23CSAI05H

Natural Language Processing

NLP CW2: Question Answering System.

Group 13

ID:	Name:	Contribution:
205001	Soha Ashraf	BERT
211320	Mohammed Amein	T5
217764	Mohammed Essam	RoBERTa
213109	Omar Tarek	GPT

Introduction:

Question Answering System (QA system) is a task that involves identifying the answers to the questions from a large corpus of text. It involves various information such as information retrieval, text understanding, and extracting the information to answer the questions

accurately based on the meaning of the input and the context. There are two types of QA system which is extracting answer from the input / given context and generating an answer from the context that answers the questions correctly. The QA system will be implemented using t5, Bert, and GPT transformers using SQuAD dataset.[1][2].

In QA system, the trained model should understand the questions' meaning and the to differentiate between words. Also retrieving the needed information from large data. For accurate answers, the model should understand how to represent the answers data by using word tokenization and analyze the data. In the final step, the model should validate the answer accuracy.[7].

Dataset:

Stanford Question Answering Dataset (SQuAD) is a dataset that is used in training, testing, validating, and evaluating questions answering system. It is a collection of Wikipedia articles that consists of pairs of questions answer. These questions answers are extracted from these articles. The dataset has two versions which are SQuAD v.1.1 and SQuAD v.2.0. SQuAD 1.1 encompasses 107,785 question-answer pairs across 536 articles. The most recent iteration, SQuAD2.0 (referred to as open-domain SQuAD or SQuAD-Open), amalgamates the 100,000 questions from SQuAD1.1 with an additional 50,000 unanswerable questions formulated adversarially by crowdworkers in formats resembling the answerable ones.[3][4].

- Sample of the dataset:

	context	question	answer
0	The Normans (Norman: Nourmands; French: Norman...	In what country is Normandy located?	France
1	The Normans (Norman: Nourmands; French: Norman...	When were the Normans in Normandy?	10th and 11th centuries
2	The Normans (Norman: Nourmands; French: Norman...	From which countries did the Norse originate?	Denmark, Iceland and Norway
3	The Normans (Norman: Nourmands; French: Norman...	Who was the Norse leader?	Rollo
4	The Normans (Norman: Nourmands; French: Norman...	What century did the Normans first gain their ...	10th century

Data Preprocessing:

The dataset has 86821 passages/ texts, queries, and answers from training data. For validation data, it has 20302 passages/ texts, queries, and answers.

```
    print(len(train_texts))
    print(len(train_queries))
    print(len(train_answers))
[6]                                                                    Python

86821
86821
86821

    print("Passage: ",train_texts[0])
    print("Query: ",train_queries[0])
    print("Answer: ",train_answers[0])
[7]                                                                    Python

Passage:  Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songw
Query:  When did Beyonce start becoming popular?
Answer:  {'text': 'in the late 1990s', 'answer_start': 269}

    print(len(val_texts))
    print(len(val_queries))
    print(len(val_answers))
[8]                                                                    Python

20302
20302
20302
```

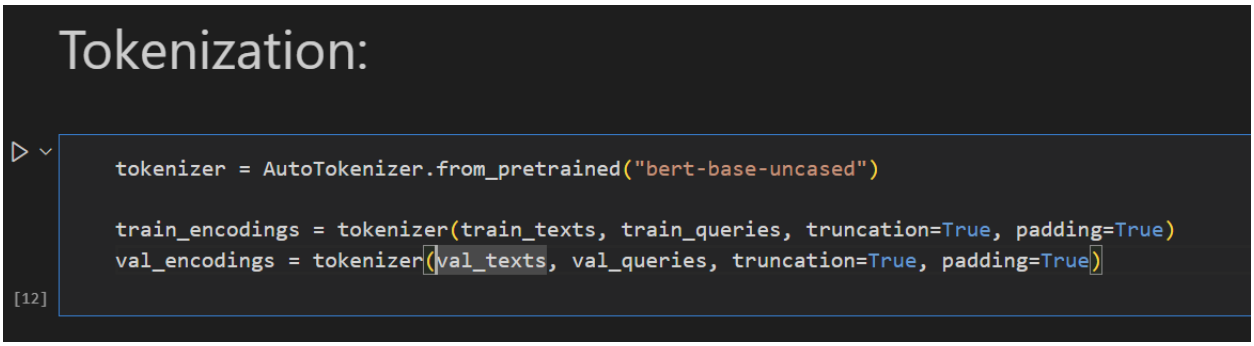
The Used Models:

1. BERT (Bidirectional Encoder Representations from Transformers).
2. T5 (Text-To-Text Transfer Transformer).
3. GPT (Generative Pre-trained Transformer).
4. XLNet (Generalized Autoregressive Pretraining for Language Understanding).
5. RoBERTa (Robustly optimized BERT approach).

