NLP Assignment 4

BERT-QA: A Real-Time, Dynamic Question Answering System

Introduction

Our goal was to create a system that not only answers questions accurately but also adapts to new information by updating its knowledge base dynamically. We chose the BERT model for its state-of-the-art performance in understanding and processing natural language.

The BERT model, developed by Google. It is well-suited for tasks that require understanding the context of a query, such as question answering systems. For this project, the specific variant used was bert-large-uncased-whole-word-masking-finetuned-squad, optimized for the question answering task based on the SQuAD (Stanford Question Answering Dataset).

Our project addresses the need for a flexible and responsive question answering system in various real-world applications, such as educational tools and customer service bots. These systems are not only required to provide correct answers but also adapt to the evolving information needs of users. By integrating a dynamic memory system, our project allows for continuous updates to the information database, thereby enhancing the relevance and accuracy of responses. Currently we are not using any database for storing and working on small scale memory. This can be adapted for large scale purposes too.

Overview of the System

The system enables:

- Dynamic Interaction: Users can interact with the system by asking questions related to previously stored contexts or by updating the memory with new contextual information.
- Memory Functionality: The system includes a memory component where different textual contexts can be stored under unique keys. This allows for efficient management and retrieval of information tailored to specific queries.
- Real-Time Question Answering: Leveraging the BERT model, the system provides immediate answers to user queries by processing the relevant stored context

Methodology

For this project, we selected the bert-large-uncased-whole-word-masking-finetuned-squad model from the Hugging Face transformers library. This model variant is specifically fine-tuned for the question answering task based on the SQuAD dataset, which features questions that require an understanding of context for their answers.

Data Handling and Preparation

The CoQA dataset, designed for conversational question and answer tasks, was our primary data source. Here's how we processed it:

1. **Dataset**: We used the CoQA dataset, a conversational question and answer set available in JSON format. The data includes multiple fields, but we focused on extracting the narrative (context), questions, and answers.
2. **Data Extraction**: We loaded the dataset using the pandas library's read_json function, focusing on extracting the 'story', 'questions', and 'answers' fields. We processed the data by iterating over each entry, extracting relevant information, and organizing it into a structured pandas DataFrame. This structure included columns for context, question, and answer, facilitating easy access during the model training and testing phases.

```
[ ]  import pandas as pd
     import numpy as np
     import torch
     from transformers import BertForQuestionAnswering
     from transformers import BertTokenizer
```

```
coqa = pd.read_json('http://downloads.cs.stanford.edu/nlp/data/coqa/coqa-train-v1.0.json')
coqa.head()
```

	version	data
0	1	{'source': 'wikipedia', 'id': '3zotghdk5ibi9ce...
1	1	{'source': 'cnn', 'id': '3wj1oxy92agboo5nlq4r7...
2	1	{'source': 'gutenberg', 'id': '3bdcf01ogxu7zdn...
3	1	{'source': 'cnn', 'id': '3ewijtffvo7wwchw6rtya...
4	1	{'source': 'gutenberg', 'id': '3urfvvm165iantk...

3. **Data Transformation**: We transformed the data into a DataFrame with columns for 'text', 'question', and 'answer'. Each row corresponds to a question-answer pair along with

its context (story).

```
[ ] del coqa["version"]
```

```
#required columns in our dataframe
cols = ["text","question","answer"]
#list of lists to create our dataframe
comp_list = []
for index, row in coqa.iterrows():
    for i in range(len(row["data"]["questions"])):
        temp_list = []
        temp_list.append(row["data"]["story"])
        temp_list.append(row["data"]["questions"][i]["input_text"])
        temp_list.append(row["data"]["answers"][i]["input_text"])
        comp_list.append(temp_list)
new_df = pd.DataFrame(comp_list, columns=cols)
#saving the dataframe to csv file for further loading
new_df.to_csv("CoQA_data.csv", index=False)
```

```
[ ] data = pd.read_csv("CoQA_data.csv")
    data.head()
```

