**Phase 5**

**PROJECT DOCUMENTATION & SUBMISSION**

|  |  |
| --- | --- |
| **Date** | **31-10-2023** |
| **Team ID** | **3918** |
| **Project Name** | **Create a Chatbot using Python** |

**Problem Statement :**

Artificial intelligence chatbot is a technology that makes interactions between man and machines using natural language possible. A chatbot can give different responses from the same input given by the user according to the current conversation issue". By using our "Intelligent ChatBot" you can overcome all the above-given issues, you do not need humans to do manual work, your clients will be happy.A chatbot is a conventional agent that is capable to communicate with operators by using natural languages. As numerous chatbot platforms already exist, there are still some problems in building data-driven system because a huge amount of data is required for their development.

**Design Thinking Process:**

1. Empathize:

- Understand the pain points of the existing customer support system.

- Analyze customer feedback, support ticket data, and user needs.

- Identify the need for 24/7 support and quick query resolution.

2. Define:

- Define the primary goals: improve customer support and engagement.

- Specify key features: chatbot, knowledge base, FAQ, and real-time chat support.

- Determine target users: customers seeking information and support.

3. Ideate:

- Brainstorm solutions to meet the defined goals.

- Consider integrating a chatbot for quick responses.

- Plan the architecture and user interface for the web application.

4. Prototype:

- Create a mockup of the web application's user interface.

- Design the chatbot's conversational flow.

- Develop a knowledge base and FAQs.

5. Test:

- Gather feedback from potential users on the prototype.

- Adjust the user interface and chatbot's flow based on user feedback.

- Ensure usability and performance.

**Phases of Development:**

1. Backend Development:

- Here we use python stack

- And Develope a database for storing product information and user data.

- Implement a RESTful API for the web application.

2. Chatbot Development:

- Here we use library like Rasa, Dialogflow, or Microsoft Bot Framework.

- Then train the chatbot using NLP techniques and the company's FAQ.

- By using NLP technique we can implement natural language understanding for complex queries.

- Integrate chatbot with the web application via API.

3. Web Application Development:

- Design the user interface for the web application.

- Implement real-time chat support with the chatbot.

- Develop a knowledge base and FAQ section.

- Ensure a responsive and user-friendly design.

4. **NLP Integration:**

- Utilize libraries like spaCy, NLTK, or transformers for NLP tasks.

- Implement sentiment analysis for customer feedback.

- Integrate entity recognition for product-related queries.

- Continuously improve chatbot's language understanding using machine learning.

**Innovative Techniques and Approaches:**

1. Multi-step Conversations:

Enable the chatbot to handle multi-turn conversations for complex queries.

- Store context and user preferences to provide relevant responses.

2. Sentiment Analysis:

- Implement sentiment analysis to understand customer satisfaction and detect issues requiring human intervention.

3. Dynamic FAQ Generation:

- Develop a system to dynamically generate FAQs based on user interactions to continuously improve the knowledge base.

4. A/B Testing:

- Continuously improve the chatbot's responses by A/B testing different conversation flows and NLP models.

5. Integration with E-commerce Data:

- Connect the chatbot to e-commerce data sources to provide real-time product information and inventory status.

**Interaction with Users:**

- Users access the web application and engage with the chatbot via a chat interface.

- The chatbot provides responses, information, and suggests relevant FAQ articles.

- Users can switch to human customer support if the chatbot can't resolve their issues.

- The chatbot uses NLP to understand and respond to user queries naturally.

The development process outlined above should lead to an effective chatbot-integrated web application that significantly improves customer support and engagement for the e-commerce company while utilizing innovative NLP techniques and approaches.

**A chatbot interacts with users and a web application as follows:**

Users engage with the chatbot through a chat interface, entering text or voice queries.The chatbot uses NLP algorithms to understand and interpret user input.

