size":149,
    "lstm_cells":256,
    "embedding_dim":256,
    "buffer_size":10000
}
learning_rate=params['learning_rate']
batch_size=params['batch_size']
embedding_dim=params['embedding_dim']
lstm_cells=params['lstm_cells']
vocab_size=params['vocab_size']
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

Out[ ]:

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>

Tokenization

In [ ]:

```
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 [ ]:

```
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 [ ]:

```
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] ...
```

In [ ]:

```
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)

Build Models

Build Encoder

In [ ]:

In [ ]:

```
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[ ]:
(<tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[ 0.00683492, -0.12691063,  0.03438168, ..., -0.19123387,
         -0.03266538,  0.01474974],
        [ 0.26062647,  0.00872722,  0.01738667, ..., -0.15847038,
          0.10110811, -0.00113067],
        [ 0.10284232,  0.10111945, -0.1827511 , ..., -0.23390858,
         -0.04210129,  0.12075958],
        ...,
        [ 0.07773641,  0.14510773,  0.01463656, ..., -0.12716043,
          0.1294901 ,  0.12533897],
        [-0.02646102, -0.13782485, -0.26884595, ..., -0.13503425,
          0.05957591,  0.06664355],
        [ 0.13366045, -0.00506709, -0.07805239, ..., -0.28715044,
          0.12336213,  0.09981678]], dtype=float32)>,
 <tf.Tensor: shape=(149, 256), dtype=float32, numpy=
 array([[ 0.01340534, -0.28561354,  0.06181405, ..., -0.31711257,
         -0.06295014,  0.04365762],
        [ 0.5745282 ,  0.01914767,  0.03018959, ..., -0.26132357,
```

```
  0.18548957, -0.00365305],
 [ 0.214178  ,  0.22660005, -0.34121054, ..., -0.40459988,
  -0.0801721 ,  0.38690996],
 ...,
 [ 0.15541458,  0.32465366,  0.02621953, ..., -0.1997912 ,
   0.24276456,  0.4055249 ],
 [-0.05272985, -0.29396173, -0.51504457, ..., -0.22166635,
   0.10580399,  0.19867381],
 [ 0.2815432 , -0.0115253 , -0.1385937 , ..., -0.50680274,
   0.23774335,  0.29312295]], dtype=float32)>)
```

Build Encoder## Build Decoder

In [ ]:

```
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[ ]:
<tf.Tensor: shape=(1, 30, 2443), dtype=float32, numpy=
array([[[7.35320828e-06, 2.03370728e-04, 1.30172208e-04, ...,
         5.36758707e-05, 3.80577898e-04, 2.81948924e-05],
        [1.04525934e-04, 3.62109604e-05, 8.12704893e-05, ...,
         4.62053483e-03, 3.41308332e-04, 3.56241726e-05],
        [2.79375701e-04, 1.78691378e-04, 2.61527548e-05, ...,
         2.22322834e-03, 2.17821755e-04, 8.10659956e-04],
        ...,
        [1.36487652e-03, 1.47994282e-03, 5.33279264e-04, ...,
         3.74978263e-05, 1.43658062e-05, 2.47654225e-05],
        [1.36487652e-03, 1.47994282e-03, 5.33279264e-04, ...,
         3.74978263e-05, 1.43658062e-05, 2.47654225e-05],
        [1.36487652e-03, 1.47994282e-03, 5.33279264e-04, ...,
         3.74978263e-05, 1.43658062e-05, 2.47654225e-05]]], dtype=float32)>
```

Build Training Model

In [ ]:

```
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


In [ ]:
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[ ]:
<tf.Tensor: shape=(149, 30, 2443), dtype=float32, numpy=
array([[[7.35321146e-06, 2.03370611e-04, 1.30172397e-04, ...,
         5.36758671e-05, 3.80577345e-04, 2.81948905e-05],
        [1.04525869e-04, 3.62109713e-05, 8.12703947e-05, ...,
         4.62053064e-03, 3.41308361e-04, 3.56241508e-05],
        [2.79375789e-04, 1.78691378e-04, 2.61527675e-05, ...,
```