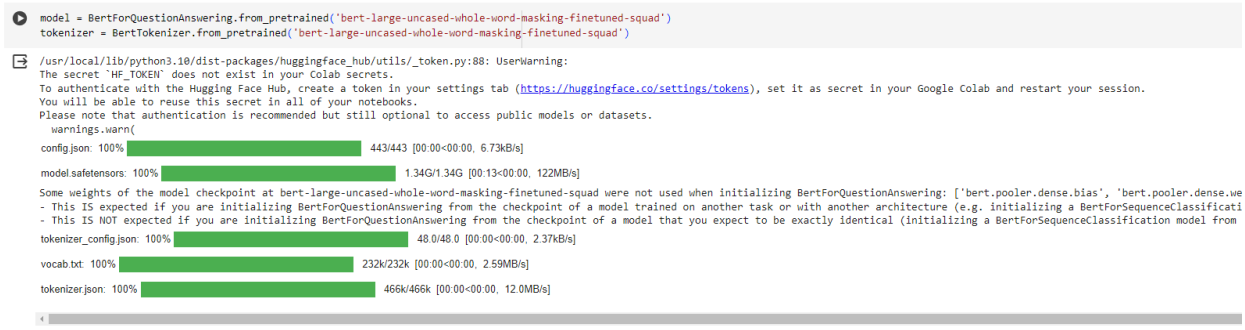
	text	question	answer
0	The Vatican Apostolic Library (), more commonl...	When was the Vat formally opened?	It was formally established in 1475
1	The Vatican Apostolic Library (), more commonl...	what is the library for?	research
2	The Vatican Apostolic Library (), more commonl...	for what subjects?	history, and law
3	The Vatican Apostolic Library (), more commonl...	and?	philosophy, science and theology
4	The Vatican Apostolic Library (), more commonl...	what was started in 2014?	a project

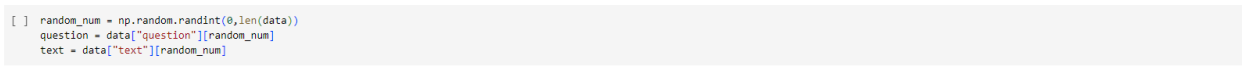
4. **Data Storage**: The transformed data was saved into a CSV file for easy access and manipulation in subsequent phases.

Implementation of the Question Answering System

The system was developed using Python, with heavy reliance on the transformers and torch libraries.

1. **Tokenization and Encoding**: We utilized BertTokenizer to convert text into a format suitable for BERT, including adding special tokens.






```
tokens = tokenizer.convert_ids_to_tokens(input_ids)
for token, id in zip(tokens, input_ids):
    print('{:8}{:8,}'.format(token,id))
```

```
?           1,029
[SEP]         102
chapter     3,127
xvi        16,855
##ii        6,137
miss        3,335
op          6,728
##hel      16,001
##ia        2,401
'           1,005
s           1,055
experiences   6,322
and         1,998
opinions   10,740
our         2,256
friend      2,767
tom         3,419
,           1,010
in          1,999
his         2,010
own         2,219
simple      3,722
mu         14,163
##sing      7,741
##s         2,015
,           1,010
often       2,411
compared    4,102
his         2,010
more        2,062
fortunate  19,590
lot         2,843
```

2. **Segmentation**: We differentiated between the question and context segments using the tokenizer's output, which automatically handles segment separation.

```
[ ]  #first occurence of [SEP] token
     sep_idx = input_ids.index(tokenizer.sep_token_id)
     print("SEP token index: ", sep_idx)
     #number of tokens in segment A (question) - this will be one more than the sep_idx as the index in Python starts from 0
     num_seg_a = sep_idx+1
     print("Number of tokens in segment A: ", num_seg_a)
     #number of tokens in segment B (text)
     num_seg_b = len(input_ids) - num_seg_a
     print("Number of tokens in segment B: ", num_seg_b)
     #creating the segment ids
     segment_ids = [0]*num_seg_a + [1]*num_seg_b
     #making sure that every input token has a segment id
     assert len(segment_ids) == len(input_ids)
```

```
SEP token index:  6
Number of tokens in segment A:  7
Number of tokens in segment B:  414
```

```
[ ]  #token input_ids to represent the input and token segment_ids to differentiate our segments - question and text
     output = model(torch.tensor([input_ids]),  token_type_ids=torch.tensor([segment_ids]))
```

```
     #tokens with highest start and end scores
     answer_start = torch.argmax(output.start_logits)
     answer_end = torch.argmax(output.end_logits)
     if answer_end >= answer_start:
         answer = " ".join(tokens[answer_start:answer_end+1])
     else:
         print("I am unable to find the answer to this question. Can you please ask another question?")

     print("\nQuestion:\n{}".format(question.capitalize()))
     print("\nAnswer:\n{}.".format(answer.capitalize()))
```