The chatbot identifies the user's intent and extracts relevant information from their query.Based on the intent and extracted data, the chatbot generates a response.Users receive the chatbot's response and can provide feedback or request further information.The chatbot communicates with the web application via APIs to access data or perform actions.The chatbot fetches information or processes user requests from the web application's database or services.The chatbot triggers actions within the web application, such as form submissions or data updates.The chatbot provides users with relevant information or updates from the web application.The chatbot can improve over time by learning from user interactions and web application data, enhancing its ability to assist users. Top of Form

**Literature and Survey papers**

|  |  |  |
| --- | --- | --- |
| SURVEY PAPER NAME | YEAR OF THE PAPER | ALGORITHM USED |
| Design and Development of CHATBOT | April 2021 | ML Algorihtm |
| An overview of chatbot technology | May 2020 | NLP(Natural language processing )  NLU() |
| A Survey on Evaluation Methods for Chatbots | March 2019 | ML & NLP(Natural language processing ) |
| A Survey on Chatbot Implementation in Customer Service Industry through  Deep Neural Networks | October 2018 | Template-based model |

# Importing Dependencies

import pandas as pd  
import numpy as np  
import string  
from string import digits  
import matplotlib.pyplot as plt  
import seaborn as sns  
import re  
from sklearn.model\_selection import train\_test\_split  
import tensorflow as tf  
from keras.layers import Input, LSTM, Embedding, Dense, Bidirectional, Concatenate, Dot, Activation, TimeDistributed  
from keras.models import Model  
from keras.utils import plot\_model

data\_path = "/kaggle/input/simple-dialogs-for-chatbot/dialogs.txt"

with open(data\_path, 'r', encoding='utf-8') as f:  
 lines = f.read().split('\n')  
inputs = []  
targets = []  
num\_samples = 10000 # Number of samples to train on.  
for line in lines[: min(num\_samples, len(lines) - 1)]:  
  
 input, target = line.split('\t')  
 inputs.append(input)  
 targets.append(target)

lines = pd.DataFrame({'input':inputs, 'target':targets})

lines.shape

(3724, 2)

lines.head()

input \  
0 hi, how are you doing?   
1 i'm fine. how about yourself?   
2 i'm pretty good. thanks for asking.   
3 no problem. so how have you been?   
4 i've been great. what about you?   
  
 target   
0 i'm fine. how about yourself?   
1 i'm pretty good. thanks for asking.   
2 no problem. so how have you been?   
3 i've been great. what about you?   
4 i've been good. i'm in school right now.

# Text Cleaning

def cleanup(lines):  
 # Since we work on word level, if we normalize the text to lower case, this will reduce the vocabulary. It's easy to recover the case later.  
 lines.input=lines.input.apply(lambda x: x.lower())  
 lines.target=lines.target.apply(lambda x: x.lower())  
  
 # To help the model capture the word separations, mark the comma with special token:  
 lines.input=lines.input.apply(lambda x: re.sub("'", '', x)).apply(lambda x: re.sub(",", ' COMMA', x))  
 lines.target=lines.target.apply(lambda x: re.sub("'", '', x)).apply(lambda x: re.sub(",", ' COMMA', x))  
  
 # Clean up punctuations and digits. Such special chars are common to both domains, and can just be copied with no error.  
 exclude = set(string.punctuation)  
 lines.input=lines.input.apply(lambda x: ''.join(ch for ch in x if ch not in exclude))  
 lines.target=lines.target.apply(lambda x: ''.join(ch for ch in x if ch not in exclude))  
  
 remove\_digits = str.maketrans('', '', digits)  
 lines.input=lines.input.apply(lambda x: x.translate(remove\_digits))  
 lines.target=lines.target.apply(lambda x: x.translate(remove\_digits))

st\_tok = 'START\_'  
end\_tok = '\_END'  
def data\_prep(lines):  
 cleanup(lines)  
 lines.target = lines.target.apply(lambda x : st\_tok + ' ' + x + ' ' + end\_tok)

data\_prep(lines)

lines.head()

input \  
0 hi COMMA how are you doing   
1 im fine how about yourself   
2 im pretty good thanks for asking   
3 no problem so how have you been   
4 ive been great what about you   
  
 target   
0 START\_ im fine how about yourself \_END   
1 START\_ im pretty good thanks for asking \_END   
2 START\_ no problem so how have you been \_END   
3 START\_ ive been great what about you \_END   
4 START\_ ive been good im in school right now \_END

def tok\_split\_word2word(data):  
 return data.split()