```
  2.22322741e-03, 2.17821856e-04, 8.10659549e-04],
 ...,
 [1.36487652e-03, 1.47994317e-03, 5.33278857e-04, ...,
  3.74978335e-05, 1.43658153e-05, 2.47654280e-05],
 [1.36487652e-03, 1.47994317e-03, 5.33278857e-04, ...,
  3.74978335e-05, 1.43658153e-05, 2.47654280e-05],
 [1.36487652e-03, 1.47994317e-03, 5.33278857e-04, ...,
  3.74978335e-05, 1.43658153e-05, 2.47654280e-05]],

[[8.12307917e-06, 2.26678312e-04, 3.35824676e-04, ...,
  6.44014290e-05, 2.77780171e-04, 1.95703396e-05],
 [1.43197889e-04, 1.88543141e-04, 8.58079438e-05, ...,
  2.27473502e-05, 2.25123833e-04, 2.67564872e-04],
 [6.68397872e-04, 9.65616055e-05, 1.64049707e-04, ...,
  8.83801840e-06, 1.89825296e-04, 1.03418977e-04],
 ...,
 [2.33584229e-04, 1.18358024e-04, 1.45454134e-04, ...,
  9.25302855e-04, 1.11702277e-04, 5.09256170e-05],
 [2.33584229e-04, 1.18358024e-04, 1.45454134e-04, ...,
  9.25302855e-04, 1.11702277e-04, 5.09256170e-05],
 [2.33584229e-04, 1.18358024e-04, 1.45454134e-04, ...,
  9.25302855e-04, 1.11702277e-04, 5.09256170e-05]],

[[3.91767344e-06, 1.78576491e-04, 2.50676123e-04, ...,
  5.22288246e-05, 6.84133498e-04, 2.06620516e-05],
 [1.14590039e-04, 2.39925692e-03, 1.99405458e-05, ...,
  3.82910686e-04, 1.80035341e-03, 5.46994015e-05],
 [4.60276875e-04, 5.12765779e-04, 8.05553718e-05, ...,
  5.52555139e-04, 1.96732013e-04, 4.01684738e-05],
 ...,
 [6.25975896e-04, 2.15928478e-04, 3.59277765e-05, ...,
  5.02479597e-05, 4.56685702e-05, 1.72647342e-04],
 [6.25975896e-04, 2.15928478e-04, 3.59277765e-05, ...,
  5.02479597e-05, 4.56685702e-05, 1.72647342e-04],
 [6.25975896e-04, 2.15928478e-04, 3.59277765e-05, ...,
  5.02479597e-05, 4.56685702e-05, 1.72647342e-04]],
```

...,

[[1.80345269e-05, 3.15794081e-04, 6.76140480e-04, ...,
6.83017497e-05, 2.19673981e-04, 2.63155507e-05],
[2.12730683e-05, 2.58617714e-04, 4.70598228e-04, ...,
3.06146161e-04, 7.53366257e-05, 5.34516330e-05],
[1.41523793e-04, 1.33979760e-04, 5.13620151e-04, ...,
2.08758749e-04, 8.80955922e-05, 1.61569205e-05],
...,
[8.73827841e-04, 5.58517459e-05, 6.65809959e-04, ...,
1.19724515e-04, 2.03713444e-05, 8.87004717e-05],
[8.73827841e-04, 5.58517459e-05, 6.65809959e-04, ...,
1.19724515e-04, 2.03713444e-05, 8.87004717e-05],
[8.73827841e-04, 5.58517459e-05, 6.65809959e-04, ...,
1.19724515e-04, 2.03713444e-05, 8.87004717e-05]],

[[6.42919213e-06, 9.36520591e-05, 2.74398248e-04, ...,
5.37733431e-05, 2.13372172e-04, 2.46061099e-05],
[1.51957420e-05, 2.13956562e-04, 1.73553562e-04, ...,
1.51805769e-04, 8.84862384e-04, 1.24010257e-04],
[1.04202758e-04, 1.33096677e-04, 4.13597336e-05, ...,
1.37053532e-04, 7.33523397e-04, 9.79960023e-05],
...,
[1.59213215e-03, 6.51407681e-05, 8.05182062e-05, ...,
7.67890306e-05, 3.83427941e-05, 5.87052455e-05],
[1.59213215e-03, 6.51407681e-05, 8.05182062e-05, ...,
7.67890306e-05, 3.83427941e-05, 5.87052455e-05],
[1.59213215e-03, 6.51407681e-05, 8.05182062e-05, ...,
7.67890306e-05, 3.83427941e-05, 5.87052455e-05]],

[[7.17818239e-06, 5.90792391e-04, 4.71014471e-04, ...,
8.07284523e-05, 3.83246283e-04, 1.69587929e-05],
[1.32166970e-05, 8.53058882e-04, 2.15214954e-04, ...,
1.93219763e-04, 8.04165786e-04, 5.38237000e-05],
[6.18584600e-05, 1.66647384e-04, 1.23752441e-04, ...,
6.42304294e-05, 6.39051723e-04, 1.68932951e-04],

