CREATE A CHATBOT USING PYTHON

PHASE 2 SUBMISSION DOCUMENT.....

Project : Create Chatbot Using Python.

Introduction:

- Creating a chatbot using Python involves developing a program that can simulate human conversation.
- You'll need to decide on the type of chatbot, such as rule-based or AI-powered, select the right tools and libraries, design the conversation flow, write the code, test it thoroughly, and then deploy it.
- Continuous maintenance and improvement are crucial for keeping your chatbot effective.
- Python's versatility and the availability of NLP and ML libraries make it a powerful choice for chatbot development, offering the potential to enhance user experiences, automate tasks, and provide valuable support in various applications.

Content for phase 2 in project:

consider exploring advanced techniques like using pre-trained language models (e.g., GPT-3) to enhance the quality of responses.

Data Source :

A good data source for creating a chatbot should contain accurate ,complete ,and easy accessible one for users.

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

hi, how are you doing? i'm fine. how about yourself?
i'm fine. how about yourself? i'm pretty good. thanks for asking.
i'm pretty good. thanks for asking. no problem. so how have you been?
no problem. so how have you been? i've been great. what about you?
i've been great. what about you? i've been good. i'm in school right now.
i've been good. i'm in school right now. what school do you go to?
what school do you go to? i go to pcc.
i go to pcc. do you like it there?
do you like it there? it's okay. it's a really big campus.
it's okay. it's a really big campus. good luck with school.
good luck with school. thank you very much.
how's it going? i'm doing well. how about you?
i'm doing well. how about you? never better, thanks.
never better, thanks. so how have you been lately?
so how have you been lately? i've actually been pretty good. you?
i've actually been pretty good. you? i'm actually in school right now.
i'm actually in school right now. which school do you attend?
which school do you attend? i'm attending pcc right now.
i'm attending pcc right now. are you enjoying it there?
are you enjoying it there? it's not bad. there are a lot of people there.
it's not bad. there are a lot of people there. good luck with that.
good luck with that. thanks.
how are you doing today? i'm doing great. what about you?
i'm doing great. what about you? i'm absolutely lovely, thank you.
i'm absolutely lovely, thank you. everything's been good with you?
everything's been good with you? i haven't been better. how about yourself?
i haven't been better. how about yourself? i started school recently.
i started school recently. where are you going to school?
where are you going to school? i'm going to pe.....

.....

Pre-trained Language Models :

Pre-trained language models, such as GPT-3 (Generative Pre-trained Transformer 3), are a type of artificial intelligence model used for various natural language processing tasks. These models have gained significant attention and popularity in recent years due to their ability to understand and generate human-like text. Here's an explanation of pre-trained language models:

1.Pre-training:

Pre-trained language models are created by training on a massive corpus of text from the internet. They use a type of neural network architecture called a Transformer. During pre-training, the model learns to predict the next word in a sentence based on the context of the preceding words. This process helps the model learn grammar, vocabulary, and a general understanding of language from diverse sources of text.

2. Transfer Learning:

After pre-training on this vast amount of text, the model is fine-tuned for specific downstream tasks. Transfer learning is a key concept here. Instead of training a separate model for each task (e.g., text classification, language translation, question answering), pre-trained models can be adapted for various tasks by fine-tuning them on smaller, task-specific datasets. This fine-tuning process allows the model to specialize and adapt to specific tasks without having to start from scratch.

3. Versitality:

Pre-trained language models like GPT-3 are known for their versatility. They can be applied to a wide range of natural language understanding and generation tasks, including but not limited to:

Text generation: Creating human-like text, such as articles, stories, or code.

Sentiment analysis: Determining the sentiment (positive, negative, neutral) of a piece of text.

- Language translation: Translating text from one language to another.
- Text summarization: Condensing long texts into shorter summaries.
- Question answering: Providing answers to questions based on a given text.
- Fine-tuning and Customization:
- One of the strengths of pre-trained models is their adaptability. Users can fine-tune these models on specific datasets to make them more domain-specific or task-

specific. This allows organizations and developers to customize the model's behavior for their particular applications.

- ✓ In summary, pre-trained language models like GPT-3 are powerful tools that have the potential to revolutionize natural language processing tasks.
- ✓ They are versatile, adaptable, and accessible, but their use also comes with ethical responsibilities and the need to address biases and potential misuse.

A High-level overview for creating a chatbot using the Hugging Face Transformers library and the GPT-3 model :

Install necessary libraries:

You will need the Transformers library from Hugging Face and the Torch library for PyTorch. You can install them using pip:

---->pip install transformers torch

Set up your chatbot code:

python

Copy code

```
import torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load pre-trained GPT-3 model and tokenizer
model_name = "gpt2"
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
model = GPT2LMHeadModel.from_pretrained(model_name)

# Set the device (CPU or GPU)
device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)
```

```
# Function to generate a response
def generate_response(prompt):
    input_ids = tokenizer.encode(prompt, return_tensors="pt").to(device)
    output = model.generate(input_ids, max_length=100, num_return_sequences=1, pad_token_id=50256)
    response = tokenizer.decode(output[0], skip_special_tokens=True)
    return response

# Chat loop
print("Chatbot: Hi, how can I help you today?")
while True:
    user_input = input("You: ")
    if user_input.lower() == 'exit':
        print("Chatbot: Goodbye!")
        break
    response = generate_response(user_input)
    print("Chatbot:", response)
```

- This code sets up a chatbot using the GPT-2 model from Hugging Face's Transformers library.
- It takes user input, generates a response, and prints it
- You can extend and refine this code to fit your specific requirements.