# Tokenization

tok\_split\_fn = tok\_split\_word2word

def data\_stats(lines, input\_tok\_split\_fn, target\_tok\_split\_fn):  
 input\_tokens=set()  
 for line in lines.input:  
 for tok in input\_tok\_split\_fn(line):  
 if tok not in input\_tokens:  
 input\_tokens.add(tok)  
  
 target\_tokens=set()  
 for line in lines.target:  
 for tok in target\_tok\_split\_fn(line):  
 if tok not in target\_tokens:  
 target\_tokens.add(tok)  
 input\_tokens = sorted(list(input\_tokens))  
 target\_tokens = sorted(list(target\_tokens))  
  
 num\_encoder\_tokens = len(input\_tokens)  
 num\_decoder\_tokens = len(target\_tokens)  
 max\_encoder\_seq\_length = np.max([len(input\_tok\_split\_fn(l)) for l in lines.input])  
 max\_decoder\_seq\_length = np.max([len(target\_tok\_split\_fn(l)) for l in lines.target])  
  
 return input\_tokens, target\_tokens, num\_encoder\_tokens, num\_decoder\_tokens, max\_encoder\_seq\_length, max\_decoder\_seq\_length

input\_tokens, target\_tokens, num\_encoder\_tokens, num\_decoder\_tokens, max\_encoder\_seq\_length, max\_decoder\_seq\_length = data\_stats(lines, input\_tok\_split\_fn=tok\_split\_fn, target\_tok\_split\_fn=tok\_split\_fn)  
print('Number of samples:', len(lines))  
print('Number of unique input tokens:', num\_encoder\_tokens)  
print('Number of unique output tokens:', num\_decoder\_tokens)  
print('Max sequence length for inputs:', max\_encoder\_seq\_length)  
print('Max sequence length for outputs:', max\_decoder\_seq\_length)

Number of samples: 3724  
Number of unique input tokens: 2338  
Number of unique output tokens: 2400  
Max sequence length for inputs: 20  
Max sequence length for outputs: 22

pad\_tok = 'PAD'  
sep\_tok = ' '  
special\_tokens = [pad\_tok, sep\_tok, st\_tok, end\_tok]  
num\_encoder\_tokens += len(special\_tokens)  
num\_decoder\_tokens += len(special\_tokens)

def vocab(input\_tokens, target\_tokens):  
  
 input\_token\_index = {}  
 target\_token\_index = {}  
 for i,tok in enumerate(special\_tokens):  
 input\_token\_index[tok] = i  
 target\_token\_index[tok] = i  
  
 offset = len(special\_tokens)  
 for i, tok in enumerate(input\_tokens):  
 input\_token\_index[tok] = i+offset  
  
 for i, tok in enumerate(target\_tokens):  
 target\_token\_index[tok] = i+offset  
  
 # Reverse-lookup token index to decode sequences back to something readable.  
 reverse\_input\_tok\_index = dict(  
 (i, tok) for tok, i in input\_token\_index.items())  
 reverse\_target\_tok\_index = dict(  
 (i, tok) for tok, i in target\_token\_index.items())  
 return input\_token\_index, target\_token\_index, reverse\_input\_tok\_index, reverse\_target\_tok\_index

input\_token\_index, target\_token\_index, reverse\_input\_tok\_index, reverse\_target\_tok\_index = vocab(input\_tokens, target\_tokens)

# Build Models

# Build Encoder

max\_encoder\_seq\_length = 30  
max\_decoder\_seq\_length = 30

def init\_model\_inputs(lines, max\_encoder\_seq\_length, max\_decoder\_seq\_length, num\_decoder\_tokens):  
 encoder\_input\_data = np.zeros(  
 (len(lines.input), max\_encoder\_seq\_length),  
 dtype='float32')  
 decoder\_input\_data = np.zeros(  
 (len(lines.target), max\_decoder\_seq\_length),  
 dtype='float32')  
 decoder\_target\_data = np.zeros(  
 (len(lines.target), max\_decoder\_seq\_length, num\_decoder\_tokens),  
 dtype='float32')  
  