BERT Model:

BERT is a transformer model based on transformer encoder that was developed by Google. It is a machine learning framework for NLP. It aims to enhance and improve contextual comprehension of unannotated text in various tasks by training for text prediction. It transforms words into numbers. This model is based on transformer that uses attention to get the relationship between sentence words without depending on sequential processing.[5][6].

The tokenization for BERT model:



As BERT model should have start and end position characters for both training and valid data:

```
for answer, text in zip(train_answers, train_texts):
    real_answer = answer['text']
    start_idx = answer['answer_start']
    end_idx = start_idx + len(real_answer)

    if text[start_idx:end_idx] == real_answer:
        answer['answer_end'] = end_idx
    elif text[start_idx-1:end_idx-1] == real_answer:
        answer['answer_start'] = start_idx - 1
        answer['answer_end'] = end_idx - 1
    elif text[start_idx-2:end_idx-2] == real_answer:
        answer['answer_start'] = start_idx - 2
        answer['answer_end'] = end_idx - 2
```

[10]

```
for answer, text in zip(val_answers, val_texts):
    real_answer = answer['text']
    start_idx = answer['answer_start']
    end_idx = start_idx + len(real_answer)

    if text[start_idx:end_idx] == real_answer:
        answer['answer_end'] = end_idx
    elif text[start_idx-1:end_idx-1] == real_answer:
        answer['answer_start'] = start_idx - 1
        answer['answer_end'] = end_idx - 1
    elif text[start_idx-2:end_idx-2] == real_answer:
        answer['answer_start'] = start_idx - 2
        answer['answer_end'] = end_idx - 2
```

[11]

For Bert model building, BertForQuestionAnswering is selected from transformers library. The optimizer used is “AdamW” that can deal with weight decay.

```
model = BertForQuestionAnswering.from_pretrained('bert-base-uncased').to(device)
# OPTIMIZER = Adam(MODEL.parameters(), lr=0.00001)
# DEVICE = "cuda:0"
optim = AdamW(model.parameters(), lr=5e-5)
Q_LEN = 256   # Question Length
T_LEN = 32    # Target Length
BATCH_SIZE = 4
epochs = 4
```

Model training and evaluation:

```
whole_train_eval_time = time.time()

train_losses = []
val_losses = []

print_every = 50

for epoch in range(epochs):
  epoch_time = time.time()

  model.train()

  loss_of_epoch = 0

  print("Train:")

  for batch_idx,batch in enumerate(train_loader):

    optim.zero_grad()

    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    start_positions = batch['start_positions'].to(device)
    end_positions = batch['end_positions'].to(device)

    outputs = model(input_ids, attention_mask=attention_mask, start_positions=start_positions, end_positions=end_positions)
    loss = outputs[0]
    loss.backward()

    optim.step()

    loss_of_epoch += loss.item()

    if (batch_idx+1) % print_every == 0:
      print("Batch {:} / {:}".format(batch_idx+1,len(train_loader)),"\nLoss:", round(loss.item(),1),"\n")

  loss_of_epoch /= len(train_loader)
  train_losses.append(loss_of_epoch)

  model.eval()

  print("Evaluate:")
```

```
  print("Evaluate:")

  loss_of_epoch = 0

  for batch_idx,batch in enumerate(val_loader):

    with torch.no_grad():

      input_ids = batch['input_ids'].to(device)
      attention_mask = batch['attention_mask'].to(device)
      start_positions = batch['start_positions'].to(device)
      end_positions = batch['end_positions'].to(device)

      outputs = model(input_ids, attention_mask=attention_mask, start_positions=start_positions, end_positions=end_positions)
      loss = outputs[0]
      loss_of_epoch += loss.item()

    if (batch_idx+1) % print_every == 0:
      print("Batch {:} / {:}".format(batch_idx+1,len(val_loader)),"\nLoss:", round(loss.item(),1),"\n")

  loss_of_epoch /= len(val_loader)
  val_losses.append(loss_of_epoch)

  print("\n-------Epoch ", epoch+1,
        "-------"
        "\nTraining Loss:", train_losses[-1],
        "\nValidation Loss:", val_losses[-1],
        "\nTime: ",(time.time() - epoch_time),
        "\n------------------------",
        "\n\n")

print("Total training and evaluation time: ", (time.time() - whole_train_eval_time))
```

T5 Model:

GPT Model:

GPT (Generative Pre-trained Transformer) is a state-of-the-art language model that uses the transformer architecture to generate coherent and contextually relevant text. Trained on a vast corpus of diverse text from the internet, GPT has the ability to understand and generate human-like language. It excels in a wide range of natural language processing tasks, including text completion, summarization, translation, and question answering. By leveraging its deep understanding of language patterns and semantics, GPT can generate meaningful and coherent responses to user queries, making it a versatile tool for various applications in the field of natural language processing and artificial intelligence.