```
       ...,
       [3.09769576e-03, 5.42585549e-05, 5.77429659e-04, ...,
        5.29697718e-05, 5.21456132e-05, 9.70589826e-05],
       [3.09769576e-03, 5.42585549e-05, 5.77429659e-04, ...,
        5.29697718e-05, 5.21456132e-05, 9.70589826e-05],
       [3.09769576e-03, 5.42585549e-05, 5.77429659e-04, ...,
        5.29697718e-05, 5.21456132e-05, 9.70589826e-05]]], dtype=float32)>
```

Train Model

In [ ]:

```
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: 1.6530 - accuracy: 0.2194
Epoch 1: val_loss improved from inf to 1.27846, saving model to ckpt
```

Visualize Metrics

In [ ]:

```
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

In [ ]:

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

In [ ]:

```
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

In [ ]:

```
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)
    for i,word in enumerate(text.split()):
        seq[:,i]=sequences2ids(word).numpy()[0]
    return seq

def postprocess(self,text):
    text=re.sub(' - ','-',text.lower())
    text=re.sub(' [.] ','. ',text)
    text=re.sub(' [1] ','1',text)
    text=re.sub(' [2] ','2',text)
    text=re.sub(' [3] ','3',text)
    text=re.sub(' [4] ','4',text)
    text=re.sub(' [5] ','5',text)
    text=re.sub(' [6] ','6',text)
    text=re.sub(' [7] ','7',text)
    text=re.sub(' [8] ','8',text)
    text=re.sub(' [9] ','9',text)
    text=re.sub(' [0] ','0',text)
    text=re.sub(' [,] ',', ',text)
    text=re.sub(' [?] ','? ',text)
    text=re.sub(' [!] ','! ',text)
    text=re.sub(' [$] ','$ ',text)
    text=re.sub(' [&] ','& ',text)
    text=re.sub(' [/] ','/ ',text)
    text=re.sub(' [:] ',': ',text)
    text=re.sub(' [;] ','; ',text)
    text=re.sub(' [*] ','* ',text)
    text=re.sub(' [\'] ','\"',text)
    text=re.sub(' [\"] ','\'',text)
    return text

def call(self,text,config=None):
    input_seq=self.preprocess(text)
    states=self.encoder(input_seq,training=False)
```

```
        target_seq=np.zeros((1,1))
        target_seq[:,:]=sequences2ids(['<start>']).numpy()[0][0]
        stop_condition=False
        decoded=[]
        while not stop_condition:
            decoder_outputs,new_states=self.decoder([target_seq,states],training=False)
#             index=tf.argmax(decoder_outputs[:,-1,:],axis=-1).numpy().item()
            index=self.sample(decoder_outputs[0,0,:]).item()
            word=ids2sequences([index])
            if word=='<end> ' or len(decoded)>=max_sequence_length:
                stop_condition=True
            else:
                decoded.append(index)
                target_seq=np.zeros((1,1))
                target_seq[:,:]=index
                states=new_states
        return self.postprocess(ids2sequences(decoded))

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


In [ ]:
tf.keras.utils.plot_model(chatbot.encoder,to_file='encoder.png',show_shapes=True,show_layer_activations=True)


In [ ]:
tf.keras.utils.plot_model(chatbot.decoder,to_file='decoder.png',show_shapes=True,show_layer_activations=True)



Time to Chat

In [ ]:
def print_conversation(texts):
    for text in texts:
```

```
        print(f'You: {text}')
        print(f'Bot: {chatbot(text)}')
        print('========================')


In [ ]:
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.'"
])
```