 return encoder\_input\_data, decoder\_input\_data, decoder\_target\_data

def vectorize(lines, max\_encoder\_seq\_length, max\_decoder\_seq\_length, num\_decoder\_tokens, input\_tok\_split\_fn, target\_tok\_split\_fn):  
 encoder\_input\_data, decoder\_input\_data, decoder\_target\_data = init\_model\_inputs(lines, max\_encoder\_seq\_length, max\_decoder\_seq\_length, num\_decoder\_tokens)  
 for i, (input\_text, target\_text) in enumerate(zip(lines.input, lines.target)):  
 for t, tok in enumerate(input\_tok\_split\_fn(input\_text)):  
 encoder\_input\_data[i, t] = input\_token\_index[tok]  
 encoder\_input\_data[i, t+1:] = input\_token\_index[pad\_tok]  
 for t, tok in enumerate(target\_tok\_split\_fn(target\_text)):  
 # decoder\_target\_data is ahead of decoder\_input\_data by one timestep  
 decoder\_input\_data[i, t] = target\_token\_index[tok]  
 if t > 0:  
 # decoder\_target\_data will be ahead by one timestep  
 # and will not include the start character.  
 decoder\_target\_data[i, t - 1, target\_token\_index[tok]] = 1.  
 decoder\_input\_data[i, t+1:] = target\_token\_index[pad\_tok]  
 decoder\_target\_data[i, t:, target\_token\_index[pad\_tok]] = 1.  
  
 return encoder\_input\_data, decoder\_input\_data, decoder\_target\_data

encoder\_input\_data, decoder\_input\_data, decoder\_target\_data = vectorize(lines, max\_encoder\_seq\_length, max\_decoder\_seq\_length, num\_decoder\_tokens, input\_tok\_split\_fn=tok\_split\_fn, target\_tok\_split\_fn=tok\_split\_fn)

Build Encoder#Build Decoder

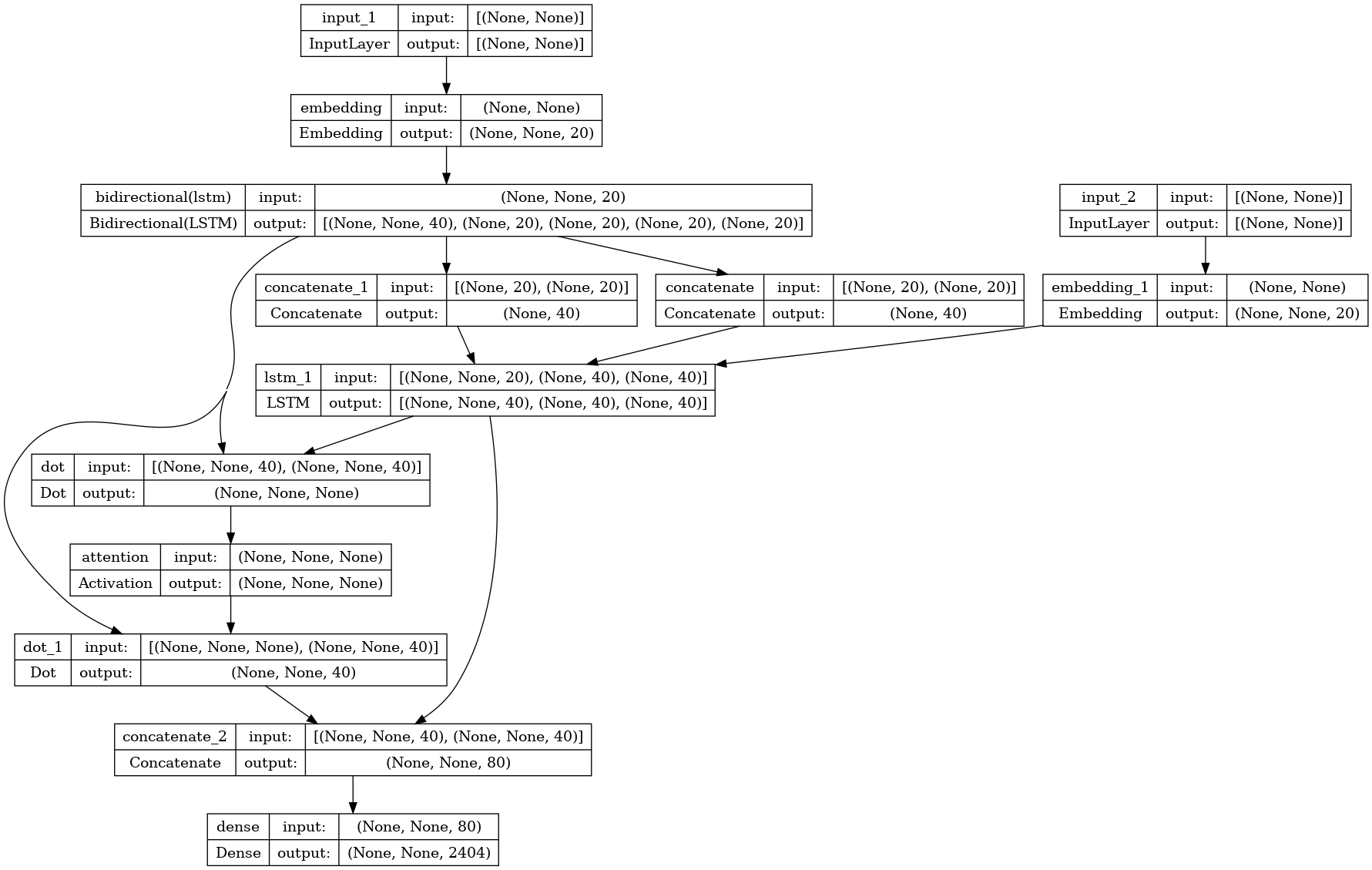
def seq2seq\_attention(num\_encoder\_tokens, num\_decoder\_tokens, emb\_sz, latent\_dim):  
 # Define an input sequence and process it.  
 encoder\_inputs = Input(shape=(None,), dtype='float32')  
 encoder\_inputs\_ = Embedding(num\_encoder\_tokens, emb\_sz, mask\_zero=True)(encoder\_inputs)   
   
