CREATE A CHATBOT IN PYTHON

Presented by:-

Ahila. K

Bhuvaneshwari. S

Gopika. K

Harshini. A

Kavya. K

Phase 5 : Project Documentation & Submission

Topic : Document the Project and Prepare it for Submission

Problem statement

Develop a Python-based chatbot that can engage in natural language conversations with users, answer questions on a specific topic, and provide relevant information. The chatbot should be capable of understanding and generating text in a conversational manner, and it should integrate with a messaging platform or a user interface for user interaction. The goal is to create an intelligent and user-friendly chatbot that can assist users with their inquiries and tasks efficiently.

Objectives

Chatbot is a Python library that is developed to provide automated responses to user inputs. It makes utilization of a combination of Machine Learning algorithms in order to generate multiple types of responses.

Data sources

With a data source, you can connect your Interfaces AI Chatbot to your own knowledge sources and tailor the responses for your business or project. You can restrict your bot from using its training information to provide answers and set custom responses when information doesn't exist.

Tasks

- ✓ Chat bot questions and answers preparing manually.
- ✓ Storing questions and answers in database.

✓ Connecting front page with questions and answers using python code.

Design thinking

Design thinking is a human-centered method that aims to understand the user's problems and generate ideas to solve them. It can be used for digital products and services, but also chatbots. Follow our lesson, learn the fundamentals of design thinking, and find out how to apply it to build user-friendly chatbots.

1.Environment Setup:

- Install Python: Make sure you have Python installed on your system.
- Choose an IDE or text editor for coding (e.g., VSCode, PyCharm, Jupyter Notebook).

2. Select a Chatbot Framework:
   - Choose a Python chatbot framework/library to work with, like ChatterBot, NLTK, or Rasa.
3. Data Collection:

- Gather or create a dataset of conversation examples to train your chatbot. This data is crucial for teaching the chatbot how to respond.

4. Data Preprocessing:
    - Clean and preprocess the conversation data, including text normalization, tokenization, and stemming.
5. Training:
    - Use your chosen framework to train the chatbot on the preprocessed data. This involves teaching the chatbot how to understand and respond to user queries.
6. Integration with Natural Language Processing (NLP):
    - Implement NLP techniques to improve the chatbot's ability to understand and generate human-like responses. Libraries like spaCy or NLTK can be helpful here.
7. Create User Interfaces:
    - Develop a user interface for your chatbot. This can be a web app, a command-line interface, or an integration with a messaging platform.
8. Testing and Debugging:
    - Test your chatbot thoroughly to identify and fix issues. Pay attention to both the chatbot's ability to understand input and generate coherent responses.

9. Deployment:
    - Deploy your chatbot on a server or platform of your choice so that users can interact with it.
10. Continuous Improvement:
    - Continuously collect user feedback and improve your chatbot's responses. You can also consider adding more features and capabilities over time.

PHASES OF DEVELOPMENT

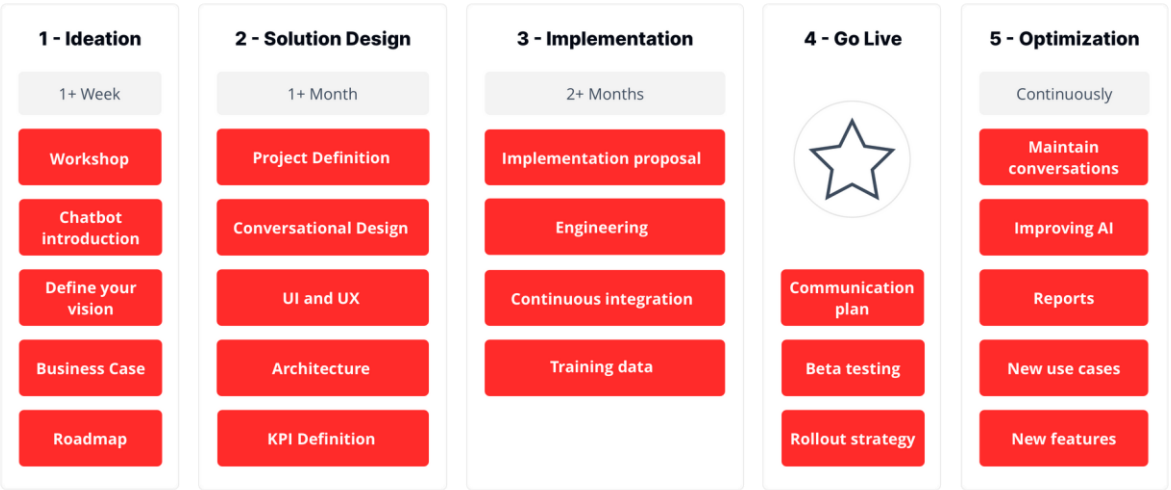
Step 1: Make a list of ideas and define business cases.

Step 2: Bring all the experts together and define the solution end to end.

Step 3: Build your ideas into the chatbot.

Step 4: From training to go-live.

Step 5: Optimization.



LIBRARIES USED IN CHATBOT

Creating a chatbot in Python often involves using various libraries and frameworks to facilitate natural language processing, web interactions, and other essential functionalities. Here are some commonly used libraries and frameworks for building chatbots in Python:

1.Natural Language Processing (NLP) Libraries:

- NLTK (Natural Language Toolkit): NLTK is a popular library for NLP tasks, including tokenization, stemming, lemmatization, and sentiment analysis.
- spaCy: spaCy is a fast and efficient NLP library for various NLP tasks, such as part-of-speech tagging, entity recognition, and dependency parsing.
- TextBlob: TextBlob is a simple library for processing textual data, including sentiment analysis, translation, and text classification.

2.Machine Learning Libraries:

Scikit-Learn: If your chatbot requires machine learning, Scikit-Learn is a powerful library for tasks like text classification and clustering.

-

- TensorFlow and Keras: These libraries are suitable for creating and training deep learning models for more complex chatbots.

3.Chatbot Frameworks:

- Rasa: Rasa is an open-source framework specifically designed for building conversational AI chatbots. It provides tools for dialogue management, intent recognition, and entity extraction.
- BotPress: BotPress is another open-source chatbot framework that offers a visual interface for building and managing chatbots.
- Microsoft Bot Framework: If you want to develop chatbots that work across multiple platforms (e.g.,

  Microsoft Teams, Skype, Slack), the Microsoft Bot Framework is a valuable option.

4.Web Frameworks:

- Flask and Django: These web frameworks are commonly used for building web-based chatbots that interact with users via a web interface.

5.Database Libraries:

- SQLAlchemy and Django ORM: These libraries help in connecting your chatbot to databases to store and retrieve information.

6.API Libraries:

- Requests: The Requests library is used to make HTTP requests to external APIs, which can be valuable for integrating data and services into your chatbot.

7.Front-End Libraries:

- JavaScript libraries like React or Vue.js can be used if you're building a web-based chatbot with a front-end

interface.

8.Deployment and Hosting:

- Docker: Docker can be used to containerize your chatbot application for easy deployment.
- Cloud Platforms (e.g., AWS, Azure, Google Cloud): These platforms provide hosting and serverless options for deploying your chatbot.

9.Version Control and Collaboration:

- Git and platforms like GitHub or GitLab are useful for version control and collaboration when developing chatbots with a team.

10.Other Specialized Libraries:

-

OpenAI's GPT-3 (or its successors): These libraries can be used to integrate powerful natural language generation capabilities into your chatbot.

- Speech recognition libraries (e.g., SpeechRecognition) if your chatbot needs to handle voice interactions.

The choice of libraries and frameworks depends on your specific chatbot requirements and the technologies you are comfortable with. When developing a chatbot, it's crucial to select the tools that best match the objectives and capabilities you want to implement.

INTEGRATION OF NLP TECHNIQUES

Integrating Natural Language Processing (NLP) techniques into a chatbot in Python is a fundamental aspect of building a chatbot that can understand and generate human-like text responses.

Here are the key steps for integrating NLP techniques into a chatbot,

1. Text Preprocessing:
    - Tokenization: Break the user's input and responses into individual words or tokens.

- Lowercasing: Convert all text to lowercase to ensure consistency.
- Stopword Removal: Remove common words like "a," "an," "the" to reduce noise.
- Lemmatization or Stemming: Reduce words to their base form to improve matching.

2.Intent Recognition:

- Use techniques like rule-based approaches, machine learning, or deep learning to determine the user's intent. This involves understanding what the user wants from their input.

3.Entity Recognition:

- Identify specific pieces of information (entities) in the user's input, such as dates, names, or locations. This is important for extracting actionable information from the user's request.

4.Response Generation:

- Given the recognized intent and entities, generate a response that is contextually appropriate and relevant.

You can use templates, rule-based responses, or more advanced techniques like generative models (e.g., GPT-3).

5.Dialog Management:

Keep track of the conversation's context and manage the flow of conversation. Ensure that the chatbot remembers the user's previous inputs and responses for context-aware interactions.

6.Sentiment Analysis:

- Analyze the sentiment of user input to understand the user's emotional state. This can be used to tailor responses accordingly.

7.Named Entity Recognition (NER):

- Identify and extract named entities, such as names of people, organizations, and locations, which can be crucial for understanding and responding to user queries accurately.

8.Word Embeddings:

- Utilize word embeddings like Word2Vec, FastText, or pre-trained embeddings (e.g., GloVe) to represent words in a dense vector space, enabling better understanding of word relationships.

9.Language Models:

- Employ pre-trained language models like BERT, GPT-2, or GPT-3 to improve the chatbot's language understanding and generation capabilities.

10.Conversational Memory:

- Maintain a memory of the conversation history to allow the chatbot to maintain context and provide more coherent responses

11.Error Handling:

- Implement robust error handling to manage situations where the chatbot does not understand the user's input.

12. Testing and Fine-Tuning:
    - Continuously test and fine-tune the chatbot's NLP components using real user interactions to improve its performance and understanding over time.

13. Integration with APIs and Databases:
    - Integrate the chatbot with external data sources, APIs, and databases to fetch information or perform specific tasks.

14. User Experience (UX) Design:
    - Consider the user experience and design a conversational flow that guides users effectively and ensures a natural interaction.

15. Deployment:
    - Deploy the chatbot using a suitable web framework (e.g., Flask, Django) and host it on a web server or a cloud platform.

When integrating NLP techniques into a chatbot, you can use libraries like NLTK, spaCy, TextBlob, and machine learning frameworks (e.g., Scikit-Learn, TensorFlow) for the various NLP tasks mentioned above.

HOW THE CHATBOT INTERACTS WITH USERS AND THE WEB APPLICATION

A chatbot interacts with users and a web application through a combination of frontend and backend components, communication protocols, and data exchange.

Here's an overview of how this interaction typically works,

1. User Interface (Frontend):
   - Users interact with the chatbot through a user interface on a web application. This can be a chat window embedded in a website, a messaging platform, or a dedicated chatbot app.
2. User Input:
   - Users enter text or voice input in the chat interface, which serves as their communication with the chatbot.
3. Web Application (Frontend):
   - The user input is typically collected and sent to the web application's frontend.

4. User Input Processing (Frontend):
   - The frontend may perform basic preprocessing on the user input, such as removing extra whitespace or formatting.
5. Communication with Backend:
   - The frontend communicates with the backend of the web application, where the chatbot logic resides. This communication can happen via HTTP requests or WebSocket connections, depending on the architecture.
6. Backend (Chatbot Logic):
   - The backend is responsible for processing user input and generating responses. It contains the chatbot's core logic, which includes:
   - Natural Language Processing (NLP): Understanding the user's intent and extracting entities from the input.
   - Dialog Management: Maintaining the conversation context, tracking previous messages, and managing the flow of the conversation.
   - Response Generation: Generating text or voice responses that are contextually relevant to the user's input.
   - Interaction with External Services: If the chatbot needs to access external databases, APIs, or services, this is where the interaction happens.
7. Backend Response:
   - The chatbot's backend generates a response based on the user's input and context.
8. Communication with Frontend:

- The backend sends the response back to the frontend using an appropriate protocol (e.g., HTTP or WebSocket).