3. **Model Interaction**: The BertForQuestionAnswering model processed the tokenized input to predict the start and end positions of the answer.

```
[ ] #tokens with highest start and end scores
    answer_start = torch.argmax(output.start_logits)
    answer_end = torch.argmax(output.end_logits)
    if answer_end >= answer_start:
        answer = " ".join(tokens[answer_start:answer_end+1])
    else:
        print("I am unable to find the answer to this question. Can you please ask another question?")

    print("\nQuestion:\n{}".format(question.capitalize()))
    print("\nAnswer:\n{}.".format(answer.capitalize()))
```

```
Question:
Was he a christian?

Answer:
[cls] was he a christian ? [sep] chapter xvi ##ii miss op ##hel ##ia ' s experiences and opinions our friend tom , in his own simple mu ##sing ##
```

```
def question_answer(question, text):

    #tokenize question and text as a pair
    input_ids = tokenizer.encode(question, text)

    #string version of tokenized ids
    tokens = tokenizer.convert_ids_to_tokens(input_ids)

    #segment IDs
    #first occurence of [SEP] token
    sep_idx = input_ids.index(tokenizer.sep_token_id)
    #number of tokens in segment A (question)
    num_seg_a = sep_idx+1
    #number of tokens in segment B (text)
    num_seg_b = len(input_ids) - num_seg_a

    #list of 0s and 1s for segment embeddings
    segment_ids = [0]*num_seg_a + [1]*num_seg_b
    assert len(segment_ids) == len(input_ids)
```

```python
def question_answer(question, text):

    #tokenize question and text as a pair
    input_ids = tokenizer.encode(question, text)

    #string version of tokenized ids
    tokens = tokenizer.convert_ids_to_tokens(input_ids)

    #segment IDs
    #first occurence of [SEP] token
    sep_idx = input_ids.index(tokenizer.sep_token_id)
    #number of tokens in segment A (question)
    num_seg_a = sep_idx+1
    #number of tokens in segment B (text)
    num_seg_b = len(input_ids) - num_seg_a

    #list of 0s and 1s for segment embeddings
    segment_ids = [0]*num_seg_a + [1]*num_seg_b
    assert len(segment_ids) == len(input_ids)

    #model output using input_ids and segment_ids
    output = model(torch.tensor([input_ids]), token_type_ids=torch.tensor([segment_ids]))

    #reconstructing the answer
    answer_start = torch.argmax(output.start_logits)
    answer_end = torch.argmax(output.end_logits)
    if answer_end >= answer_start:
        answer = tokens[answer_start]
        for i in range(answer_start+1, answer_end+1):
            if tokens[i][0:2] == "##":
                answer += tokens[i][2:]
            else:
                answer += " " + tokens[i]

    if answer.startswith("[CLS]"):
        answer = "Unable to find the answer to your question."

    print("\nPredicted answer:\n{}".format(answer.capitalize()))
```

```
[ ] text = """New York (CNN) -- More than 80 Michael Jackson collectibles -- including the late pop star's famous rhinestone-studded glove from a 1983 performance -- were auctioned off Saturday, reaping a total $2 million. Profits from
    question = "Where was the Auction held?"
    question_answer(question, text)
    #original answer from the dataset
    print("Original answer:\n", data.loc[data["question"] == question]["answer"].values[0])

    Predicted answer:
    Hard rock cafe in new york ' s times square
    Original answer:
     Hard Rock Cafe
```