 encoder = Bidirectional(LSTM(latent\_dim, return\_state=True, return\_sequences=True)) # Bi LSTM  
 encoder\_outputs, state\_f\_h, state\_f\_c, state\_b\_h, state\_b\_c = encoder(encoder\_inputs\_)# Bi LSTM  
 state\_h = Concatenate()([state\_f\_h, state\_b\_h])# Bi LSTM  
 state\_c = Concatenate()([state\_f\_c, state\_b\_c])# Bi LSTM  
  
 # We discard `encoder\_outputs` and only keep the states.  
 encoder\_states = [state\_h, state\_c]# Bi GRU, LSTM, BHi LSTM  
 print(encoder\_states)  
   
 decoder\_inputs = Input(shape=(None,))  
 decoder\_inputs\_ = Embedding(num\_decoder\_tokens, emb\_sz, mask\_zero=True)(decoder\_inputs)   
 # We set up our decoder to return full output sequences,  
 # and to return internal states as well. We don't use the  
 # return states in the training model, but we will use them in inference.  
 decoder\_lstm = LSTM(latent\_dim\*2, return\_sequences=True, return\_state=True)# Bi LSTM  
   
 decoder\_outputs, \_, \_ = decoder\_lstm(decoder\_inputs\_, initial\_state=encoder\_states)  
  
 # Equation (7) with 'dot' score from Section 3.1 in the paper.  
 # Note that we reuse Softmax-activation layer instead of writing tensor calculation  
 print(decoder\_outputs)  
 print(encoder\_outputs)  
 att\_dot = Dot(axes=[2, 2])  
 attention = att\_dot([decoder\_outputs, encoder\_outputs])  
 att\_activation = Activation('softmax', name='attention')  
 attention = att\_activation(attention)  
 print('attention', attention)  
 context\_dot = Dot(axes=[2,1])  
 context = context\_dot([attention, encoder\_outputs])  
 att\_context\_concat = Concatenate()  
 decoder\_combined\_context = att\_context\_concat([context, decoder\_outputs])  
  
 # Has another weight + tanh layer as described in equation (5) of the paper  
  
 decoder\_dense = Dense(num\_decoder\_tokens, activation='softmax')  
 #decoder\_outputs = decoder\_dense(decoder\_outputs)  
 decoder\_outputs = decoder\_dense(decoder\_combined\_context)  
  