9. Frontend Rendering:
   - The frontend receives the response and renders it in the chat interface, displaying the chatbot's reply to the user.
10. User Interaction Continues: -
    - The user can continue the conversation by providing further input, and the process repeats.
11. Context Maintenance: -
    - The backend is responsible for maintaining the conversation context, allowing the chatbot to remember and refer to prior messages in a conversation, ensuring a coherent interaction.
12. Data Storage (if necessary): -
    - Depending on the chatbot's design, the backend may store conversation history and user data in a database for later reference or analysis.
13. Error Handling: -
    - The chatbot should have error-handling mechanisms to deal with cases where it cannot understand the user's input or when errors occur in external service interactions.

This interaction cycle continues as long as the user engages with the chatbot. The chatbot's ability to understand and respond to user input effectively, along with its ability to maintain context, is crucial for providing a seamless and natural user experience.

The specific implementation details, including the choice of programming languages, libraries, and frameworks, may vary based on the design and requirements of the web application and chatbot.

INNOVATIVE TECHNIQUES USED DURING THE DEVELOPMENT

During the development of chatbots in Python, several innovative techniques and approaches can be applied to enhance the chatbot's functionality, interactivity, and user experience.

Here are some innovative techniques and approaches that can be incorporated into chatbot development,

1.Conversational AI and Pre-trained Models:

- Leveraging state-of-the-art pre-trained language models like GPT-3, BERT, or T5 for more natural language understanding and generation.

2.Generative Chatbots:

- Developing chatbots that can generate creative and contextually relevant responses by using deep learning and generative models. These can create engaging and interactive conversations.

3.Multimodal Chatbots:

- Integrating both text and voice interactions, allowing users to communicate with the chatbot through both written messages and spoken language.

4.Personalization:

- Implementing personalization techniques to tailor responses and recommendations to individual users based on their past interactions and preferences.

5.Emotion Detection:

- Integrating sentiment analysis and emotion detection to recognize and respond to users' emotional states, providing empathetic and appropriate responses.

6.Contextual Memory:

- Enhancing the chatbot's ability to maintain context throughout a conversation, allowing it to recall previous messages and respond coherently.

7.Transfer Learning:

- Applying transfer learning techniques to adapt pre-trained models to specific chatbot tasks, reducing the need for extensive training data.

8.Behavior Analysis:

- Analyzing user behavior to gain insights into their preferences, patterns, and conversational history, enabling more personalized and effective interactions.

9.Interactive Learning:

- Implementing reinforcement learning and interactive learning techniques to improve the chatbot's performance over time through user feedback.

10.Dynamic Responses:

- Creating dynamic and context-aware responses by integrating data from real-time sources, such as live news feeds or weather updates.

11.Knowledge Graphs:

- Building and utilizing knowledge graphs to enhance the chatbot's understanding of complex topics and relationships between entities.

12.Conversational Flow Control:

- Implementing advanced conversational flow control techniques that guide users through complex interactions, such as multi-step tasks or decision trees.

13.Content Generation and Summarization:

- Integrating content generation and summarization techniques to provide users with concise information or to create detailed responses from extensive text.

14.Multi-Language Support:

- Supporting multiple languages and enabling translation capabilities, allowing users to interact with the chatbot in their preferred language.

15.Privacy and Security Measures:

- Implementing advanced privacy and security features to protect user data and ensure compliance with data protection regulations.

16.Voice Biometrics:

- Integrating voice biometric recognition for secure authentication and user identification in voice-enabled chatbots.

17.Augmented Reality (AR) Integration:

- Exploring AR integration for chatbots, allowing them to provide information or guidance in augmented reality environments.

18.Human-Agent Hybrid Chatbots:

- Combining human agents with chatbots in a seamless manner, where chatbots assist and collaborate with human agents to provide better customer support.

GIVEN DATASET

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

CHATBOT IMPLEMENTATION

- Preparing the Dependencies.

- The right dependencies need to be established before we can create a chatbot.
- Creating and Training the Chatbot.
- Once the dependence has been established, we can build and train our chatbot.

PROGRAM

In[1]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt import tensorflow as tf
import keras from keras.layers import Dense import json
import re import string from
sklearn.feature_extraction.text import TfidfVectorizer
import unicodedata

from sklearn.model_selection import train_test_split
```

In[2]:

```
question =[] answer = [] with open("../input/simple-dialogs-for-
chatbot/dialogs.txt",'r') as f :

for line in f :

line = line.split('\t') question.append(line[0])
answer.append(line[1])

print(len(question) == len(answer))
```

True

In[3]:

```
Question[:5]
```

Out[3]:

['hi, how are you doing?',

"i'm fine. how about yourself?",

"i'm pretty good. thanks for asking.",

'no problem. so how have you been?',

"i've been great. what about you?"]

In[4]:

```
Answer[:5]
```

Out[4]:

["i'm fine. how about yourself?\n",

"i'm pretty good. thanks for asking.\n",

'no problem. so how have you been?\n',

"i've been great. what about you?\n", "i've been good. i'm in school right now.\n"]

In[5]:

```
answer = [ i.replace("\n","") for i in answer]
```

In[6]:

```
Answer[:5]
```

Out[6]:

["i'm fine. how about yourself?",

"i'm pretty good. thanks for asking.",

'no problem. so how have you been?',

"i've been great. what about you?",

"i've been good. i'm in school right now."]

In[7]:

```
data = pd.DataFrame({"question" : question ,"answer":answer})
data.head()
```

	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.

In[8]:

```
def unicode_to_ascii(s):

return ''.join(c for c in unicodedata.normalize('NFD', s)
    if unicodedata.category(c) != 'Mn')
```

In[9]:

```
def clean_text(text):

    text =
unicode_to_ascii(text.lower().strip()) text
= re.sub(r"i'm", "i am", text) text =
re.sub(r"\r", "", text) text = re.sub(r"he's",
"he is", text) text = re.sub(r"she's", "she
is", text) text = re.sub(r"it's", "it is", text)
text = re.sub(r"that's", "that is", text) text =
re.sub(r"what's", "that is", text) text =
re.sub(r"where's", "where is", text) text =
re.sub(r"how's", "how is", text) text =
re.sub(r"\'ll", " will", text) text =
re.sub(r"\'ve", " have", text) text =
re.sub(r"\'re", " are", text) text =
re.sub(r"\'d", " would", text) text =
re.sub(r"\'re", " are", text) text =
re.sub(r"won't", "will not", text) text =
re.sub(r"can't", "cannot", text) text =
```

```
        re.sub(r"n't", " not", text) text =
          re.sub(r"n'", "ng", text) text =
          re.sub(r"'bout", "about", text)

      text = re.sub(r"'til", "until", text) text = re.sub(r"[-
            ()\"#/@;:<>{}`+=~|.!?,]", "", text) text =
   text.translate(str.maketrans('', '', string.punctuation)) text
   = re.sub("(\\W)"," ",text) text = re.sub('\S*\d\S*\s*','', text)

  text = "<sos> " + text + " <eos>" return
                        text
```

In[10]:

```
Data["question"][0]
```

Out[10]:

```
'hi, how are you doing?'
```

In[11]:

```
data["question"] = data.question.apply(clean_text)
```

In[12]:

```
Data["question"][0]
```

Out[12]:

```
'<sos> hi how are you doing <eos>'
```

In[13]:

```
data["answer"] = data.answer.apply(clean_text)
```

In[14]:

```
question = data.question.values.tolist() answer
         = data.answer.values.tolist()
```

In[15]:

```
def tokenize(lang):

lang_tokenizer = tf.keras.preprocessing.text.Tokenizer(
filters='')

lang_tokenizer.fit_on_texts(lang) tensor =
lang_tokenizer.texts_to_sequences(lang)

tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor,

padding='post')

return tensor, lang_tokenizer
```

In[16]:

```
input_tensor , inp_lang = tokenize(question)
```

In[17]:

```
target_tensor , targ_lang = tokenize(answer)
```

In[18]:

```
#len(inp_question) == len(inp_answer)
```

In[19]:

```
def remove_tags(sentence): return
sentence.split("<start>")[-1].split("<end>")[0]
```

In[20]:

```
max_length_targ, max_length_inp = target_tensor.shape[1],
input_tensor.shape[1]
```

In[21]:

```
# Creating training and validation sets using an 80-20 split
input_tensor_train, input_tensor_val, target_tensor_train, target_tensor_val =
train_test_split(input_tensor, target_tensor, test_size=0.2)
```

In[22]:

#print(len(train_inp) , len(val_inp) , len(train_target) , len(val_target))

In[23]:

BUFFER_SIZE = len(input_tensor_train)
BATCH_SIZE = 64

steps_per_epoch = len(input_tensor_train)//BATCH_SIZE
embedding_dim = 256 units = 1024 vocab_inp_size =
len(inp_lang.word_index)+1 vocab_tar_size =
len(targ_lang.word_index)+1

dataset = tf.data.Dataset.from_tensor_slices((input_tensor_train,
target_tensor_train)).shuffle(BUFFER_SIZE) dataset =
dataset.batch(BATCH_SIZE, drop_remainder=True)
example_input_batch, example_target_batch =
next(iter(dataset)) example_input_batch.shape,
example_target_batch.shape

2022-10-20 06:33:56.495284:

I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero

2022-10-20 06:33:56.619975: I

tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero

2022-10-20 06:33:56.620805: I

tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero

2022-10-20 06:33:56.624402: I
tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX2 AVX512F

FMA

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2022-10-20 06:33:56.624816: I

tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero

2022-10-20 06:33:56.625829: I

tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero

2022-10-20 06:33:56.626693: I

tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero

2022-10-20 06:33:59.460823: I

tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero

2022-10-20 06:33:59.461762: I

tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero

2022-10-20 06:33:59.462456: I

tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero

2022-10-20 06:33:59.463056: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 15401 MB memory: ->

device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:04.0, compute capability: 6.0

Out[23]:

(TensorShape([64, 22]), TensorShape([64, 22]))

In[24]:

```
class Encoder(tf.keras.Model):

def _init_(self, vocab_size, embedding_dim, enc_units, batch_sz):

super(Encoder, self)._init_() self.batch_sz = batch_sz
self.enc_units = enc_units self.embedding =
tf.keras.layers.Embedding(vocab_size,

embedding_dim) self.gru =
tf.keras.layers.GRU(self.enc_units,

return_sequences=True,
return_state=True,
recurrent_initializer='glorot_uniform')

def call(self, x,hidden): x = self.embedding(x)
output, state = self.gru(x, initial_state =
hidden) return output, state

def initialize_hidden_state(self): return
tf.zeros((self.batch_sz, self.enc_units))
```

In[25]:

```
encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE)

# sample input sample_hidden =
encoder.initialize_hidden_state() sample_output, sample_hidden
= encoder(example_input_batch, sample_hidden) print ('Encoder
output shape: (batch size, sequence length, units)

{}'.format(sample_output.shape)) print ('Encoder
Hidden state shape: (batch size, units)
{}'.format(sample_hidden.shape))
```

2022-10-20 06:34:00.854919: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005

Encoder output shape: (batch size, sequence length, units) (64, 22, 1024)

Encoder Hidden state shape: (batch size, units) (64, 1024)

In[26]:

```
class BahdanauAttention(tf.keras.layers.Layer):

    def _init_(self, units):

super(BahdanauAttention, self)._init_()
 self.W1 = tf.keras.layers.Dense(units)
 self.W2 = tf.keras.layers.Dense(units)
  self.V = tf.keras.layers.Dense(1)

    def call(self, query, values):

        # query hidden state shape == (batch_size, hidden size)

        # query_with_time_axis shape == (batch_size, 1, hidden size)

        # values shape == (batch_size, max_len, hidden size)

    # we are doing this to broadcast addition along the time axis to calculate the score query_with_time_axis = tf.expand_dims(query, 1)

        # score shape == (batch_size, max_length, 1)

    # we get 1 at the last axis because we are applying score to self.V

    # the shape of the tensor before applying self.V is (batch_size, max_length, units) score = self.V(tf.nn.tanh(
 self.W1(query_with_time_axis) + self.W2(values)))

