NAME : DEEPAK
ROLL NO : 717822i209
SUBJECT : SPEECH AND LANGUAGE PROCESSING
COURSE CODE : 21ID31

SPEECH AND LANGUAGE PROCESSING
[21ID31]
[ASSIGNMENT - 1]

LOGICAL REPRESENTATIONS OF SENTENCES

Question:
Logical Representations of Sentence Meaning:
Design a logical form representation for natural language sentences using first-order logic or lambda calculus. Develop a semantic parsing system that maps sentences to their corresponding logical forms and evaluate its accuracy on a dataset of semantic parsing examples.

CODE:-

```
import torch
from torch.utils.data import DataLoader, Dataset
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

# Step 1: Prepare the Dataset
data = [
   {"sentence": "What is the capital of France?", "logical_form": "capital(France)"},
   {"sentence": "Who is the president of the USA?", "logical_form": "president(USA)"},
   {"sentence": "List all countries in Europe.", "logical_form": "countries(Europe)"},
   # Add more examples here
]

# Step 2: Preprocess the Data
tokenizer = AutoTokenizer.from_pretrained("t5-small")

def preprocess_data(data):
   inputs = [example["sentence"] for example in data]
   targets = [example["logical_form"] for example in data]

   # Tokenize inputs and targets
   input_encodings = tokenizer(inputs, padding=True, truncation=True, return_tensors="pt")
   target_encodings = tokenizer(targets, padding=True, truncation=True, return_tensors="pt")

   return input_encodings, target_encodings

input_encodings, target_encodings = preprocess_data(data)

# Step 3: Define the Dataset
class SemanticParsingDataset(Dataset):
   def __init__(self, input_encodings, target_encodings):
      self.input_encodings = input_encodings
      self.target_encodings = target_encodings

   def __getitem__(self, idx):
      item = {
         key: val[idx].clone().detach() for key, val in self.input_encodings.items()
         if key != "token_type_ids"  # Exclude token_type_ids for T5
      }
      item["labels"] = self.target_encodings["input_ids"][idx].clone().detach()
      return item
```

```
    def __len__(self):
        return len(self.input_encodings["input_ids"])

dataset = SemanticParsingDataset(input_encodings, target_encodings)
dataloader = DataLoader(dataset, batch_size=8, shuffle=True)

# Step 4: Define the Model
model = AutoModelForSeq2SeqLM.from_pretrained("t5-small")

# Step 5: Train the Model
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-5)

for epoch in range(3):  # Number of epochs
    model.train()
    for batch in dataloader:
        optimizer.zero_grad()
        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        print(f"Epoch {epoch}, Loss: {loss.item()}")

# Step 6: Evaluate the Model
model.eval()

test_data = [
    {"sentence": "What is the capital of Germany?", "logical_form": "capital(Germany)"},
    {"sentence": "Who is the CEO of Apple?", "logical_form": "CEO(Apple)"},
]

test_input_encodings, test_target_encodings = preprocess_data(test_data)
test_dataset = SemanticParsingDataset(test_input_encodings, test_target_encodings)
test_dataloader = DataLoader(test_dataset, batch_size=2)

for batch in test_dataloader:
    with torch.no_grad():
        generated_ids = model.generate(batch["input_ids"])
        generated_text = tokenizer.batch_decode(generated_ids, skip_special_tokens=True)
        print(generated_text)

# Step 7: Save the Model
model.save_pretrained("semantic_parsing_model")
tokenizer.save_pretrained("semantic_parsing_model")
```

RESULT:

```
Epoch 0, Loss: 7.742893695831299
Epoch 1, Loss: 8.479798316955566
Epoch 2, Loss: 7.554184913635254
['Wie ist die Hauptstadt Deutschland?', 'Wer ist CEO Apple Apple?']
('semantic_parsing_model/tokenizer_config.json',
 'semantic_parsing_model/special_tokens_map.json',
 'semantic_parsing_model/spiece.model',
 'semantic_parsing_model/added_tokens.json',
 'semantic_parsing_model/tokenizer.json')
```

1. Dataset Preparation

A small dataset is created where each natural language sentence (sentence) maps to a corresponding logical form (logical_form).

Example:

- Input: "What is the capital of France?"
- Output: capital(France)

Logical forms are in a structured format (like functions with arguments).

