`

Google Collab: GPT_Neo_(4) (1).ipynb - Colaboratory (google.com)

Source: Blender Bot 2.0: An open source chatbot that builds long-term memory and searches the internet (facebook.com)

Dataset: "blended_skill_talk" dataset from hugging face library

Abstract:

Social stigma still prevails around mental health issues in the Indian society. People sometimes struggle alone with no support, or the people are scared or to talk about this to other people or they just don't want to talk. To deal with this we want to build a generative chat bot that will be the place where people can interact and express their feelings. The chat bot will be the place where they just speak their mind out. Here the person won't be talking alone. He will be getting response from the AI so it will feel more natural without any breach to the person's privacy. There are some social media sites where you can do the same by making yourself anonymous but complete anonymity is not easy to achieve risking the loss of data privacy. The chat both will be safe from such issues as we won't be storing the user data making it more secure and safe.

Challenges:

- Selecting the domain:
  The first challenge was to get the requirements and exact domain we want to center the chat bot to be specific for. We wanted to work under the mental health domain, and we went with the emotional conversational chat-bot.

- Dataset selection:
  Now we started exploring the dataset which will be suitable for our project. There were already some pre-processed datasets that were available on GitHub, but their results didn't meet our expectations. So, went with the transformers model that is trained on massive datasets. They are trained on billions of datasets and generate more accurate and probable output.

- Selection of transformer model:
  Transformer models was a new challenge for us. We had to learn the concepts from the beginning and start exploring different chat-bot models that we could utilize to build our project upon. We faced issues like device compatibility, google collab memory running out because of the size of the model and asking us to get the premium version, etc. Another

`

major issue was that some of the popular and probably best performing models like chatgpt 3 or 4 are not open-source. We can access them only through the API. So we searched alternatives for this and stumbled upon a recently developed baize model. The Baize model is developed by researchers to facilitate research and exploration of the NLP models. The baize model is an effective model that generates its dataset by collecting the questionaries from the stackOverflow and quora and pass them to the chatgpt and the conversation keep on going until a natural end is reached and then those conversation are converted into dataset and this is passed to the Llama model where we fine tune the dataset. But again because of the compatibility issue with the system, we had to drop the plan of using the model.

- Model fine-tuning:
  Finally, we decided to work blenderbot-400M model that will read the text and give and appropriate reply to the text. We had to fine tune the mdel by selecting the paramet values that better suit our need and also providing a good prompt for the model.

- User Interface:
  Now after this, the implementation of chat-bot is was the challenge. We want to have the user give input in text, image and audio format. For this implementing the UI, various errors was encountered. We fixed all the errors and implemented the final project for the NLP.

Project Implementation Description:

- **Install dependencies:**

```
!pip3 install torch torchvision torchaudio
!pip install transformers
!pip install pytesseract
!sudo apt install tesseract-ocr
!pip install speechrecognition
!pip install datasets
!pip install gradio
```

⇨ Installing the necessary dependencies for our project

- **Import Libraries**

```
import warnings
warnings.filterwarnings(“ignore”)
from datasets import load_dataset
```

```
import pandas as pd
from transformers import pipeline
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
import pytesseract
import speech_recognition as sr
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import cv2
import gradio as gr
```

⇨ The above code imports necessary packages for Natural Language Processing (NLP), Computer Vision, and Gradio. It also sets up a machine learning pipeline for conversational tasks and creates a user interface using Gradio for easy use of the pipeline. Additionally, it filters out any warnings that may appear during the code execution.

- ## Dataset:

```
# Load the Blended Skill Talk dataset
dataset = load_dataset("blended_skill_talk")

# Convert the dictionary to a pandas DataFrame
df = pd.DataFrame.from_dict(dataset['train'])

# View the DataFrame
df.head()
```

⇨ We are importing the "blender skill task" (a facebook dataset) consisting of various conversations and its metadata from the huggig face website using a python library.

- ## Build Model:

```
tokenizer = AutoTokenizer.from_pretrained("facebook/blenderbot-400M-distill")


model = AutoModelForSeq2SeqLM.from_pretrained("facebook/blenderbot-400M-disti
```

⇨ The code initializes two objects, **tokenizer** and **model**, for performing Sequence-to-Sequence Learning using the pre-trained "facebook/blenderbot-400M-distill" model.

AutoTokenizer.from_pretrained("facebook/blenderbot-400M-distill") initializes the tokenizer object using the pre-trained "facebook/blenderbot-400M-distill" model for **tokenizing input tex**t data into numerical form, which can be used as input for machine learning models.

`

AutoModelForSeq2SeqLM.from_pretrained("facebook/blenderbot-400M-distill") initializes the model object using the pre-trained “facebook/blenderbot-400M-distill” model for **performing Sequence-to-Sequence Learning**. It uses the same pre-trained model as the tokenizer object, which ensures that the tokenizer and model are compatible and can be used together.

- ## Speech to text:

```
def speech_to_text(audio):
  with sr.AudioFile(audio) as source:
    audio = r.record(source)

    # Transcribe audio
    try:
        text = r.recognize_google(audio)
        return text
    except sr.UnknownValueError:
          return "Could not understand audio"
    except sr.RequestError as e:
          return "Could not request results"
```

⇨ The code defines a Python function named **speech_to_tex**t that takes an audio file as input, uses the **SpeechRecognition** library to recognize and **transcribe** speech from the audio file, and returns the transcribed text. The function also handles potential errors that may occur during the transcription process, such as the inability to recognize speech or request transcription results.