        # attention_weights shape == (batch_size, max_length, 1)
  attention_weights = tf.nn.softmax(score, axis=1)

    # context_vector shape after sum == (batch_size,

    hidden_size) context_vector = attention_weights * values
```

```
context_vector = tf.reduce_sum(context_vector, axis=1) return
context_vector, attention_weights
```

In[27]:

```
attention_layer = BahdanauAttention(10) attention_result,
attention_weights = attention_layer(sample_hidden, sample_output)

print("Attention result shape: (batch size, units)

{}".format(attention_result.shape)) print("Attention weights
shape: (batch_size, sequence_length, 1)

{}".format(attention_weights.shape))
```

```
Attention result shape: (batch size, units) (64, 1024)

Attention weights shape: (batch_size, sequence_length, 1) (64, 22, 1)
```

In[28]:

```
class Decoder(tf.keras.Model):

def _init_(self, vocab_size, embedding_dim, dec_units, batch_sz):

super(Decoder, self)._init_() self.batch_sz = batch_sz
self.dec_units = dec_units self.embedding =
tf.keras.layers.Embedding(vocab_size,

embedding_dim) self.gru =
tf.keras.layers.GRU(self.dec_units,

return_sequences=True,
return_state=True,
recurrent_initializer='glorot_uniform')

self.fc = tf.keras.layers.Dense(vocab_size)

# used for attention self.attention =
BahdanauAttention(self.dec_units)

def call(self, x, hidden, enc_output):
```

```
        # enc_output shape == (batch_size, max_length, hidden_size)
        context_vector, attention_weights = self.attention(hidden, enc_output)

        # x shape after passing through embedding == (batch_size, 1,
        embedding_dim) x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim +
hidden_size) x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-
        1)

        # passing the concatenated vector to the GRU
        output, state = self.gru(x)

        # output shape == (batch_size * 1, hidden_size)
        output = tf.reshape(output, (-1, output.shape[2]))

        # output shape == (batch_size,

        vocab) x = self.fc(output) return x,

        state, attention_weights
```

In[29]:

```
decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH_SIZE)

sample_decoder_output, _, _ = decoder(tf.random.uniform((BATCH_SIZE, 1)),
                                      sample_hidden, sample_output)

print ('Decoder output shape: (batch_size, vocab size)
{}'.format(sample_decoder_output.shape))
```

Decoder output shape: (batch_size, vocab size) (64, 2347)

In[30]:

```
optimizer = tf.keras.optimizers.Adam() loss_object =
    tf.keras.losses.SparseCategoricalCrossentropy(
        from_logits=True, reduction='none')

                    def loss_function(real, pred):
```

```
    mask = tf.math.logical_not(tf.math.equal(real, 0))
                loss_ = loss_object(real, pred)

                mask = tf.cast(mask,

  dtype=loss_.dtype) loss_ *= mask return

                tf.reduce_mean(loss_)
```

In[31]:

```
@tf.function def train_step(inp, targ,
          enc_hidden): loss = 0

          with tf.GradientTape() as tape: enc_output,

      enc_hidden = encoder(inp, enc_hidden) dec_hidden

                              = enc_hidden


            dec_input = tf.expand_dims([targ_lang.word_index['<sos>']] *
                                       BATCH_SIZE, 1)

      # Teacher forcing - feeding the target as the next input
                  for t in range(1, targ.shape[1]):

                  # passing enc_output to the decoder predictions,
                  dec_hidden, _ = decoder(dec_input, dec_hidden,

    enc_output) loss += loss_function(targ[:, t],

                              predictions)


          # using teacher forcing dec_input = tf.expand_dims(targ[:, t], 1)

                  batch_loss = (loss / int(targ.shape[1])) variables =

    encoder.trainable_variables + decoder.trainable_variables gradients

                              = tape.gradient(loss, variables)

    optimizer.apply_gradients(zip(gradients, variables)) return batch_loss
```

In[32]:

```
EPOCHS = 40

for epoch in range(1, EPOCHS + 1):

    enc_hidden = encoder.initialize_hidden_state()
    total_loss = 0

    for (batch, (inp, targ)) in enumerate(dataset.take(steps_per_epoch)):

        batch_loss = train_step(inp, targ, enc_hidden)
        total_loss += batch_loss

        if(epoch % 4 == 0):

            print('Epoch:{:3d} Loss:{:.4f}'.format(epoch,
                                                   total_loss / steps_per_epoch))
```

```
2022-10-20 06:34:22.115124: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the
MLIR Optimization Passes are enabled (registered 2)

Epoch: 4 Loss:1.5734

Epoch: 8 Loss:1.3385

Epoch: 12 Loss:1.1549

Epoch: 16 Loss:0.9987

Epoch: 20 Loss:0.8251

Epoch: 24 Loss:0.6379

Epoch: 28 Loss:0.4403

Epoch: 32 Loss:0.2550

Epoch: 36 Loss:0.1160
Epoch: 40 Loss:0.0544
```

In[33]:

```
def evaluate(sentence): sentence
        = clean_text(sentence)

   inputs = [inp_lang.word_index[i] for i in sentence.split(' ')] inputs =
       tf.keras.preprocessing.sequence.pad_sequences([inputs],

                                                     maxlen=max_length_inp,
                                                         padding='post')

                                inputs = tf.convert_to_tensor(inputs)

                                                    result = ''

          hidden = [tf.zeros((1, units))] enc_out,
          enc_hidden = encoder(inputs, hidden)

                  dec_hidden = enc_hidden dec_input =
            tf.expand_dims([targ_lang.word_index['<sos>']], 0)

                                  for t in range(max_length_targ):

      predictions, dec_hidden, attention_weights = decoder(dec_input,

                                                       dec_hidden,
                                                         enc_out)

              # storing the attention weights to plot later on

         attention_weights = tf.reshape(attention_weights, (-1,

          )) predicted_id = tf.argmax(predictions[0]).numpy()

           result += targ_lang.index_word[predicted_id] + ' '


                       if targ_lang.index_word[predicted_id] == '<eos>':

                      return remove_tags(result), remove_tags(sentence)

           # the predicted ID is fed back into the model

     dec_input = tf.expand_dims([predicted_id], 0) return

         remove_tags(result), remove_tags(sentence)
```

In[34]:

```
questions =[] answers = [] with open("../input/simple-dialogs-for-
chatbot/dialogs.txt",'r') as f :

                    for line in f :

    line = line.split('\t') questions.append(line[0])
            answers.append(line[1])

                print(len(question) == len(answer))
```

True

In[35]:

```
                def ask(sentence):

    result, sentence = evaluate(sentence)
      print('Question: %s' % (sentence))
  print('Predicted answer: {}'.format(result))

                ask(questions[100])
```

Out[35]:

```
            Question: <sos> i believe so <eos>

          Predicted answer: good good you are hot <eos>
```

In[36]:

```
                ask(questions[50])

          Question: <sos> i wish it would cool off one day <eos>

    Predicted answer: that is how i feel i want winter to come soon <eos>
```

In[37]:

```
print(answers[50]) that's how i feel, i want

          winter to come soon.
```

CHATBOT WEB APPLICATION

- Creating a chatbot web application involves combining web development and chatbot development to offer a chatbot interface within a web application.
- A chatbot is software that simulates human-like conversations with users via chat. Its key task is to answer user questions with instant messages.

PROGRAM

In[1]:

```
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[2]:

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

```
Dataframe size: 3725
```

Out[2]:

	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.

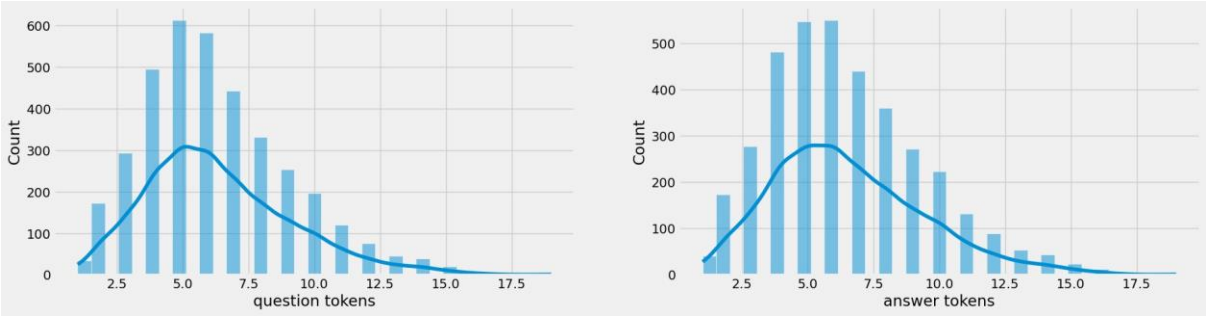
Data Preprocessing:

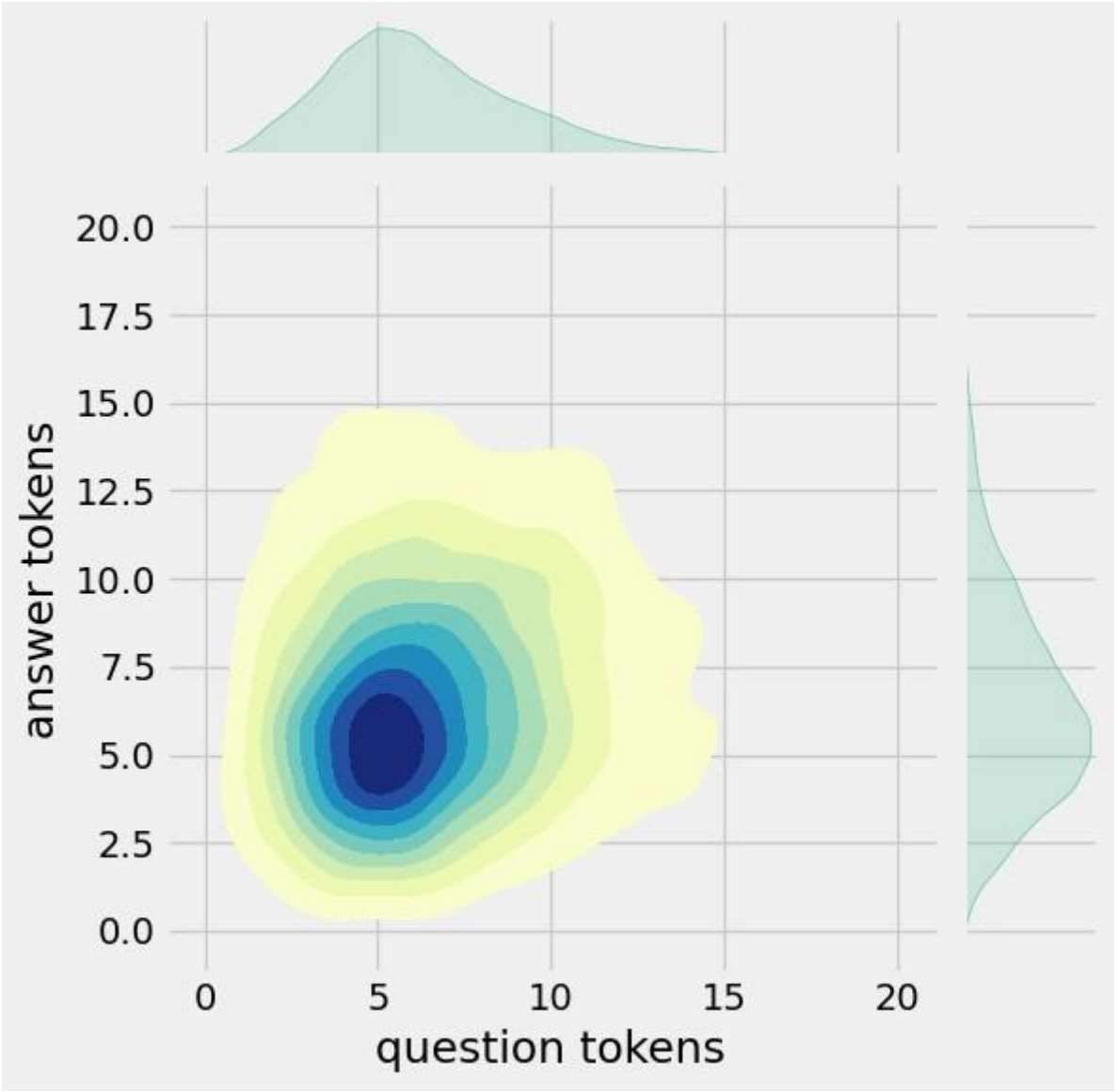
Data Visualization:

In[3]:

```
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()
```

Out[3]:





Text Cleaning:

In[4]:

```
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[4]:
```

	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...