 # Define the model that will turn  
 # `encoder\_input\_data` & `decoder\_input\_data` into `decoder\_target\_data`  
 model = Model([encoder\_inputs, decoder\_inputs], decoder\_outputs)  
  
 model.compile(optimizer=tf.keras.optimizers.Adam(lr = 0.00001), loss='categorical\_crossentropy', metrics=['acc'])  
  
 print('encoder-decoder model:')  
 print(model.summary())   
   
 print(encoder\_inputs)  
 print(encoder\_outputs)  
 print(encoder\_states)  
 encoder\_model = Model(encoder\_inputs, [encoder\_outputs] + encoder\_states)  
  
 decoder\_encoder\_inputs = Input(shape=(None, latent\_dim\*2,))  
 decoder\_state\_input\_h = Input(shape=(latent\_dim\*2,))# Bi LSTM  
 decoder\_state\_input\_c = Input(shape=(latent\_dim\*2,)) # Bi LSTM  
   
 decoder\_states\_inputs = [decoder\_state\_input\_h, decoder\_state\_input\_c]  
   
 decoder\_outputs, state\_h, state\_c = decoder\_lstm(decoder\_inputs\_, initial\_state=decoder\_states\_inputs)  
  
   
 decoder\_states = [state\_h, state\_c]  
   
 # Equation (7) with 'dot' score from Section 3.1 in the paper.  
 # Note that we reuse Softmax-activation layer instead of writing tensor calculation  
   
 attention = att\_dot([decoder\_outputs, decoder\_encoder\_inputs])  
   
 attention = att\_activation(attention)  
   
 context = context\_dot([attention, decoder\_encoder\_inputs])  
   
   
   
 decoder\_combined\_context = att\_context\_concat([context, decoder\_outputs])  
   
 # Has another weight + tanh layer as described in equation (5) of the paper  
   
 decoder\_outputs = decoder\_dense(decoder\_combined\_context)  
   
 decoder\_model = Model(  
 [decoder\_inputs, decoder\_encoder\_inputs] + decoder\_states\_inputs,  
 [decoder\_outputs, attention] + decoder\_states)  
   
 return model, encoder\_model, decoder\_model

# Create Inference Model

emb\_sz = 20

model, encoder\_model, decoder\_model = seq2seq\_attention(num\_encoder\_tokens, num\_decoder\_tokens, emb\_sz=emb\_sz, latent\_dim=emb\_sz)  
print(model.summary())  
plot\_model(model, show\_shapes=True, show\_layer\_names=True)

[<KerasTensor: shape=(None, 40) dtype=float32 (created by layer 'concatenate')>, <KerasTensor: shape=(None, 40) dtype=float32 (created by layer 'concatenate\_1')>]  
KerasTensor(type\_spec=TensorSpec(shape=(None, None, 40), dtype=tf.float32, name=None), name='lstm\_1/PartitionedCall:1', description="created by layer 'lstm\_1'")  
KerasTensor(type\_spec=TensorSpec(shape=(None, None, 40), dtype=tf.float32, name=None), name='bidirectional/concat:0', description="created by layer 'bidirectional'")  
attention KerasTensor(type\_spec=TensorSpec(shape=(None, None, None), dtype=tf.float32, name=None), name='attention/Softmax:0', description="created by layer 'attention'")  
encoder-decoder model:  
Model: "model"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param # Connected to   
==================================================================================================  
 input\_1 (InputLayer) [(None, None)] 0 []   
   
 embedding (Embedding) (None, None, 20) 46840 ['input\_1[0][0]']   
   
 input\_2 (InputLayer) [(None, None)] 0 []   
   
 bidirectional (Bidirectional) [(None, None, 40), 6560 ['embedding[0][0]']   
 (None, 20),   
 (None, 20),   
 (None, 20),   
 (None, 20)]   
   
 embedding\_1 (Embedding) (None, None, 20) 48080 ['input\_2[0][0]']   
   
 concatenate (Concatenate) (None, 40) 0 ['bidirectional[0][1]',   
 'bidirectional[0][3]']   
   
 concatenate\_1 (Concatenate) (None, 40) 0 ['bidirectional[0][2]',   
 'bidirectional[0][4]']   
   
 lstm\_1 (LSTM) [(None, None, 40), 9760 ['embedding\_1[0][0]',   
 (None, 40), 'concatenate[0][0]',   
 (None, 40)] 'concatenate\_1[0][0]']   
   
 dot (Dot) (None, None, None) 0 ['lstm\_1[0][0]',   
 'bidirectional[0][0]']   
   
 attention (Activation) (None, None, None) 0 ['dot[0][0]']   
   
 dot\_1 (Dot) (None, None, 40) 0 ['attention[0][0]',   
 'bidirectional[0][0]']   
   
 concatenate\_2 (Concatenate) (None, None, 80) 0 ['dot\_1[0][0]',   
 'lstm\_1[0][0]']   
   
 dense (Dense) (None, None, 2404) 194724 ['concatenate\_2[0][0]']   
   