Build Encoder :

```
class Encoder(tf.keras.models.Model):
    def __init__(self,units,embedding_dim,vocab_size,*args,**kwargs) -> None:
        super().__init__(*args,**kwargs)
        self.units=units
        self.vocab_size=vocab_size
        self.embedding_dim=embedding_dim
```

```
    self.embedding=Embedding(
        vocab_size,
        embedding_dim,
        name='encoder_embedding',
        mask_zero=True,
        embeddings_initializer=tf.keras.initializers.GlorotNormal()
    )
    self.normalize=LayerNormalization()
    self.lstm=LSTM(
        units,
        dropout=.4,
        return_state=True,
        return_sequences=True,
        name='encoder_lstm',
        kernel_initializer=tf.keras.initializers.GlorotNormal()
    )

  def call(self,encoder_inputs):
    self.inputs=encoder_inputs
    x=self.embedding(encoder_inputs)
    x=self.normalize(x)
    x=Dropout(.4)(x)
    encoder_outputs,encoder_state_h,encoder_state_c=self.lstm(x)
    self.outputs=[encoder_state_h,encoder_state_c]
    return encoder_state_h,encoder_state_c

encoder=Encoder(lstm_cells,embedding_dim,vocab_size,name='encoder')
```

Build Decoder :

```
class Decoder(tf.keras.models.Model):
  def __init__(self,units,embedding_dim,vocab_size,*args,**kwargs) -> None:
    super().__init__(*args,**kwargs)
    self.units=units
    self.embedding_dim=embedding_dim
    self.vocab_size=vocab_size
    self.embedding=Embedding(
        vocab_size,
        embedding_dim,
        name='decoder_embedding',
        mask_zero=True,
        embeddings_initializer=tf.keras.initializers.HeNormal()
    )
    self.normalize=LayerNormalization()
```

```
        self.lstm=LSTM(
            units,
            dropout=.4,
            return_state=True,
            return_sequences=True,
            name='decoder_lstm',
            kernel_initializer=tf.keras.initializers.HeNormal()
        )
        self.fc=Dense(
            vocab_size,
            activation='softmax',
            name='decoder_dense',
            kernel_initializer=tf.keras.initializers.HeNormal()
        )

    def call(self,decoder_inputs,encoder_states):
        x=self.embedding(decoder_inputs)
        x=self.normalize(x)
        x=Dropout(.4)(x)
        x,decoder_state_h,decoder_state_c=self.lstm(x,initial_state=encoder_states)
        x=self.normalize(x)
        x=Dropout(.4)(x)
        return self.fc(x)

decoder=Decoder(lstm_cells,embedding_dim,vocab_size,name='decoder')
```

Build Training Model :

```
class ChatBotTrainer(tf.keras.models.Model):
    def __init__(self,encoder,decoder,*args,**kwargs):
        super().__init__(*args,**kwargs)
        self.encoder=encoder
        self.decoder=decoder

    def loss_fn(self,y_true,y_pred):
        loss=self.loss(y_true,y_pred)
        mask=tf.math.logical_not(tf.math.equal(y_true,0))
        mask=tf.cast(mask,dtype=loss.dtype)
        loss*=mask
        return tf.reduce_mean(loss)

    def accuracy_fn(self,y_true,y_pred):
        pred_values = tf.cast(tf.argmax(y_pred, axis=-1), dtype='int64')
        correct = tf.cast(tf.equal(y_true, pred_values), dtype='float64')
```

```
        mask = tf.cast(tf.greater(y_true, 0), dtype='float64')
        n_correct = tf.keras.backend.sum(mask * correct)
        n_total = tf.keras.backend.sum(mask)
        return n_correct / n_total

    def call(self,inputs):
        encoder_inputs,decoder_inputs=inputs
        encoder_states=self.encoder(encoder_inputs)
        return self.decoder(decoder_inputs,encoder_states)

    def train_step(self,batch):
        encoder_inputs,decoder_inputs,y=batch
        with tf.GradientTape() as tape:
            encoder_states=self.encoder(encoder_inputs,training=True)
            y_pred=self.decoder(decoder_inputs,encoder_states,training=True)
            loss=self.loss_fn(y,y_pred)
            acc=self.accuracy_fn(y,y_pred)

        variables=self.encoder.trainable_variables+self.decoder.trainable_variables
        grads=tape.gradient(loss,variables)
        self.optimizer.apply_gradients(zip(grads,variables))
        metrics={'loss':loss,'accuracy':acc}
        return metrics

    def test_step(self,batch):
        encoder_inputs,decoder_inputs,y=batch
        encoder_states=self.encoder(encoder_inputs,training=True)
        y_pred=self.decoder(decoder_inputs,encoder_states,training=True)
        loss=self.loss_fn(y,y_pred)
        acc=self.accuracy_fn(y,y_pred)
        metrics={'loss':loss,'accuracy':acc}
        return metric
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']
)
```

CONCLUSION AND FUTURE WORK(Phase 2) :

Project Conclusion :

- In the Phase 2 conclusion, we will summarize the key findings and insights from the advanced regression techniques. We will reiterate the impact of these techniques on improving the accuracy and robustness of creating a chatbot.
- Future Work: We will discuss potential avenues for future work, such as incorporating additional data sources ,exploring deep learning model for prediction, or expanding the project into a web application with more features and interactivity