به نام خدا

امیرمحمد کمیجانی ۹۹۵۲۲۰۳۲

تمرین تئوری + گزارش کد تمرین سری ۵

۱)

نوع close domain : در این نوع سیستم QA ما برای پاسخگویی به دامنه محدود و خاصی تعریف شده است.یعنی مدلی که تعریف و آنرا آموزش داده ایم و از آن نتیجه میخواهیم دامنه مشخصی از دیتا را دریافت کرده و میتواند به سوالات مربوط به آن حوزه که آموزش دیده پاسخ دهد. مثلا اگر دیتای مربوط به پزشکی را به آن میدهیم سوالات مربوط به حوزه پزشکی میتوانیم از آن بپرسیم و در بقیه حوزه ها پاسخگویی ندارد.برای تهیه دیتای این قسمت از پایگاه های اطلاعاتی نه چندان بزرگ و حجیم استفاده میکنیم.

نوع open domain : این نوع توانایی پاسخگویی به رِنج بیشتری از حوزه های مختلف را دارد و دیتایی که ما در اختیار مدل قرار دادیم تا طبق آن آموزش ببیند فقط مربوط به یک حوزه خاص نیست و میتواند از حوزه های مختلف باشد و همچنین هنگامی که از مدل سوال میپرسیم میتواند در حوزه های مختلف به ما پاسخگویی داشته باشد.به طور مثال مدل را در حوزه های پزشکی ورزشی فیلم و سریال و علمی و اقتصادی و ... آموزش میدهیم و محدودیتی برای طیف موضوعات قرار نمیدهیم. برای دیتای این قسمت از پایگاه های اطلاعاتی بزرگ مثل ویکی پدیا استفاده میکنیم.

تفاوت :

۱. **دامنه :** همانطور که پیش تر توضیح دادیم نوع open domain به طیف بیشتری از موضوعات مختلف پاسخ دهند اما نوع close domain فقط در حوزه ای که برای آن طراحی شده اند میتوانند پاسخگویی داشته باشند.

۲. **پایگاه دانش(دیتا)** : نوع open domain از پایگاه های اطلاعاتی بسیار بزرگی استفاده میکند که تنوع اطلاعات در آن بیشتر است اما نوع close از دامنه محدودتر و تخصصی تر راجع به موضوع خودش طراحی شده است.
۳. **دقت** : در نوع close ما از دقت بیشتری برخوردار هستیم زیرا آن مدل و سیستم QA برای موضوع مشخصی طراحی شده و از اطلاعات تخصصی تری بهره مند شده و جزئیات بیشتری را شامل شده است در صورتی که در open از اطلاعات دیگری هم برخوردار هستیم و ممکن است اطلاعات را به صورت کلی دریافت کنیم و خیلی با جزئیات و ریز به ریز دریافت نکرده باشیم.
۴. **کاربرد** : نوع open برای سوالات عمومی تر و نوع close برای سوالات تخصصی تر مناسب هستند.

۲)

تعریف machine reading comprehension(MRC) : تسکی است که در آن میخواهیم سیستمی را طراحی کنیم تا یک متن را مانند انسان بخواند ، درک کند و بفهمد.

ارتباط آن نیز با QA مشخص است.در QA کاربر یک سوال میپرسد و سیستم ما باید جواب مناسب را به عنوان خروجی برگرداند.با استفاده از MRC سیستم ما میتواند سوالی که از آن پرسیده شده را بخواند و بفهمد و با توجه به فهمی که داشته است بتواند خروجی مناسب را بدهد. نحوه کار MRC به این صورت است که ابتدا متن سوال را میخواند و باید نوع سوال را درک کند مثلا سوال yes/no است یا wh question ؟

و سپس شروع به درک متن سوال و استخراج اطلاعات از آن سوال میکند و سپس شروع به تولید خروجی میکند.

۳)

سوالات factoid به این صورت هستند که به دنبال یک پاسخ کوتاه و دقیق بر اساس حقایق موجود هستیم.مثلا سوال پایتخت ایران کجاست ؟ یک جواب مشخص و کوتاه دارد و بر اساس حقایق و فکت هایی است که میدانیم.