==================================================================================================  
Total params: 305,964  
Trainable params: 305,964  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
None  
KerasTensor(type\_spec=TensorSpec(shape=(None, None), dtype=tf.float32, name='input\_1'), name='input\_1', description="created by layer 'input\_1'")  
KerasTensor(type\_spec=TensorSpec(shape=(None, None, 40), dtype=tf.float32, name=None), name='bidirectional/concat:0', description="created by layer 'bidirectional'")  
[<KerasTensor: shape=(None, 40) dtype=float32 (created by layer 'concatenate')>, <KerasTensor: shape=(None, 40) dtype=float32 (created by layer 'concatenate\_1')>]  
Model: "model"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param # Connected to   
==================================================================================================  
 input\_1 (InputLayer) [(None, None)] 0 []   
   
 embedding (Embedding) (None, None, 20) 46840 ['input\_1[0][0]']   
   
 input\_2 (InputLayer) [(None, None)] 0 []   
   
 bidirectional (Bidirectional) [(None, None, 40), 6560 ['embedding[0][0]']   
 (None, 20),   
 (None, 20),   
 (None, 20),   
 (None, 20)]   
   
 embedding\_1 (Embedding) (None, None, 20) 48080 ['input\_2[0][0]']   
   
 concatenate (Concatenate) (None, 40) 0 ['bidirectional[0][1]',   
 'bidirectional[0][3]']   
   
 concatenate\_1 (Concatenate) (None, 40) 0 ['bidirectional[0][2]',   
 'bidirectional[0][4]']   
   
 lstm\_1 (LSTM) [(None, None, 40), 9760 ['embedding\_1[0][0]',   
 (None, 40), 'concatenate[0][0]',   
 (None, 40)] 'concatenate\_1[0][0]']   
   
 dot (Dot) (None, None, None) 0 ['lstm\_1[0][0]',   
 'bidirectional[0][0]']   
   
 attention (Activation) (None, None, None) 0 ['dot[0][0]']   
   
 dot\_1 (Dot) (None, None, 40) 0 ['attention[0][0]',   
 'bidirectional[0][0]']   
   
 concatenate\_2 (Concatenate) (None, None, 80) 0 ['dot\_1[0][0]',   
 'lstm\_1[0][0]']   
   
 dense (Dense) (None, None, 2404) 194724 ['concatenate\_2[0][0]']   
   
==================================================================================================  
Total params: 305,964  
Trainable params: 305,964  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
None



# Train Model

model.fit([encoder\_input\_data, decoder\_input\_data], decoder\_target\_data,  
 batch\_size=32,  
 epochs=100)

Epoch 1/100  
117/117 [==============================] - 24s 81ms/step - loss: 6.3425 - acc: 0.1152  
Epoch 2/100  
117/117 [==============================] - 9s 79ms/step - loss: 5.2292 - acc: 0.1294  
Epoch 3/100  
117/117 [==============================] - 10s 82ms/step - loss: 4.9883 - acc: 0.1716  
Epoch 4/100  
117/117 [==============================] - 10s 83ms/step - loss: 4.8115 - acc: 0.2206  
Epoch 5/100  
117/117 [==============================] - 10s 82ms/step - loss: 4.6970 - acc: 0.2425  
Epoch 6/100  
117/117 [==============================] - 10s 83ms/step - loss: 4.6151 - acc: 0.2520  
Epoch 7/100  
 44/117 [==========>...................] - ETA: 5s - loss: 4.5502 - acc: 0.2604

def decode\_sequence\_attention(input\_seq, sep=' '):  
 # Encode the input as state vectors.  
 encoder\_outputs, h, c = encoder\_model.predict(input\_seq)  
 states\_value = [h,c]  
 # Generate empty target sequence of length 1.  
 target\_seq = np.zeros((1,1))  
 # Populate the first character of target sequence with the start character.  
 target\_seq[0, 0] = target\_token\_index[st\_tok]  
  