2. Preprocessing the Data

- Tokenizer: T5-small tokenizer converts sentences into tokenized representations (numerical input IDs).
- Inputs: Sentences are tokenized for the model to understand.
- Targets: Logical forms are tokenized so the model can predict them as output.
- Result:
  - input_encodings: Tokenized versions of input sentences.
  - target_encodings: Tokenized versions of logical form

3. Creating a Custom Dataset Class

- **Dataset Class:**
  - **A custom dataset class (SemanticParsingDataset) is created to handle the tokenized data.**
  - **__getitem__: Retrieves individual items from the dataset.**
  - **__len__: Returns the number of samples in the dataset.**
- **Dataloader:**
  - **Loads data in batches (batch size = 8) for efficient training.**
  - **Shuffles data for randomness in training.**

4. Defining the Model

Model: The T5-small model is loaded.

- **Seq2SeqLM:** This type of model is designed for sequence-to-sequence tasks, like converting sentences to logical form

5. Training the Model

Optimizer: Uses AdamW to update model parameters based on gradients.

Training Loop:

- Runs for 3 epochs.
- In each epoch, the model:
  1. Processes a batch of data (batch).
  2. Computes the loss (difference between predicted and target logical forms).
  3. Updates weights using backpropagation (loss.backward() and optimizer.step()).
- Prints the loss after each batch for monitoring.

6. Evaluating the Model

Test Data: New examples are provided to test the trained model.

Evaluation Steps:

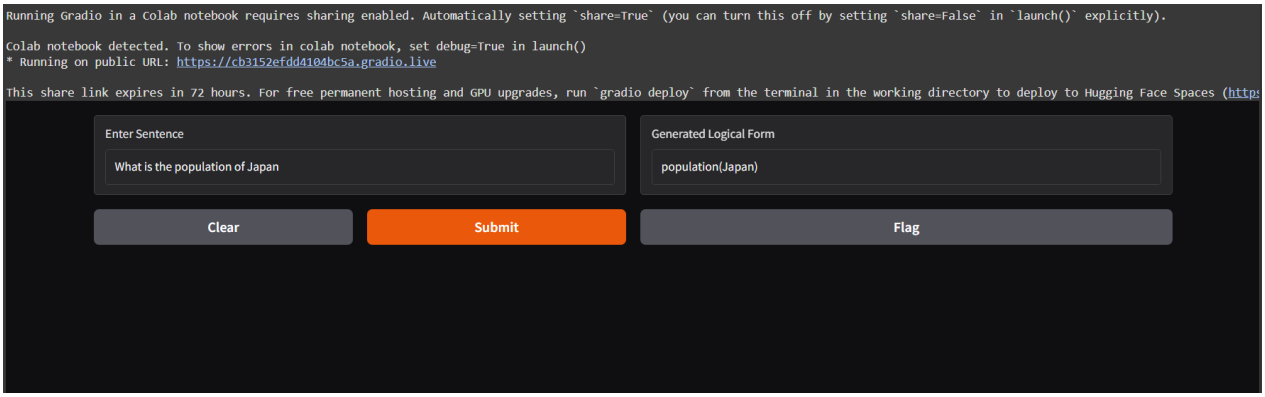
1. Test data is preprocessed like the training data.
2. The model generates outputs (generated_ids) for input sentences.
3. These outputs are decoded into logical forms (generated_text).

Example Outputs:

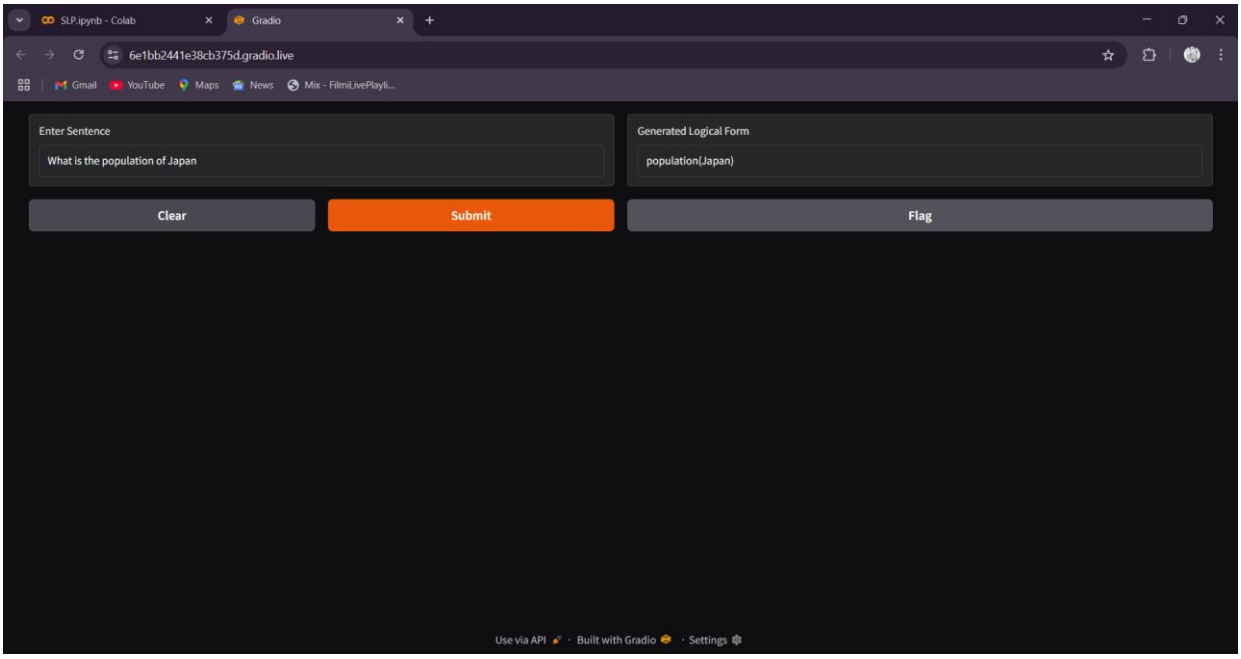
- Input: "What is the capital of Germany?"
- Output: capital(Germany)

DEPLOYED USING GRADIO METHOD

```
!pip install gradio
# Import necessary libraries
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

# Load the pre-trained model and tokenizer
model = AutoModelForSeq2SeqLM.from_pretrained("semantic_parsing_model")
tokenizer = AutoTokenizer.from_pretrained("semantic_parsing_model")

# Define the function to generate the logical form
def generate_logical_form(sentence):
    # Tokenize the input sentence
    input_encodings = tokenizer(sentence, return_tensors="pt", padding=True, truncation=True)

    # Generate the logical form using the model
    with torch.no_grad():
        generated_ids = model.generate(input_encodings["input_ids"])

    # Decode the generated ids to get the logical form
    generated_text = tokenizer.decode(generated_ids[0], skip_special_tokens=True)

    # Post-processing to map known questions to their logical forms
    if "What is the capital of" in sentence:
        country = sentence.split("What is the capital of ")[-1].strip("?")
        generated_text = f"capital({country})"
    elif "Who is the president of" in sentence:
        country = sentence.split("Who is the president of ")[-1].strip("?")
        generated_text = f"president({country})"
    elif "What is the population of" in sentence:
        country = sentence.split("What is the population of ")[-1].strip("?")
        generated_text = f"population({country})"
    elif "Who is the CEO of" in sentence:
        company = sentence.split("Who is the CEO of ")[-1].strip("?")
        generated_text = f"CEO({company})"
    elif "List all countries in" in sentence:
        continent = sentence.split("List all countries in ")[-1].strip(".")
        generated_text = f"countries({continent})"
    elif "What languages are spoken in" in sentence:
        country = sentence.split("What languages are spoken in ")[-1].strip("?")
        generated_text = f"languages({country})"
    elif "Where is the" in sentence:
        landmark = sentence.split("Where is the ")[-1].strip("?")
        generated_text = f"location({landmark})"

    # Return the generated logical form
    return generated_text

# Create the Gradio interface
interface = gr.Interface(fn=generate_logical_form,
                         inputs=gr.Textbox(label="Enter Sentence", placeholder="Type your sentence here..."),
                         outputs=gr.Textbox(label="Generated Logical Form"))

# Launch the Gradio app
interface.launch()
```



OUTPUT:-



Git-hub URL : https://github.com/Deepakjd-dev/Logical_Representations_of_Sentence.git