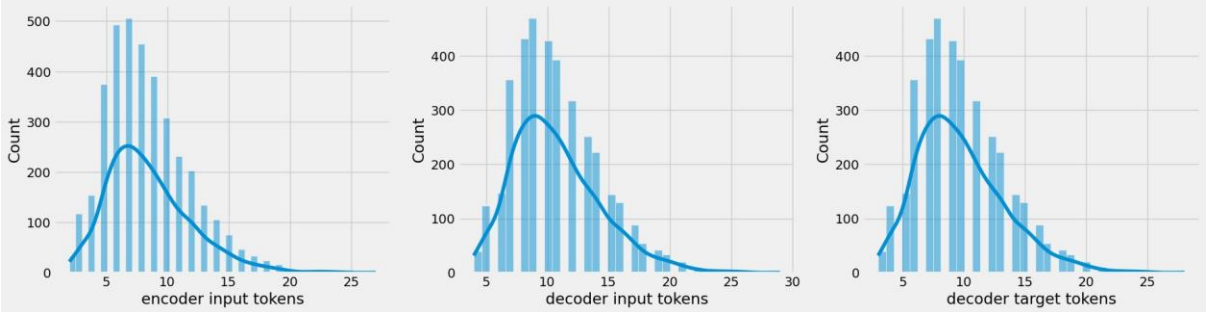
2	i'm pretty good. thanks for asking.	no problem. so how have you been?	i ' m pretty good . thanks for asking .	no problem . so how have you been ? <end>	<start> no problem . so how have you been ? ...
3	no problem. so how have you been?	i've been great. what about you?	no problem . so how have you been ?	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>

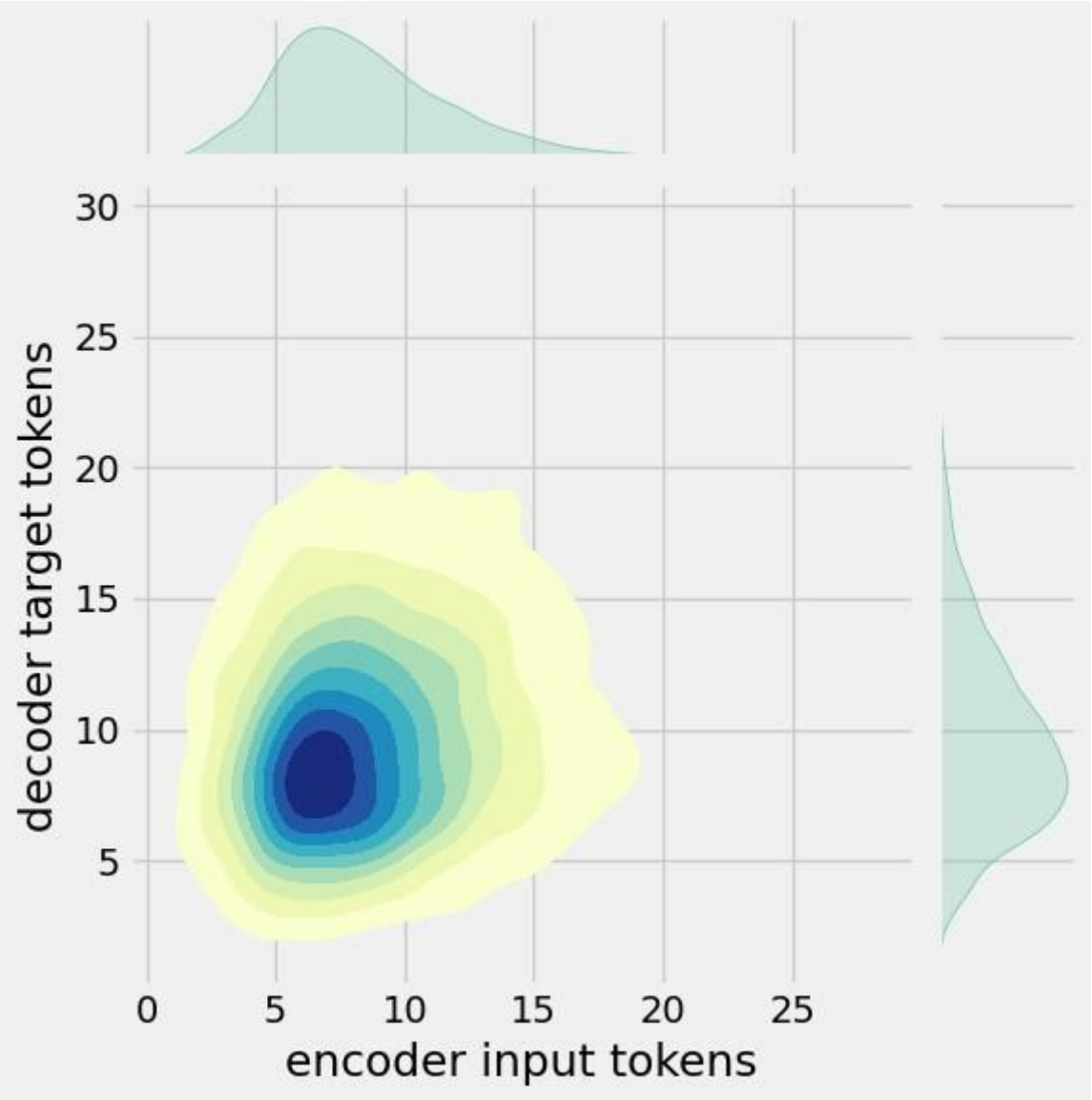
8	do you like it there?	it's okay. it's a really big campus.	do you like it there ?	it ' s okay . it ' s a really big campus . <...	<start> it ' s okay . it ' s a really big cam...
9	it's okay. it's a really big campus.	good luck with school.	it ' s okay . it ' s a really big campus .	good luck with school . <end>	<start> good luck with school . <end>

In[5]:

```
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()
```

Out[5]:





In[6]:

```
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[6]:

	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[7]:

```
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[8]:

```
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}') Out[8]:

  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[9]:


               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]} ...')

                            Out[9]:

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[10]:

```
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}')
```

Out[10]:

```
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[11]:

```
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_di
m 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[11]:

```
(<tf.Tensor: shape=(149, 256), dtype=float32, numpy= array([[
   0.16966951, -0.10419625, -0.12700348, ..., -0.12251794,

                        0.10568858, 0.14841646],

        [ 0.08443093, 0.08849293, -0.09065959, ..., -0.00959182,
                        0.10152507, -0.12077457],

        [ 0.03628462, -0.02653611, -0.11506603, ..., -0.14669597,

                        0.10292757, 0.13625325],

                                ...,

        [-0.14210635, -0.12942064, -0.03288083, ..., 0.0568463 , -
                        0.02598592, -0.22455114],

        [ 0.20819993, 0.01196991, -0.09635217, ..., -0.18782297,
                        0.10233591, 0.20114912],

        [ 0.1164271 , -0.07769038, -0.06414707, ..., -0.06539135,

           -0.05518465, 0.25142196]], dtype=float32)>,
    <tf.Tensor: shape=(149, 256), dtype=float32, numpy=

    array([[ 0.34589 , -0.30134732, -0.43572 , ..., -0.3102559 ,
                        0.34630865, 0.2613009 ],

        [ 0.14154069, 0.17045322, -0.17749965, ..., -0.02712595,
                        0.17292541, -0.2922624 ],

        [ 0.07106856, -0.0739173 , -0.3641197 , ..., -0.3794833 ,

                        0.36470377, 0.23766585],

                                ...,

        [-0.2582597 , -0.25323495, -0.06649272, ..., 0.16527973, -
                        0.04292646, -0.58768904],
```

```
[ 0.43155715, 0.03135502, -0.33463806, ..., -0.47625306,
                    0.33486888, 0.35035062],

  [ 0.23173636, -0.20141824, -0.22034441, ..., -0.16035017,

-0.17478186, 0.48899865]], dtype=float32)>)
```

Build Encoder## Build Decoder In[12]:

```
                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_sta

tes) 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[12]:

```
<tf.Tensor: shape=(1, 30, 2443), dtype=float32, numpy=
array([[[3.4059247e-04, 5.7348556e-05, 2.1294907e-05, ..., 7.2067953e-
        05, 1.5453645e-03, 2.3599296e-04],

        [1.4662130e-03, 8.0250365e-06, 5.4062020e-05, ..., 1.9187471e-
        05, 9.7244098e-05, 7.6433855e-05],

        [9.6929165e-05, 2.7441782e-05, 1.3761305e-03, ...,

        3.6009602e-05, 1.5537882e-04, 1.8397317e-04],

        ...,

        [1.9002777e-03, 6.9266016e-04, 1.4346189e-04, ..., 1.9552530e-
        04, 1.7106640e-05, 1.0252406e-04],

        [1.9002777e-03, 6.9266016e-04, 1.4346189e-04, ..., 1.9552530e-
        04, 1.7106640e-05, 1.0252406e-04],

        [1.9002777e-03, 6.9266016e-04, 1.4346189e-04, ...,

        1.9552530e-04, 1.7106640e-05, 1.0252406e-04]]],

        dtype=float32)>
```

Build Training Mode:

In[13]:

```
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_varia

 bles_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[14]:

     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[14]:

```
<tf.Tensor: shape=(149, 30, 2443), dtype=float32, numpy=
array([[[3.40592262e-04, 5.73484940e-05, 2.12948853e-05, ...,
        7.20679745e-05, 1.54536311e-03, 2.35993255e-04],

       [1.46621116e-03, 8.02504110e-06, 5.40619949e-05, ...,
        1.91874733e-05, 9.72440175e-05, 7.64339056e-05],

       [9.69291723e-05, 2.74417835e-05, 1.37613132e-03, ...,

        3.60095728e-05, 1.55378671e-04, 1.83973272e-04],

                                ...,
```

```
 [1.90027885e-03, 6.92659756e-04, 1.43461803e-04, ...,
    1.95525470e-04, 1.71066222e-05, 1.02524005e-04],

 [1.90027885e-03, 6.92659756e-04, 1.43461803e-04, ...,
    1.95525470e-04, 1.71066222e-05, 1.02524005e-04],

 [1.90027885e-03, 6.92659756e-04, 1.43461803e-04, ...,
   1.95525470e-04, 1.71066222e-05, 1.02524005e-04]],

[[9.24730921e-05, 3.46553512e-04, 2.07866033e-05, ...,
    3.65934626e-04, 7.63039337e-04, 5.52638434e-04],

 [8.46863186e-05, 3.65541164e-05, 2.54740953e-05, ...,
    7.12379551e-05, 3.62201303e-04, 4.16714087e-04],

 [2.30146630e-04, 3.91469621e-06, 2.72463716e-04, ...,

    9.26126595e-05, 1.03836363e-04, 1.40792166e-04],

                          ...,

 [6.84961735e-04, 9.07644513e-04, 2.86691647e-04, ...,
    3.87946144e-04, 6.09236558e-05, 1.12995331e-05],

 [6.84961735e-04, 9.07644513e-04, 2.86691647e-04, ...,
    3.87946144e-04, 6.09236558e-05, 1.12995331e-05],

 [6.84961735e-04, 9.07644513e-04, 2.86691647e-04, ...,
   3.87946144e-04, 6.09236558e-05, 1.12995322e-05]],

[[1.19036995e-03, 8.10516722e-05, 2.42324077e-05, ...,
    4.99442758e-05, 6.67208573e-04, 9.55566764e-04],