 # Sampling loop for a batch of sequences  
 # (to simplify, here we assume a batch of size 1).  
 stop\_condition = False  
 decoded\_sentence = ''  
 attention\_density = []  
 while not stop\_condition:  
 output\_tokens, attention, h, c = decoder\_model.predict(  
 [target\_seq, encoder\_outputs] + states\_value)  
 attention\_density.append(attention[0][0])# attention is max\_sent\_len x 1 since we have num\_time\_steps = 1 for the output  
 # Sample a token  
 sampled\_token\_index = np.argmax(output\_tokens[0, -1, :])  
 sampled\_tok = reverse\_target\_tok\_index[sampled\_token\_index]  
 decoded\_sentence += sep + sampled\_tok  
  
 # Exit condition: either hit max length  
 # or find stop character.  
 if (sampled\_tok == end\_tok or  
 len(decoded\_sentence) > 52):  
 stop\_condition = True  
  
 # Update the target sequence (of length 1).  
 target\_seq = np.zeros((1,1))  
 target\_seq[0, 0] = sampled\_token\_index  
  
 # Update states  
 states\_value = [h, c]  
 attention\_density = np.array(attention\_density)  
 return decoded\_sentence, attention\_density

# Visualize Metrics

# Save Model

lines.input[0]

word\_decoded\_sents = []  
for seq\_index in range(100): #[14077,20122,40035,40064, 40056, 40068, 40090, 40095, 40100, 40119, 40131, 40136, 40150, 40153]:  
 input\_seq = encoder\_input\_data[seq\_index: seq\_index + 1]  
 decoded\_sentence, attention = decode\_sequence\_attention(input\_seq)  
 print('-')  
 print('Input sentence:', lines.input[seq\_index: seq\_index + 1])  
 print('Decoded sentence:', decoded\_sentence)  
 word\_decoded\_sents.append(decoded\_sentence)

def calculate\_WER\_sent(gt, pred):  
 '''  
 calculate\_WER('calculating wer between two sentences', 'calculate wer between two sentences')  
 '''  
 gt\_words = gt.lower().split(' ')  
 pred\_words = pred.lower().split(' ')  
 d = np.zeros(((len(gt\_words) + 1), (len(pred\_words) + 1)), dtype=np.uint8)  
 # d = d.reshape((len(gt\_words)+1, len(pred\_words)+1))  
  
 # Initializing error matrix  
 for i in range(len(gt\_words) + 1):  
 for j in range(len(pred\_words) + 1):  
 if i == 0:  
 d[0][j] = j  
 elif j == 0:  
 d[i][0] = i  
  
 # computation  
 for i in range(1, len(gt\_words) + 1):  
 for j in range(1, len(pred\_words) + 1):  
 if gt\_words[i - 1] == pred\_words[j - 1]:  
 d[i][j] = d[i - 1][j - 1]  
 else:  
 substitution = d[i - 1][j - 1] + 1  
 insertion = d[i][j - 1] + 1  
 deletion = d[i - 1][j] + 1  
 d[i][j] = min(substitution, insertion, deletion)  
 return d[len(gt\_words)][len(pred\_words)]  
def calculate\_WER(gt, pred):  
 '''  
  
 :param gt: list of sentences of the ground truth  
 :param pred: list of sentences of the predictions  
 both lists must have the same length  
 :return: accumulated WER  
 '''  
# assert len(gt) == len(pred)  
 WER = 0  
 nb\_w = 0  
 for i in range(len(gt)):  
 #print(gt[i])  
 #print(pred[i])  
 WER += calculate\_WER\_sent(gt[i], pred[i])  
 nb\_w += len(gt[i])  
  
 return WER / nb\_w

target\_sents = list(lines.target[:100])  
target\_sents = [x[1:-1] for x in target\_sents]  
word\_decoded\_sents = [' '.join(x.split()[1:-1]) for x in word\_decoded\_sents]

WER\_word = calculate\_WER(target\_sents, word\_decoded\_sents)  
print('Word level NMT WER = ', str(WER\_word))

**Conclusion:**

Creating a chatbot in Python is an exciting journey that involves various components and requires an understanding of NLP, data management, and user experience design. This documentation serves as a valuable resource for both building a chatbot from scratch and enhancing an existing one. By following the guidelines, developers can create intelligent and engaging chatbots to meet various user needs.