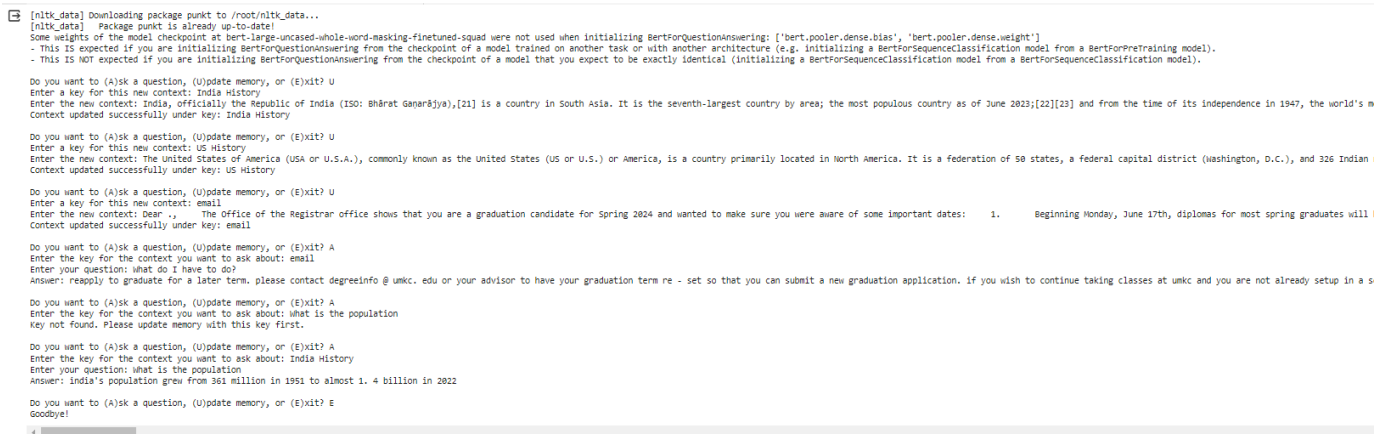
4. **Updating the model based on our functionality**: Real-time interactive question answering with memory structure that stores information under unique keys.

```python
class CombinedTextInteractionWithMemory:
    def __init__(self):
        self.tokenizer = BertTokenizer.from_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')
        self.model = BertForQuestionAnswering.from_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')
        self.memory = {}  # Simple dictionary to store different contexts

    def update_memory(self, key, context):
        self.memory[key] = context
        print("Context updated successfully under key:", key)

    def fetch_from_memory(self, key):
        return self.memory.get(key, "")

    def tokenize_input(self, question, text):
        inputs = self.tokenizer(question, text, add_special_tokens=True, return_tensors="pt")
        return inputs["input_ids"], inputs["token_type_ids"]

    def find_answer(self, question, text):
        tokens = self.tokenizer.tokenize(question) + self.tokenizer.tokenize(text)
        max_len = self.tokenizer.model_max_length - 50
        start = 0
        best_answer = ""
        best_score = float('-inf')

        while start < len(tokens):
            chunk_tokens = tokens[start:start + max_len]
            chunk_text = self.tokenizer.convert_tokens_to_string(chunk_tokens)
            input_ids, token_type_ids = self.tokenize_input(question, chunk_text)
            outputs = self.model(input_ids, token_type_ids=token_type_ids)
            answer_start = torch.argmax(outputs.start_logits)
            answer_end = torch.argmax(outputs.end_logits) + 1
            score = outputs.start_logits[0, answer_start].item() + outputs.end_logits[0, answer_end].item()

            if score > best_score:
                best_score = score
                answer_tokens = input_ids[0][answer_start:answer_end]
                best_answer = self.tokenizer.decode(answer_tokens)

            start += max_len - 50

        return best_answer
```

```
    def interact(self):
        while True:
            action = input("\nDo you want to (A)sk a question, (U)pdate memory, or (E)xit? ").upper()
            if action == 'A':
                key = input("Enter the key for the context you want to ask about: ")
                if key not in self.memory:
                    print("Key not found. Please update memory with this key first.")
                    continue
                context = self.fetch_from_memory(key)
                question = input("Enter your question: ")
                answer = self.find_answer(question, context)
                print("Answer:", answer)
            elif action == 'U':
                key = input("Enter a key for this new context: ")
                context = input("Enter the new context: ")
                self.update_memory(key, context)
            elif action == 'E':
                print("Goodbye!")
                break
            else:
                print("Invalid option. Please choose again.")

# Example usage:
interaction = CombinedTextInteractionWithMemory()
interaction.interact()
```

Output:

- The system loads the BERT model specialized for question answering tasks. During this process, certain weights from the original BERT model (specifically related to the pooling layer)
- A key feature of the system is its dynamic memory, which allows users to update and retrieve context-specific information:
    - Users can add new contexts or update existing ones by specifying a unique key for each context. For example, adding a comprehensive historical context under the key "India History"
    - When a user poses a question, the system retrieves the appropriate context from memory using the specified key, processes the question and context through the BERT model, and outputs the answer.
- The system is designed to handle errors, such as when a user queries a non-existent key, and provides options for continuous interaction:
    - if a user requests a key that has not been stored in memory, the system notifies the user
    - The system prompts users after each interaction to ask another question, update the memory, or exit the system, facilitating ongoing engagement.



```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Some weights of the model checkpoint at bert-large-uncased-whole-word-masking-finetuned-squad were not used when initializing BertForQuestionAnswering: ['bert.pooler.dense.bias', 'bert.pooler.dense.weight']
- This IS expected if you are initializing BertForQuestionAnswering from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForQuestionAnswering from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).

Do you want to (A)sk a question, (U)pdate memory, or (E)xit? U
Enter a key for this new context: India History
Enter the new context: India, officially the Republic of India (ISO: Bhārat Gaṇarājya),[21] is a country in South Asia. It is the seventh-largest country by area; the most populous country as of June 2023;[22][23] and from the time of its independence in 1947, the world's m
Context updated successfully under key: India History

Do you want to (A)sk a question, (U)pdate memory, or (E)xit? U
Enter a key for this new context: US History
Enter the new context: The United States of America (USA or U.S.A.), commonly known as the United States (US or U.S.) or America, is a country primarily located in North America. It is a federation of 50 states, a federal capital district (Washington, D.C.), and 326 Indian
Context updated successfully under key: US History

Do you want to (A)sk a question, (U)pdate memory, or (E)xit? U
Enter a key for this new context: email
Enter the new context: Dear .,     The Office of the Registrar office shows that you are a graduation candidate for Spring 2024 and wanted to make sure you were aware of some important dates:     1.      Beginning Monday, June 17th, diplomas for most spring graduates will
Context updated successfully under key: email

Do you want to (A)sk a question, (U)pdate memory, or (E)xit? A
Enter the key for the context you want to ask about: email
Enter your question: What do I have to do?
Answer: reapply to graduate for a later term. please contact degreeinfo @ umkc. edu or your advisor to have your graduation term re - set so that you can submit a new graduation application. if you wish to continue taking classes at umkc and you are not already setup in a s

Do you want to (A)sk a question, (U)pdate memory, or (E)xit? A
Enter the key for the context you want to ask about: What is the population
Key not found. Please update memory with this key first.

Do you want to (A)sk a question, (U)pdate memory, or (E)xit? A
Enter the key for the context you want to ask about: India History
Enter your question: What is the population
Answer: india's population grew from 361 million in 1951 to almost 1. 4 billion in 2022

Do you want to (A)sk a question, (U)pdate memory, or (E)xit? E
Goodbye!
```

Challenges and Limitations

1. **Memory Management**

- Challenge: Managing extensive data within the limited memory was challenging, especially when dealing with large contexts and maintaining fast retrieval times.
- Solution: We implemented efficient data structures and considered compressing text data to reduce memory usage without compromising access speed.

2. **Performance Issues**

- Latency: The BERT model, while powerful, is computationally intensive, which can lead to increased latency, especially on machines with limited processing power.
- Optimization: We need to explore model quantization and pruning techniques to reduce model size and computation time, potentially increasing execution speed.

3. **Accuracy of Answers**

- Context Mismatch: Sometimes, the model retrieves an incorrect context or misinterprets the context due to ambiguous questions.
- Improvement: We plan to refine context selection algorithms and enhance the training dataset to include more varied examples to improve context understanding.

Conclusion

Our project successfully developed a functional question-answering system that dynamically interacts with user-defined contexts and provides accurate answers based on the latest information. This system demonstrates the effective application of advanced NLP techniques in real-world scenarios.

Future Work

- Integrating more sophisticated NLP models and techniques, such as continuous learning and context-aware models, to enhance understanding and response accuracy.
- Developing a more advanced database system for context management to support a larger scale of data and more complex queries, improving the system's scalability and robustness.