 [1.53046989e-04, 9.76863957e-05, 4.96972689e-06, ...,
    3.24743196e-05, 2.12563842e-04, 1.18708890e-03],

 [9.40205529e-04, 1.80782794e-04, 7.26205144e-06, ...,

    1.96355060e-04, 8.16940737e-05, 1.38416886e-03],

                          ...,

 [3.52622545e-03, 1.26781175e-03, 1.02695449e-04, ...,
    2.35450850e-03, 3.25187625e-06, 9.46984728e-05],

 [3.52622545e-03, 1.26781175e-03, 1.02695449e-04, ...,
    2.35450850e-03, 3.25187625e-06, 9.46984728e-05],
```

```
    [3.52622545e-03, 1.26781175e-03, 1.02695449e-04, ...,

      2.35450850e-03, 3.25187625e-06, 9.46984728e-05]],

                            ...,

    [[9.03617911e-05, 1.57651404e-04, 1.02747028e-04, ...,
       2.20922651e-04, 3.61504179e-04, 2.32456136e-03],

    [1.55469708e-04, 1.53608169e-04, 1.14945491e-04, ...,

1.88878359e-04, 5.11967926e-04, 5.13108505e-04], [8.27641197e-
           05, 2.83437112e-05, 6.29429938e-04, ...,

       2.15980137e-04, 3.02832137e-04, 1.77760507e-04],

                            ...,

    [2.41102395e-03, 1.29279669e-03, 9.11735406e-05, ...,
       4.06600971e-04, 7.58682154e-06, 6.05909081e-05],

    [2.41102395e-03, 1.29279669e-03, 9.11735406e-05, ...,
       4.06600971e-04, 7.58682154e-06, 6.05909081e-05],

    [2.41102395e-03, 1.29279669e-03, 9.11735406e-05, ...,
      4.06600971e-04, 7.58682154e-06, 6.05909081e-05]],

    [[3.99837241e-04, 2.36026899e-05, 6.89777007e-05, ...,
       5.94239136e-05, 4.32556757e-04, 4.60232928e-04],

    [3.88111075e-04, 8.31133584e-05, 1.11861555e-04, ...,
       3.03280340e-05, 2.54765386e-04, 2.82170397e-04],

    [2.12516752e-03, 7.19837190e-05, 1.88700986e-04, ...,

       1.86366087e-04, 7.02239413e-05, 2.54370330e-04],

                            ...,

    [4.56329063e-03, 2.23812275e-03, 2.37343236e-04, ...,
       2.64523784e-04, 4.05454011e-05, 1.55662783e-04],

    [4.56329063e-03, 2.23812275e-03, 2.37343236e-04, ...,
       2.64523784e-04, 4.05454011e-05, 1.55662783e-04],

    [4.56329063e-03, 2.23812275e-03, 2.37343236e-04, ...,
      2.64523784e-04, 4.05454011e-05, 1.55662783e-04]],
```

```
[[3.24600202e-04, 9.31067043e-05, 4.60048941e-05, ...,
   6.66230699e-05, 5.76460850e-04, 1.52416309e-04],

 [7.51478728e-05, 7.63997741e-05, 2.09082973e-05, ...,
   2.55555002e-04, 2.28998848e-04, 4.37303359e-04],

 [1.03114333e-04, 1.55743372e-04, 9.97955431e-06, ...,

   1.12485175e-03, 4.80950950e-03, 6.83143327e-04],

                                  ...,

 [5.20280097e-03, 3.23211338e-04, 2.47709468e-05, ...,
   3.07609705e-04, 6.09844255e-06, 8.61325825e-05],

 [5.20280097e-03, 3.23211338e-04, 2.47709468e-05, ...,
   3.07609705e-04, 6.09844255e-06, 8.61325825e-05],

 [5.20280097e-03, 3.23211338e-04, 2.47709468e-05, ...,

  3.07609705e-04, 6.09844255e-06, 8.61325825e-05]]],

                      dtype=float32)>
```

Train Model:

In[15]:

```
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

                                    )

  ]
)
```

Out[15]:

```
Epoch 1/100
```

23/23 [==============================] - ETA: 0s - loss: 1.6590
accuracy: 0.2180

Epoch 1: val_loss improved from inf to 1.21875, saving model to ckpt
23/23 [==============================] - 68s 3s/step - loss: 1.6515
accuracy: 0.2198 - val_loss: 1.2187 - val_accuracy: 0.3072

Epoch 2/100

23/23 [==============================] - ETA: 0s - loss: 1.2327
accuracy: 0.3087

Epoch 2: val_loss improved from 1.21875 to 1.10877, saving model to
ckpt

23/23 [==============================] - 53s 2s/step - loss: 1.2287
accuracy: 0.3092 - val_loss: 1.1088 - val_accuracy: 0.3415

Epoch 3/100

23/23 [==============================] - ETA: 0s - loss: 1.1008
accuracy: 0.3368

Epoch 3: val_loss did not improve from 1.10877

23/23 [==============================] - 22s 973ms/step - loss: 1.0984

- accuracy: 0.3370 - val_loss: 1.1161 - val_accuracy: 0.3315 Epoch
4/100

23/23 [==============================] - ETA: 0s - loss: 1.0209
accuracy: 0.3536

Epoch 4: val_loss improved from 1.10877 to 0.95189, saving model to
ckpt

23/23 [==============================] accuracy:

Epoch

53s 2s/step - loss: 1.0186 -

0.3540 - val_loss: 0.9519 - val_accuracy: 0.3718
5/100

23/23 [==============================] - ETA: 0s - loss: 0.9622
accuracy: 0.3673

Epoch 5: val_loss did not improve from 0.95189

23/23 [==============================] - 23s 979ms/step - loss: 0.9672

- accuracy: 0.3670 - val_loss: 0.9642 - val_accuracy: 0.3666

Epoch 6/100

23/23 [==============================] - ETA: 0s - loss: 0.9159
accuracy: 0.3801

Epoch 6: val_loss improved from 0.95189 to 0.94015, saving model to
ckpt

23/23 [==============================] - 53s 2s/step - loss: 0.9182
accuracy: 0.3796 - val_loss: 0.9401 - val_accuracy: 0.3598

Epoch 7/100

23/23 [==============================] - ETA: 0s - loss: 0.8737
accuracy: 0.3908

Epoch 7: val_loss improved from 0.94015 to 0.83293, saving model to
ckpt

23/23 [==============================] - 52s 2s/step - loss: 0.8746
accuracy: 0.3900 - val_loss: 0.8329 - val_accuracy: 0.4180

Epoch 8/100

23/23 [==============================] - ETA: 0s - loss: 0.8389
accuracy: 0.4013

23/23 [==============================] accuracy:

Epoch

Epoch 8: val_loss improved from 0.83293 to 0.77748, saving model to ckpt

23/23 [==============================] - 53s 2s/step - loss: 0.8395 accuracy: 0.4013 - val_loss: 0.7775 - val_accuracy: 0.4305

Epoch 9/100

23/23 [==============================] - ETA: 0s - loss: 0.8148 accuracy: 0.4094

Epoch 9: val_loss did not improve from 0.77748

23/23 [==============================] - 23s 983ms/step - loss: 0.8187

- accuracy: 0.4084 - val_loss: 0.8608 - val_accuracy: 0.3830

Epoch 10/100

23/23 [==============================] - ETA: 0s - loss: 0.7889 accuracy: 0.4200

Epoch 10: val_loss improved from 0.77748 to 0.73131, saving model to ckpt

23/23 [==============================] - 53s 2s/step - loss: 0.7923 accuracy: 0.4188 - val_loss: 0.7313 - val_accuracy: 0.4515 Epoch 11/100

ETA: 0s - loss: 0.7624 -

0.4284

11: val_loss did not improve from 0.73131

23/23 [==============================] - 22s 965ms/step - loss: 0.7615

- accuracy: 0.4282 - val_loss: 0.8036 - val_accuracy: 0.4472

Epoch 12/100

23/23 [==============================] - ETA: 0s - loss: 0.7433 accuracy: 0.4361

23/23 [==============================] accuracy:

Epoch

Epoch 12: val_loss did not improve from 0.73131

23/23 [==============================] - 23s 984ms/step - loss: 0.7452

- accuracy: 0.4354 - val_loss: 0.7384 - val_accuracy: 0.4623

Epoch 13/100

23/23 [==============================] - ETA: 0s - loss: 0.7246
accuracy: 0.4493

Epoch 13: val_loss did not improve from 0.73131

23/23 [==============================] - 23s 988ms/step - loss: 0.7281

- accuracy: 0.4488 - val_loss: 0.8017 - val_accuracy: 0.4449

Epoch 14/100

23/23 [==============================] - ETA: 0s - loss: 0.7080
accuracy: 0.4513

Epoch 14: val_loss did not improve from 0.73131

23/23 [==============================] - 23s 995ms/step - loss: 0.7080

- accuracy: 0.4509 - val_loss: 0.7568 - val_accuracy: 0.4259

Epoch 15/100

23/23 [==============================] - ETA: 0s - loss: 0.6853
accuracy: 0.4620

Epoch 15: val_loss did not improve from 0.73131

23/23 [==============================] - 22s 974ms/step - loss: 0.6826

- accuracy: 0.4616 - val_loss: 0.7376 - val_accuracy: 0.4502

Epoch 16/100

23/23 [==============================] - ETA: 0s - loss: 0.6731
accuracy: 0.4673

23/23 [==============================] accuracy:

Epoch

Epoch 16: val_loss did not improve from 0.73131

23/23 [==============================] - 23s 983ms/step - loss: 0.6733

- accuracy: 0.4672 - val_loss: 0.7646 - val_accuracy: 0.4538

Epoch 17/100

23/23 [==============================] - ETA: 0s - loss: 0.6576
accuracy: 0.4732

Epoch 17: val_loss improved from 0.73131 to 0.66131, saving model to
ckpt

23/23 [==============================] - 52s 2s/step - loss: 0.6539
accuracy: 0.4738 - val_loss: 0.6613 - val_accuracy: 0.4714 Epoch
18/100

ETA: 0s - loss: 0.6468 -

0.4807

18: val_loss improved from 0.66131 to 0.65303, saving model to
ckpt

23/23 [==============================] - 53s 2s/step - loss: 0.6458
accuracy: 0.4805 - val_loss: 0.6530 - val_accuracy: 0.4993

Epoch 19/100

23/23 [==============================] - ETA: 0s - loss: 0.6353
accuracy: 0.4881

Epoch 19: val_loss did not improve from 0.65303

23/23 [==============================] - 23s 994ms/step - loss: 0.6357

- accuracy: 0.4876 - val_loss: 0.7331 - val_accuracy: 0.4677

Epoch 20/100

23/23 [==============================] - ETA: 0s - loss: 0.6194
accuracy: 0.4968

23/23 [==============================] accuracy:

Epoch

Epoch 20: val_loss improved from 0.65303 to 0.55054, saving model to ckpt

23/23 [==============================] - 54s 2s/step - loss: 0.6188 accuracy: 0.4967 - val_loss: 0.5505 - val_accuracy: 0.5221

Epoch 21/100

23/23 [==============================] - ETA: 0s - loss: 0.6160 accuracy: 0.4978

Epoch 21: val_loss did not improve from 0.55054

23/23 [==============================] - 23s 987ms/step - loss: 0.6182

- accuracy: 0.4965 - val_loss: 0.6790 - val_accuracy: 0.4979

Epoch 22/100

23/23 [==============================] - ETA: 0s - loss: 0.6011 accuracy: 0.5052

Epoch 22: val_loss did not improve from 0.55054

23/23 [==============================] - 23s 996ms/step - loss: 0.6011

- accuracy: 0.5051 - val_loss: 0.6221 - val_accuracy: 0.5277

Epoch 23/100

23/23 [==============================] - ETA: 0s - loss: 0.5950 accuracy: 0.5079

Epoch 23: val_loss did not improve from 0.55054

23/23 [==============================] - 23s 987ms/step - loss: 0.5934

- accuracy: 0.5081 - val_loss: 0.6142 - val_accuracy: 0.5198

Epoch 24/100

23/23 [==============================] - ETA: 0s - loss: 0.5810 accuracy: 0.5160

23/23 [==============================] accuracy:

Epoch

Epoch 24: val_loss did not improve from 0.55054

23/23 [==============================] - 22s 971ms/step - loss: 0.5803

- accuracy: 0.5170 - val_loss: 0.5759 - val_accuracy: 0.5137