Tokenizer:

```
def load_data_collator(tokenizer, mlm = False):
    data_collator = DataCollatorForLanguageModeling(
        tokenizer=tokenizer,
        mlm=mlm,
    )
    return data_collator
```

```
def train(train_file_path,model_name,
          output_dir,
          overwrite_output_dir,
          per_device_train_batch_size,
          num_train_epochs,
          save_steps):
  tokenizer = GPT2Tokenizer.from_pretrained(model_name)
  train_dataset = load_dataset(train_file_path, tokenizer)
  data_collator = load_data_collator(tokenizer)

  tokenizer.save_pretrained(output_dir)

  model = GPT2LMHeadModel.from_pretrained(model_name)

  model.save_pretrained(output_dir)

  training_args = TrainingArguments(
          output_dir=output_dir,
          overwrite_output_dir=overwrite_output_dir,
          per_device_train_batch_size=per_device_train_batch_size,
          num_train_epochs=num_train_epochs,
      )

  trainer = Trainer(
          model=model,
          args=training_args,
          data_collator=data_collator,
          train_dataset=train_dataset,
  )

  trainer.train()
  trainer.save_model()
```

```python
def load_model(model_path):
    model = GPT2LMHeadModel.from_pretrained(model_path)
    return model


def load_tokenizer(tokenizer_path):
    tokenizer = GPT2Tokenizer.from_pretrained(tokenizer_path)
    return tokenizer

def generate_text(model_path, sequence, max_length):

    model = load_model(model_path)
    tokenizer = load_tokenizer(model_path)
    ids = tokenizer.encode(f'{sequence}', return_tensors='pt')
    final_outputs = model.generate(
        ids,
        do_sample=True,
        max_length=max_length,
        pad_token_id=model.config.eos_token_id,
        top_k=50,
        top_p=0.95,
    )
    print(tokenizer.decode(final_outputs[0], skip_special_tokens=True))
```

BERT Model:

OverView

This report is on the development of a question-answering system using the RoBERTa transformer model pre-trained on the SQuAD dataset.

Model Selection

Robustly Optimized BERT Pretraining (RoBERTa) the model is optimized on the selected BERT (Bidirectional Encoder Representations from Transformers). I chose RoBERTa because of the state-of-the-art performance in a number of NLP benchmarks. In this work, I have used the model in its “base” configuration, which is the best with a good tradeoff between computational efficiency and performance.

Implementation Details

Data Loading and Preprocessing

I loaded the data with the Hugging Face datasets library. Each example is a pair of context-question with the corresponding answer. Textual data for both contexts and questions were tokenized for preprocessing with RoBERTa's tokenizer. In this process, the text gets turned into a format suitable for the model to process. It would involve the tokenization of the text into tokens and the mapping of these tokens to respective indices in the tokenizer vocabulary.

Training

The model was trained with a batch size of 16, which is mainly because of computational requirements but also gives a good trade-off between memory consumption and computational efficiency in regards to model performance. The model was trained using the AdamW optimizer with a learning rate of 5×10^5 over 3 epochs. This set-up was chosen as it is customary in the field to balance training duration and convergence rates.

Evaluation

the model's performance was evaluated on the validation set using the following metrics:

- Accuracy, Precision, Recall, F1 Score: it measures how answers from the model is like the actual answers. **EM** calculates the concurrence proportion for the prediction and ground truth answers, while **F1** measures the average overlap of prediction and ground truth answers. Findings and results
- Exact Match (EM) and F1 Score: Specific to SQuAD, these metrics measure how well the model's answers match the actual answers: EM measures the percentage of predictions that match any one of the ground truth answers exactly, and F1 measures the average overlap between the prediction and ground truth answers.

References:

[1] G. Lokare, “Question Answering with Hugging Face Transformers: A Beginner’s Guide,” Medium, Feb. 12, 2023. [Online]. Available: https://medium.com/@lokaregns/question-answering-with-hugging-face-transformers-a-beginners-guide-487ae1a91b9a

[2] “Question answering.” https://huggingface.co/docs/transformers/tasks/question_answering

[3] “What is SQuAD (Stanford Question Answering Dataset)?” https://h2o.ai/wiki/squad/#:~:text=SQuAD%2C%20short%20for%20Stanford%20Question,text%20within%20the%20corresponding%20article.

[4] “The Stanford question answering Dataset.” https://rajpurkar.github.io/SQuAD-explorer/

[5] C. Hashemi-Pour and B. Lutkevich, “BERT language model,” Enterprise AI, Feb. 15, 2024. https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model#:~:text=What%20is%20BERT%3F,surrounding%20text%20to%20establish%20context.

[6] “What is BERT and how is it Used in AI?” https://h2o.ai/wiki/bert/

[7] W&B, “Weights & biases,” W&B, May 03, 2024. https://wandb.ai/mostafaibrahim17/ml-articles/reports/The-Answer-Key-Unlocking-the-Potential-of-Question-Answering-With-NLP--VmlldzozNTcxMDE3#:~:text=Question%20Answering%20in%20NLP%2C%20is,and%20present%20an%20accurate%20answer.