- ## Extract text from image:

```
def extract_text_from_image(image):
  extractedText = pytesseract.image_to_string(image,lang='eng+tam+hin+kan+tel+mal',config='--
psm 6')
  extractedText= extractedText.replace('\n', ' ')
  return extractedText
```

⇨ The code defines a Python function named extract_text_from_image that takes an image as input and uses the Tesseract OCR (Optical Character Recognition) library to extract text from the image. The function then formats the extracted text by replacing line breaks with spaces and returns the formatted text.

`

- ## Generate response from model

```
def generateText(input_text):
 if type(input_text)==np.ndarray:
  image=input_text
  text=extract_text_from_image(image)
 elif type(input_text)==str and input_text.split(".")[-1]=="wav":
  audio=input_text
  text=speech_to_text(audio)
 else:
  text=input_text
 encoded_input = tokenizer.encode(text, return_tensors="pt")
 output = model.generate(encoded_input,
                         max_length=150,
                         num_beams=5,
                         no_repeat_ngram_size=3,
                         early_stopping=True,
                         top_k=0,
                         temperature=0.9)
 op = tokenizer.decode(output[0], skip_special_tokens=True)
 return op
```

⇨ The code defines a Python function named **generateText** that takes input in three possible formats: an image file (as a NumPy array), an audio file (as a string with a '.wav' file extension), or a string of text. The function uses the **extract_text_from_image** if the input is an image and returns the text from the image or if the input is an audio it uses **speech_to_text** function to extract the text from the audio and then encodes the input data (in text format) using the **tokenizer** object initialized earlier. The function then generates output text using the **model** object with specific parameters such as maximum length, number of beams, and temperature, and returns the generated output text as a string.

- ## Image Processing to get Text

```
# original image
ori_image = cv2.imread("/content/2023-04-14_10h26_28.jpg") #Extracting the image
ori_img = cv2.cvtColor(ori_image, cv2.COLOR_BGR2RGB) # Appying color
plt.imshow(ori_img)
plt.axis('off')
plt.show()
```

```
fixed_img = cv2.resize(ori_img, None, fx=1.2, fy=1.2, interpolation=cv2.INTER_CUBIC)
plt.imshow(fixed_img)
plt.axis("off")
```

```
ogimg = cv2.cvtColor(fixed_img, cv2.COLOR_RGB2GRAY) #to convert an RGB image to gray scale
```

```
kernel = cv2.getStructuringElement(cv2.MORPH_CROSS,(3,3)) # library to create a structuring element for morphological operations.
dilation = cv2.dilate(ogimg, kernel, iterations=1) #applying dilation
plt.imshow(dilation)
plt.axis("off")
```

```
kernel = cv2.getStructuringElement(cv2.MORPH_CROSS,(3,3))
erosion= cv2.erode(dilation, kernel, iterations=1) # erostion
plt.imshow(erosion)
plt.axis("off")
```

```
binary = cv2.threshold(cv2.medianBlur(erosion, 3), 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1] # thresholding for converting the image to bianry
plt.imshow(binary)
plt.axis("off")
```

```
gsbin = binary.astype(np.uint8) #storing the binary image in 8 bit
# gsbin= 255 * gsbin
plt.imshow(gsbin)
```

```
image2 = cv2.cvtColor(255-gsbin, cv2.COLOR_GRAY2RGB) #to convert a grayscale image represented by the 255-gsbin variable to an RGB image.
plt.imshow(image2)
plt.axis('off')
plt.show()
```

```
extract_text_from_image(image2)
```

⇨ The code processes an image using various image processing techniques, such as dilation, erosion, and thresholding, to convert it into a binary image. Then it converts the binary image to a grayscale image and then to an RGB image. Finally, it extracts the text from the image using the "extract_text_from_image" function.

`

- **Passing Images to ChatBot Model to test the response:**

```
image1=cv2.imread("/content/1681447969 (1).jpg") #Extracting the image
image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
plt.imshow(image1)
plt.axis('off')
plt.show()
```

⇨ This code loads an image file and displays it using matplotlib's **imshow** function after converting the image color space from BGR to RGB.

- **Build a Gradio Interface for deploying the chatbot:**

```
with gr.Blocks() as demo:
    gr.Markdown("Blenderbot Chatbot")
    with gr.Tab("Text"):
        text_input = gr.inputs.Textbox(label="Enter your message here")
        text_output = gr.outputs.Textbox()
        text_button = gr.Button("Chat")
    with gr.Tab("Image"):
        image_input = gr.inputs.Image(label="Upload an image")
        image_output = gr.outputs.Textbox()
        image_button = gr.Button("IChat")
    with gr.Tab("Audio"):
        audio_input=gr.inputs.Audio(source="microphone",type="filepath")
        audio_output=gr.outputs.Textbox()
        audio_button=gr.Button("AChat")

    text_button.click(generateText, inputs=[text_input], outputs=text_output)
    image_button.click(generateText, inputs=[image_input], outputs=image_output)
    audio_button.click(generateText, inputs=[audio_input], outputs=audio_output)

if __name__ == "__main__":
    demo.launch()
```

⇨ This code defines a GUI interface for a chatbot using the Graphical User Interface (GUI) package **Gradio**. The interface has three tabs for different input types - text, image, and audio. Each tab has an input field and a button to trigger the chatbot. The **click** function of the button is assigned to the **generateText** function, which takes the input from the corresponding input field and returns the output from the model we trained to the corresponding output field. Finally, the launch function of the **demo** object is called to launch the GUI interface.

`

Conclusion:

The chat bot can accept input in the form of text, image and audio. We are taking the input and successfully generating appropriate response to the user. In future, we plan to integrate this into a website and provide a better interaction interface. Thus the chat bot using for mental health is a exciting field where there is scope for many solutions. We plan to keep exploring the field and develop exciting products.