```
Epoch 25/100
23/23 [==============================] - ETA: 0s - loss: 0.5716
accuracy: 0.5227
Epoch 25: val_loss did not improve from 0.55054
23/23 [==============================] - 23s 986ms/step - loss: 0.5733
- accuracy: 0.5229 - val_loss: 0.6344 - val_accuracy: 0.5169
Epoch 26/100
23/23 [==============================] - ETA: 0s - loss: 0.5676
accuracy: 0.5225
Epoch 26: val_loss did not improve from 0.55054
23/23 [==============================] - 22s 963ms/step - loss: 0.5708
- accuracy: 0.5210 - val_loss: 0.6254 - val_accuracy: 0.4882
Epoch 27/100
23/23 [==============================] - ETA: 0s - loss: 0.5616
accuracy: 0.5291
Epoch 27: val_loss did not improve from 0.55054
23/23 [==============================] - 23s 988ms/step - loss: 0.5624
- accuracy: 0.5280 - val_loss: 0.6774 - val_accuracy: 0.5379
Epoch 28/100
23/23 [==============================] - ETA: 0s - loss: 0.5531
accuracy: 0.5318
Epoch 28: val_loss did not improve from 0.55054
23/23 [==============================] - 22s 949ms/step - loss: 0.5543
- accuracy: 0.5310 - val_loss: 0.7284 - val_accuracy: 0.5302
Epoch 29/100
23/23 [==============================] - ETA: 0s - loss: 0.5398
accuracy: 0.5389
```

```
Epoch 29: val_loss did not improve from 0.55054

23/23 [==============================] - 23s 1s/step - loss: 0.5391
accuracy: 0.5398 - val_loss: 0.7385 - val_accuracy: 0.5193

Epoch 30/100

23/23 [==============================] - ETA: 0s - loss: 0.5375
accuracy: 0.5416

Epoch 30: val_loss improved from 0.55054 to 0.50346, saving model to
ckpt

23/23 [==============================] - 53s 2s/step - loss: 0.5384
accuracy: 0.5417 - val_loss: 0.5035 - val_accuracy: 0.5411

Epoch 31/100

23/23 [==============================] - ETA: 0s - loss: 0.5270
accuracy: 0.5481

Epoch 31: val_loss did not improve from 0.50346

23/23 [==============================] - 22s 958ms/step - loss: 0.5262

- accuracy: 0.5477 - val_loss: 0.5805 - val_accuracy: 0.5457

Epoch 32/100

23/23 [==============================] - ETA: 0s - loss: 0.5304
accuracy: 0.5447

Epoch 32: val_loss did not improve from 0.50346

23/23 [==============================] - 22s 963ms/step - loss: 0.5329

- accuracy: 0.5435 - val_loss: 0.5374 - val_accuracy: 0.5725

Epoch 33/100

23/23 [==============================] - ETA: 0s - loss: 0.5196
accuracy: 0.5520

Epoch 33: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 975ms/step - loss: 0.5211
```

- accuracy: 0.5518 - val_loss: 0.6217 - val_accuracy: 0.5066

Epoch 34/100

23/23 [==============================] - ETA: 0s - loss: 0.5129
accuracy: 0.5558

Epoch 34: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 1000ms/step - loss: 0.5129

- accuracy: 0.5556 - val_loss: 0.6070 - val_accuracy: 0.5653

Epoch 35/100

23/23 [==============================] - ETA: 0s - loss: 0.5059
accuracy: 0.5620

Epoch 35: val_loss did not improve from 0.50346

23/23 [==============================] - 22s 966ms/step - loss: 0.5081

- accuracy: 0.5614 - val_loss: 0.6153 - val_accuracy: 0.5452

Epoch 36/100

23/23 [==============================] - ETA: 0s - loss: 0.5037
accuracy: 0.5619

Epoch 36: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 980ms/step - loss: 0.5063

- accuracy: 0.5617 - val_loss: 0.5328 - val_accuracy: 0.5873

Epoch 37/100

23/23 [==============================] - ETA: 0s - loss: 0.4977
accuracy: 0.5682

Epoch 37: val_loss did not improve from 0.50346

23/23 [==============================] - 22s 969ms/step - loss: 0.4980

- accuracy: 0.5682 - val_loss: 0.5976 - val_accuracy: 0.5693

Epoch 38/100

23/23 [==============================] - ETA: 0s - loss: 0.4939 accuracy: 0.5704

Epoch 38: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 993ms/step - loss: 0.4953 - accuracy: 0.5687 - val_loss: 0.5937 - val_accuracy: 0.5236 Epoch 39/100

23/23 [==============================] - ETA: 0s - loss: 0.4860 accuracy: 0.5758

Epoch 39: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 986ms/step - loss: 0.4868 - accuracy: 0.5746 - val_loss: 0.6155 - val_accuracy: 0.5457

Epoch 40/100

23/23 [==============================] - ETA: 0s - loss: 0.4809 accuracy: 0.5778

Epoch 40: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 1s/step - loss: 0.4821 accuracy: 0.5760 - val_loss: 0.5046 - val_accuracy: 0.5662

Epoch 41/100

23/23 [==============================] - ETA: 0s - loss: 0.4781 accuracy: 0.5817

Epoch 41: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 990ms/step - loss: 0.4782 - accuracy: 0.5821 - val_loss: 0.5256 - val_accuracy: 0.5907

Epoch 42/100

23/23 [==============================] - ETA: 0s - loss: 0.4713 accuracy: 0.5836

Epoch 42: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 982ms/step - loss: 0.4729 - accuracy: 0.5824 - val_loss: 0.6387 - val_accuracy: 0.5456

Epoch 43/100

23/23 [==============================] - ETA: 0s - loss: 0.4641 accuracy: 0.5904

Epoch 43: val_loss did not improve from 0.50346

23/23 [==============================] - 23s 1s/step - loss: 0.4627 accuracy: 0.5908 - val_loss: 0.5668 - val_accuracy: 0.5741

Epoch 44/100

23/23 [==============================] - ETA: 0s - loss: 0.4608 accuracy: 0.5921

Epoch 44: val_loss improved from 0.50346 to 0.49920, saving model to ckpt

23/23 [==============================] - 53s 2s/step - loss: 0.4618 accuracy: 0.5920 - val_loss: 0.4992 - val_accuracy: 0.5768

Epoch 45/100

23/23 [==============================] - ETA: 0s - loss: 0.4592 accuracy: 0.5902

Epoch 45: val_loss did not improve from 0.49920

23/23 [==============================] - 22s 970ms/step - loss: 0.4599 - accuracy: 0.5887 - val_loss: 0.5423 - val_accuracy: 0.5854

Epoch 46/100

23/23 [==============================] - ETA: 0s - loss: 0.4535
accuracy: 0.5978

Epoch 46: val_loss improved from 0.49920 to 0.48429, saving model to
ckpt

23/23 [==============================] - 53s 2s/step - loss: 0.4552
accuracy: 0.5966 - val_loss: 0.4843 - val_accuracy: 0.6049

Epoch 47/100

23/23 [==============================] - ETA: 0s - loss: 0.4528
accuracy: 0.5987

Epoch 47: val_loss improved from 0.48429 to 0.47868, saving model to
ckpt

23/23 [==============================] - 54s 2s/step - loss: 0.4537
accuracy: 0.5990 - val_loss: 0.4787 - val_accuracy: 0.5906

Epoch 48/100

23/23 [==============================] - ETA: 0s - loss: 0.4441
accuracy: 0.6016

Epoch 48: val_loss did not improve from 0.47868

23/23 [==============================] - 23s 982ms/step - loss: 0.4439

- accuracy: 0.6025 - val_loss: 0.5746 - val_accuracy: 0.5542

Epoch 49/100

23/23 [==============================] - ETA: 0s - loss: 0.4436
accuracy: 0.6041

Epoch 49: val_loss did not improve from 0.47868
Epoch

23/23 [==============================] accuracy:

Epoch did not improve from

23/23 [==============================] - loss:

- accuracy: - val_loss: - val_accuracy:

Epoch

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch

23/23 [==============================] - 22s 951ms/step - loss: 0.4432

- accuracy: 0.6045 - val_loss: 0.5058 - val_accuracy: 0.5753

Epoch 50/100

23/23 [==============================] - ETA: 0s - loss: 0.4435
accuracy: 0.6033

Epoch 50: val_loss did not improve from 0.47868

23/23 [==============================] - 22s 949ms/step - loss: 0.4441

- accuracy: 0.6043 - val_loss: 0.6037 - val_accuracy: 0.5473

Epoch 51/100

23/23 [==============================] - ETA: 0s - loss: 0.4382
accuracy: 0.6069

Epoch 51: val_loss did not improve from 0.47868

23/23 [==============================] - 22s 957ms/step - loss: 0.4383

- accuracy: 0.6067 - val_loss: 0.5206 - val_accuracy: 0.6154 52/100

ETA: 0s - loss: 0.4293 -

0.6125

52: val_loss 0.47868

23s 971ms/step - 0.4284

0.6123 0.4997 0.5840

Epoch

23/23 [==============================] accuracy:

Epoch did not improve from

23/23 [==============================] - loss:

- accuracy: - val_loss: - val_accuracy:

Epoch

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch did not improve from

23/23 [==============================] - - accuracy:
- val_loss: -

Epoch

53/100

0.4309

0.6109

53: val_loss improved from 0.47868 to 0.42987, saving model to ckpt

23/23 [==============================] - 52s 2s/step - loss: 0.4317 accuracy: 0.6094 - val_loss: 0.4299 - val_accuracy: 0.6062

Epoch 54/100

23/23 [==============================] - ETA: 0s - loss: 0.4292 accuracy: 0.6120

Epoch 54: val_loss did not improve from 0.42987

23/23 [==============================] - 22s 980ms/step - loss: 0.4309

- accuracy: 0.6115 - val_loss: 0.6996 - val_accuracy: 0.5592

Epoch 55/100

23/23 [==============================] - ETA: 0s - loss: 0.4225 accuracy: 0.6115

Epoch 55: val_loss did not improve from 0.42987

Epoch

23/23 [==============================] accuracy:

Epoch did not improve from

23/23 [==============================] - loss:

- accuracy: - val_loss: - val_accuracy:

Epoch

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch

23/23 [==============================] - 22s 976ms/step - loss: 0.4224

- accuracy: 0.6102 - val_loss: 0.5500 - val_accuracy: 0.5769

Epoch 56/100

23/23 [==============================] - ETA: 0s - loss: 0.4220
accuracy: 0.6180

Epoch 56: val_loss did not improve from 0.42987

23/23 [==============================] - 23s 995ms/step - loss: 0.4236

- accuracy: 0.6169 - val_loss: 0.5689 - val_accuracy: 0.5817

Epoch 57/100

23/23 [==============================] - ETA: 0s - loss: 0.4173
accuracy: 0.6210

Epoch 57: val_loss did not improve from 0.42987

23/23 [==============================] - 22s 976ms/step - loss: 0.4161

- accuracy: 0.6217 - val_loss: 0.4614 - val_accuracy: 0.6048

Epoch 58/100

23/23 [==============================] - ETA: 0s - loss: 0.4183
accuracy: 0.6198

Epoch 58: val_loss did not improve from 0.42987

23/23 [==============================] - 23s 1s/step - loss: 0.4183
accuracy: 0.6201 - val_loss: 0.4372 - val_accuracy: 0.6067 59/100

Epoch

23/23 [==============================] accuracy:

Epoch did not improve from

23/23 [==============================] - loss:

- accuracy: - val_loss: - val_accuracy:

Epoch

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch did not improve from

23/23 [==============================] - - accuracy:
- val_loss: -

Epoch

ETA: 0s - loss: 0.4120 -

0.6251

59: val_loss 0.42987

23s 994ms/step - 0.4136

0.6237 0.6183 0.5948

60/100

0.4090

0.6239

60: val_loss 0.42987

23s 980ms/step loss: 0.4101

0.6225 0.5042 val_accuracy: 0.6161

61/100

23/23 [==============================] - ETA: 0s - loss: 0.4051
accuracy: 0.6314

Epoch 61: val_loss did not improve from 0.42987

23/23 [==============================] - 23s 1s/step - loss: 0.4077
accuracy: 0.6296 - val_loss: 0.5100 - val_accuracy: 0.6128
Epoch

23/23 [==============================] accuracy:

Epoch did not improve from

23/23 [==============================] - loss:

- accuracy: - val_loss: - val_accuracy:

Epoch

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch

Epoch 62/100

23/23 [==============================] - ETA: 0s - loss: 0.4016
accuracy: 0.6326

Epoch 62: val_loss did not improve from 0.42987

23/23 [==============================] - 24s 1s/step - loss: 0.4029
accuracy: 0.6322 - val_loss: 0.5295 - val_accuracy: 0.6005

Epoch 63/100

23/23 [==============================] - ETA: 0s - loss: 0.4049
accuracy: 0.6323

Epoch 63: val_loss did not improve from 0.42987

23/23 [==============================] - 23s 981ms/step - loss: 0.4069

- accuracy: 0.6316 - val_loss: 0.5103 - val_accuracy: 0.6088

Epoch 64/100

23/23 [==============================] - ETA: 0s - loss: 0.3951
accuracy: 0.6335

Epoch 64: val_loss did not improve from 0.42987

23/23 [==============================] - 22s 981ms/step - loss: 0.3943

- accuracy: 0.6341 - val_loss: 0.5366 - val_accuracy: 0.5869

Epoch 65/100

Epoch

23/23 [==============================] accuracy:

Epoch did not improve from

23/23 [==============================] - loss:

- accuracy: - val_loss: - val_accuracy:

Epoch

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch did not improve from

23/23 [==============================] - - accuracy:
- val_loss: -

Epoch

23/23 [==============================] - ETA: 0s - loss: 0.3967
accuracy: 0.6344

Epoch 65: val_loss improved from 0.42987 to 0.40702, saving model to
ckpt

23/23 [==============================] - 53s 2s/step - loss: 0.3972
accuracy: 0.6352 - val_loss: 0.4070 - val_accuracy: 0.6452 66/100

ETA: 0s - loss: 0.3942 -

0.6351

66: val_loss 0.40702

22s 961ms/step - 0.3954

0.6337 0.4963 0.6039

Epoch

23/23 [==============================] accuracy:

Epoch did not improve from

23/23 [==============================] - loss:

- accuracy: - val_loss: - val_accuracy:

Epoch

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch did not improve from

23/23 [==============================] - - accuracy:
- val_loss: -

Epoch

67/100

0.3884

0.6409

67: val_loss 0.40702

22s 951ms/step loss: 0.3879

0.6424 0.4651 val_accuracy: 0.6276

68/100

23/23 [==============================] - ETA: 0s - loss: 0.3876
accuracy: 0.6398

Epoch 68: val_loss improved from 0.40702 to 0.38016, saving model to
ckpt

23/23 [==============================] - 52s 2s/step - loss: 0.3870
accuracy: 0.6388 - val_loss: 0.3802 - val_accuracy: 0.6614

Epoch 69/100

23/23 [==============================] - ETA: 0s - loss: 0.3897
accuracy: 0.6394

Epoch 69: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 961ms/step - loss: 0.3895

- accuracy: 0.6395 - val_loss: 0.4046 - val_accuracy: 0.6587

Epoch 70/100

Epoch

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch did not improve from

23/23 [==============================] - - accuracy:
- val_loss: -

Epoch

23/23 [==============================] - ETA: 0s - loss: 0.3855
accuracy: 0.6433

Epoch 70: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 967ms/step - loss: 0.3870

- accuracy: 0.6432 - val_loss: 0.4162 - val_accuracy: 0.6475

Epoch 71/100

23/23 [==============================] - ETA: 0s - loss: 0.3828
accuracy: 0.6422

Epoch 71: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 986ms/step - loss: 0.3828

- accuracy: 0.6423 - val_loss: 0.4099 - val_accuracy: 0.6612

Epoch 72/100

23/23 [==============================] - ETA: 0s - loss: 0.3825
accuracy: 0.6460

Epoch 72: val_loss did not improve from 0.38016

23/23 [==============================] - 24s 1s/step - loss: 0.3831
accuracy: 0.6449 - val_loss: 0.5160 - val_accuracy: 0.6117

Epoch 73/100

23/23 [==============================] - ETA: 0s - loss: 0.3795
accuracy: 0.6451

Epoch 73: val_loss did not improve from 0.38016

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch did not improve from 0.38016

23/23 [==============================] - - accuracy:
- val_loss: -

Epoch

23/23 [==============================] -

23/23 [==============================] - 23s 1s/step - loss: 0.3797
accuracy: 0.6448 - val_loss: 0.4963 - val_accuracy: 0.6231

74/100

0.3769

0.6479

74: val_loss 0.38016

22s 975ms/step loss: 0.3783

0.6459 0.4888 val_accuracy: 0.6084

75/100

23/23 [==============================] - ETA: 0s - loss: 0.3719
accuracy: 0.6541

Epoch 75: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 971ms/step - loss: 0.3724

- accuracy: 0.6538 - val_loss: 0.5175 - val_accuracy: 0.6032

Epoch 76/100

23/23 [==============================] - ETA: 0s - loss: 0.3697
accuracy: 0.6555

Epoch 76: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 1s/step - loss: 0.3687
accuracy: 0.6548 - val_loss: 0.4598 - val_accuracy: 0.6059

Epoch 77/100

Epoch

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch did not improve from

23/23 [==============================] - - accuracy:
- val_loss: -

Epoch

23/23 [==============================] - ETA: 0s - loss: 0.3702
accuracy: 0.6552

Epoch 77: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 954ms/step - loss: 0.3713

- accuracy: 0.6540 - val_loss: 0.5650 - val_accuracy: 0.5824

Epoch 78/100

23/23 [==============================] - ETA: 0s - loss: 0.3685
accuracy: 0.6548

Epoch 78: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 982ms/step - loss: 0.3675

- accuracy: 0.6557 - val_loss: 0.4115 - val_accuracy: 0.6292

Epoch 79/100

23/23 [==============================] - ETA: 0s - loss: 0.3659
accuracy: 0.6584

Epoch 79: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 970ms/step - loss: 0.3662

- accuracy: 0.6577 - val_loss: 0.3868 - val_accuracy: 0.6516

Epoch 80/100

23/23 [==============================] - ETA: 0s - loss: 0.3626
accuracy: 0.6628

Epoch 80: val_loss did not improve from 0.38016

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch did not improve from 0.38016

23/23 [==============================] - - accuracy:
- val_loss: -

Epoch

23/23 [==============================] -

23/23 [==============================] - 23s 994ms/step - loss: 0.3627

- accuracy: 0.6638 - val_loss: 0.4733 - val_accuracy: 0.6388 Epoch
81/100

0.3623

0.6578

81: val_loss

22s 970ms/step loss: 0.3621

0.6577 0.5189 val_accuracy: 0.5979

82/100

ETA: 0s - loss: 0.3603 -

accuracy: 0.6612

Epoch 82: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 982ms/step - loss: 0.3600

- accuracy: 0.6614 - val_loss: 0.4210 - val_accuracy: 0.6280

Epoch 83/100

23/23 [==============================] - ETA: 0s - loss: 0.3608
accuracy: 0.6604

Epoch 83: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 1s/step - loss: 0.3627
accuracy: 0.6592 - val_loss: 0.5621 - val_accuracy: 0.6082

Epoch 84/100

Epoch

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch did not improve from

23/23 [==============================] - - accuracy:
- val_loss: -

Epoch

23/23 [==============================] - ETA: 0s - loss: 0.3605
accuracy: 0.6640

Epoch 84: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 998ms/step - loss: 0.3628

- accuracy: 0.6634 - val_loss: 0.4241 - val_accuracy: 0.6462

Epoch 85/100

23/23 [==============================] - ETA: 0s - loss: 0.3498
accuracy: 0.6713

Epoch 85: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 976ms/step - loss: 0.3484

- accuracy: 0.6713 - val_loss: 0.4425 - val_accuracy: 0.6489

Epoch 86/100

23/23 [==============================] - ETA: 0s - loss: 0.3537
accuracy: 0.6663

Epoch 86: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 1s/step - loss: 0.3543
accuracy: 0.6656 - val_loss: 0.4006 - val_accuracy: 0.6716

Epoch 87/100

23/23 [==============================] - ETA: 0s - loss: 0.3503
accuracy: 0.6698

Epoch 87: val_loss did not improve from 0.38016

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch did not improve from 0.38016

23/23 [==============================] - - accuracy:
- val_loss: -

Epoch

23/23 [==============================] -

23/23 [==============================] - 23s 987ms/step - loss: 0.3493

- accuracy: 0.6697 - val_loss: 0.4375 - val_accuracy: 0.6527 Epoch
88/100

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch did not improve from 0.38016

23/23 [==============================] - - accuracy:
- val_loss: -

Epoch

23/23 [==============================] -

0.3497

0.6714

88: val_loss

23s 986ms/step loss: 0.3495

0.6710 0.5339 val_accuracy: 0.6160

89/100

ETA: 0s - loss: 0.3500 -

accuracy: 0.6671

Epoch 89: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 970ms/step - loss: 0.3501

- accuracy: 0.6666 - val_loss: 0.4148 - val_accuracy: 0.6438

Epoch 90/100

23/23 [==============================] - ETA: 0s - loss: 0.3494
accuracy: 0.6661

Epoch 90: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 995ms/step - loss: 0.3529

- accuracy: 0.6647 - val_loss: 0.4992 - val_accuracy: 0.6324

Epoch 91/100

23/23 [==============================] - ETA: 0s - loss: 0.3479
accuracy: 0.6718

Epoch 91: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 986ms/step - loss: 0.3482

23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch did not improve from 0.38016

23/23 [==============================] - - accuracy:
- val_loss: -

Epoch

23/23 [==============================] -

- accuracy: 0.6715 - val_loss: 0.6037 - val_accuracy: 0.6195

Epoch 92/100

23/23 [==============================] - ETA: 0s - loss: 0.3436
accuracy: 0.6767

Epoch 92: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 964ms/step - loss: 0.3452

- accuracy: 0.6764 - val_loss: 0.4368 - val_accuracy: 0.6462

Epoch 93/100

23/23 [==============================] - ETA: 0s - loss: 0.3377
accuracy: 0.6793

Epoch 93: val_loss did not improve from 0.38016

23/23 [==============================] - 23s 984ms/step - loss: 0.3372

- accuracy: 0.6795 - val_loss: 0.5267 - val_accuracy: 0.6275

Epoch 94/100

23/23 [==============================] - ETA: 0s - loss: 0.3433
accuracy: 0.6743

Epoch 94: val_loss did not improve from 0.38016

23/23 [==============================] - 22s 964ms/step - loss: 0.3453

- accuracy: 0.6736 - val_loss: 0.4532 - val_accuracy: 0.6314 Epoch
95/100

0.3409

0.6780

95: val_loss

23s 987ms/step loss: 0.3407

0.6775 0.4901 val_accuracy: 0.6680

96/100

ETA: 0s - loss: 0.3378 -

accuracy: 0.6791

Epoch 96: val_loss did not improve from 0.38016

23/23 [==========] - 23s 991ms/step - loss: 0.3388

- accuracy: 0.6793 - val_loss: 0.5620 - val_accuracy: 0.6063

Epoch 97/100

23/23 [==========] - ETA: 0s - loss: 0.3389
accuracy: 0.6763

Epoch 97: val_loss improved from 0.38016 to 0.33265, saving model to ckpt

23/23 [==========] - 53s 2s/step - loss: 0.3402
accuracy: 0.6765 - val_loss: 0.3327 - val_accuracy: 0.6854

Epoch 98/100

23/23 [==========] - ETA: 0s - loss: 0.3408
accuracy: 0.6768

Epoch 98: val_loss did not improve from 0.33265

23/23 [==========] - 22s 974ms/step - loss: 0.3407

- accuracy: 0.6766 - val_loss: 0.4046 - val_accuracy: 0.6695

Epoch 99/100

23/23 [==========] - ETA: 0s - loss: 0.3388
accuracy: 0.6795

Epoch 99: val_loss did not improve from 0.33265

23/23 [==========] - 23s 985ms/step - loss: 0.3394

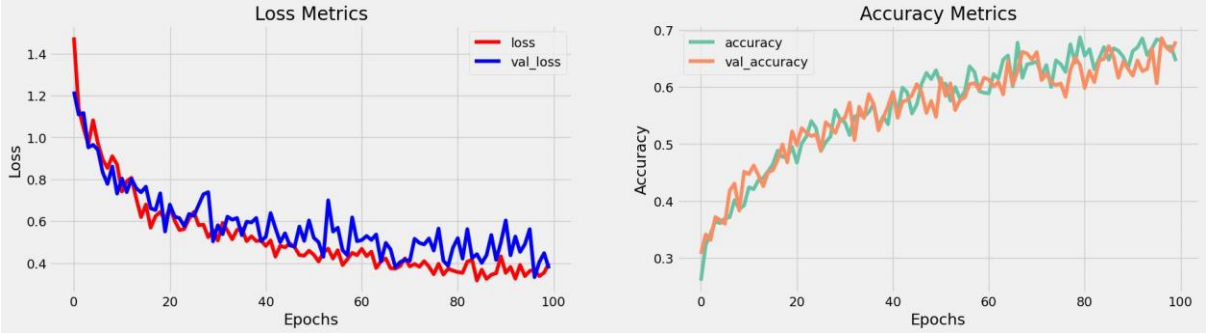
```
23/23 [==============================] - ETA: 0s - loss: accuracy:

Epoch              did not improve from 0.38016

23/23 [==============================] -      - accuracy:
                        - val_loss: -

Epoch

23/23 [==============================] -

- accuracy: 0.6791 - val_loss: 0.4475 - val_accuracy: 0.6622

Epoch 100/100

23/23 [==============================] - ETA: 0s - loss: 0.3358
                        accuracy: 0.6787

Epoch 100: val_loss did not improve from 0.33265

23/23 [==============================] - 22s 968ms/step - loss: 0.3385

- accuracy: 0.6773 - val_loss: 0.3742 - val_accuracy: 0.6796
```

Visualize Metrics:

In[16]:

```
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()
```

Out[16]:



Save Model:

In[17]:

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

In[18]:

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

Out[18]:

```
Encoder layers:

<keras.layers.core.embedding.Embedding object at 0x782084b9d190>
<keras.layers.normalization.layer_normalization.LayerNormalization
object at 0x7820e56f1b90>

<keras.layers.rnn.lstm.LSTM object at 0x7820841bd650>
--------------------Decoder layers:

<keras.layers.core.embedding.Embedding object at 0x78207c258590>
<keras.layers.normalization.layer_normalization.LayerNormalization
object at 0x78207c78bd10>

<keras.layers.rnn.lstm.LSTM object at 0x78207c258a10>

<keras.layers.core.dense.Dense object at 0x78207c2636d0>

----------
```

Create Inference Model:

In[19]:

```
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_st
ate_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,ini
tial_stat

e=[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='chatb

ot_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=Fa

lse)
  #

    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()
```

Out[19]:

```
Model: "chatbot_encoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 encoder_embedding (Embeddin (None, None, 256)         625408
 g)

 layer_normalization (LayerN (None, None, 256)         512
 ormalization)
```

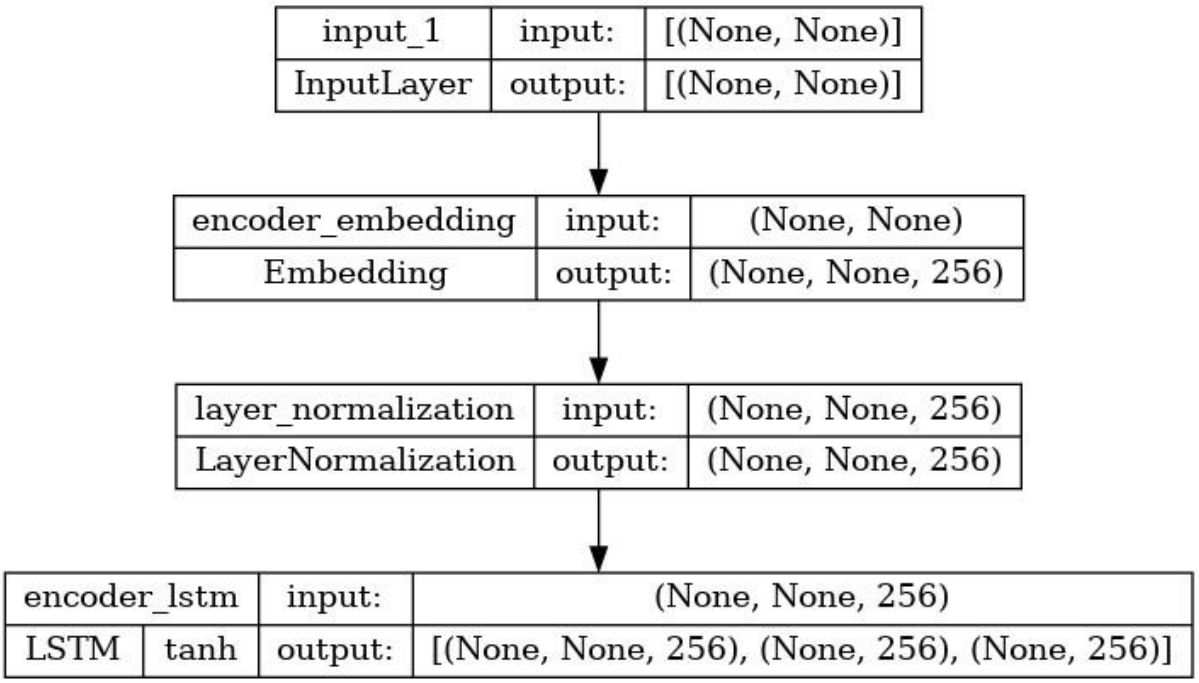
```
encoder_lstm (LSTM)          [(None, None, 256),       525312
                              (None, 256),
                              (None, 256)]
==================================================================

Total params: 1,151,232

Trainable params: 1,151,232

Non-trainable params: 0

_________________________________________________________________

Model: "chatbot_decoder"

_________________________________________________________________
                              _________________________________

Layer (type)                 Output Shape              Param #

                             Connected to

==================================================================
                              ================================

input_4 (InputLayer)         [(None, None)]            0           []

decoder_embedding (Embedding) (None, None, 256)        625408

                             ['input_4[0][0]']

layer_normalization (LayerNorm (None, None, 256)       512

['decoder_embedding[0][0]']
          alization)

input_2 (InputLayer)         [(None, 256)]             0           []

input_3 (InputLayer)         [(None, 256)]             0           []

                             [(None, None, 256),       525312
decoder_lstm (LSTM)
                              (None, 256),
['layer_normalization[1][0]',
                              (None, 256)]
 'input_2[0][0]',

 'input_3[0][0]']
decoder_dense (Dense)        (None, None, 2443)        627851
```

```
['decoder_lstm[0][0]']

==================================================================================

============================

Total params: 1,779,083

Trainable params: 1,779,083

Non-trainable params: 0

__________________________________________________________________________________

_________________________________
```

In[20]:

```
tf.keras.utils.plot_model(chatbot.encoder,to_file='encoder.png',show_shapes=True,show_layer_activations=True)
```

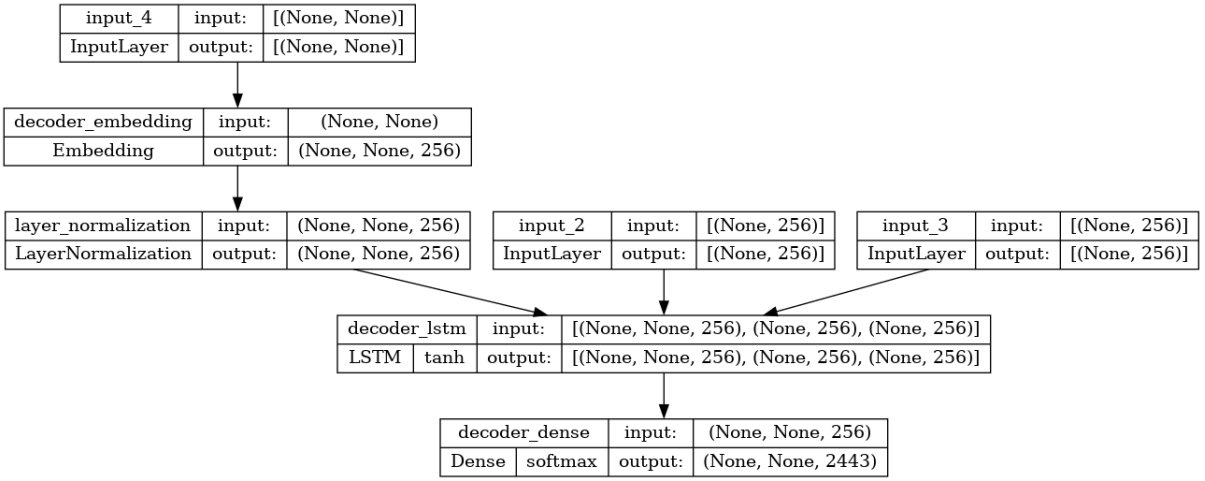
Out[20]:



In[21]:

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

Out[21]:



Time to Chat:

In[22]:

```
def print_conversation(texts):

    for text in texts:

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

In[23]:

```
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.'''

])

Out[23]:

You: hi

Bot: i have to go to the bathroom.

==========

You: do yo know me?

Bot: yes, it's too close to the other.

==========

You: what is your name?

Bot: i have to walk the house.

==========

You: you are bot?

Bot: no, i have. all my life.

==========

You: hi, how are you doing?

Bot: i'm going to be a teacher.

==========

You: i'm pretty good. thanks for asking.

Bot: no problem. i'll have to give you the english assignments from my mind.

==========

You: Don't ever be in a hurry
Bot: it's not a great.

==========

You: I'm gonna put some dirt in your eye
Bot: that's a good idea.

==========

You: You're trash

Bot: the tv news is reporting a bank robbery.

==========

You: I've read all your research on nano-technology
Bot: it's the weather. i've gone around the world.

==========

You: You want forgiveness? Get religion
Bot: no, i'll be my.

==========

You: While you're using the bathroom, i'll order some food.

Bot: don't order for me. i've been a cheater.

==========

You: Wow! that's terrible.

Bot: never park your car under the house.

==========

You: We'll be here forever.

Bot: we'll be there in half an hour.

==========

You: I need something that's reliable.

Bot: you need a car with low mileage.

```
========================

You: A speeding car ran a red light, killing the girl.

Bot: what happened?

========================

You: Tomorrow we'll have rice and fish for lunch.
Bot: i'll make a sandwich.

========================

You: I like this restaurant because they give you free bread.

Bot: well, i think that's a good idea.
========================
```

Chatbot benefits for businesses

- Improve service with every interaction.
- Collect customer feedback.
- Reduce customer requests.
- Detect customer intent for added context.
- Boost customer engagement.
- Streamline service with routing and triage.
- Boost sales.
- Increase lead generat.

Conclusion

- Chatbots are conversational tools that perform routine tasks efficiently. People like them because

they help them get through those tasks quickly so they can focus their attention on high-level, strategic, and engaging activities that require human capabilities that cannot be replicated by machines.

- Users can easily type their query in natural language and retrieve information.
- AI chatbots offered personalized, real-time feedback and on-demand support to users continuously and indefinitely.