سوالات non-factoid به این صورت نیستند و بیشتر سوالاتی هستند که به دنبال دلیل و استدلال و منطق برای پاسخگویی به آنها هستیم.سوالاتی که معمولا با "چرا" آغاز میشوند معمولا از این نوع هستند چون ما دنبال دلیل یک مسئله هستیم و جواب های ما هم طولانی تر هستند.

از دیگر تفاوت های این دو نوع سوال میتوان به این اشاره کرد سوالات factoid به نسبت ساده تر هستند و معمولا سیستم های QA ما به راحتی جواب های آنرا پیدا میکنند و معمولا هم از دقت بسیار بالایی برخوردار هستند اما در نوع non factiod چون به دنبال دلیل یک مسئله هستیم تولید جواب برای سیستم سخت تر است و به میزان پاسخ های سوالات factoid دقیق نیستند.

۴)

در RNN میدانیم با توجه به خاصیت بازگشتی ای که داشتند اطلاعات را از گذشته دریافت میکردند تا بتوانند برای پیش بینی استفاده کنند.اما چند محدودیت داشتیم که البته بعضی از انها در LSTM رفع شد ولی این بار برای مقایسه با transformer ها آنها را بیان میکنیم.

محدودیت اول RNN مسئله vanishing/exploding gradient بود که در transformer ها به خوبی این مسئله کنترل شد.این مشکل باعث میشد long-range dependeny ها به خوبی دریافت نشوند و معنا و مفهوم کل جمله با طولانی تر شدن آن به خوبی دریافت و درک نشود. در transformer ها این مشکل به خوبی کنترل شد.(**مزایا**)

یکی دیگر از مشکلاتی که در RNN ها داشتیم سختی parallelism میباشد.چون RNN ها بازگشتی هستند خیلی توانایی برای parallelism ندارند(ممکنه باعث کندی بشه). اما در

transformer ها این موضوع نیز کنترل شد و تمام ورودی ها به صورت موازی یا همان parallel هندل و process شدن.(**مزایا**)

یکی دیگر از توانایی های transformer ها که در RNN نداشتیم این بود که میتواند به کل متن نگاه کند و قسمت هایی که برای پیش بینی مهم تر هستند را شناسایی کند که به این مورد اصطلاحا interpretable attention نیز گفته میشود.(**مزایا**)

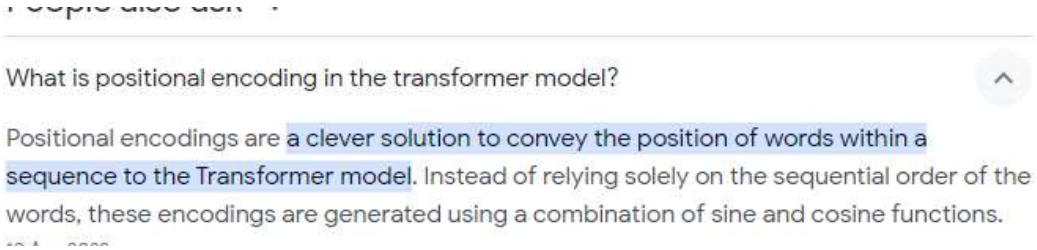
در transformer ها ما پیچیدگی بسیار بیشتری داریم و همچنین نیازمند حجم داده بسیار بیشتری نیز هستیم.(**معایب**)

۵)

Positional encoding برای این است که برای هر توکن یا کلمه یک موقعیت یا position مشخص کنیم.

در مدل‌های مبتنی برTransformer ، رمزگذاری موقعیتی تکنیکی است که به مدل کمک می‌کند تا ترتیب کلمات در یک جمله یا توکن‌ها در یک دنباله را درک کند .این امر به این دلیل مهم است که ترتیب کلمات در زبان‌های طبیعی اغلب معنی را تعیین می‌کند .به عنوان مثال، جمله "گربه موش را گرفت" معنای متفاوتی با جمله "موش گربه را گرفت" دارد.

مدل‌های Transformer از مکانیزم توجه(attention) برای مدل‌سازی روابط بین کلمات در یک جمله استفاده می‌کنند .با این حال، مکانیزم توجه به تنهایی نمی‌تواند ترتیب کلمات را درک کند .به همین دلیل، مدل‌های Transformer از رمزگذاری موقعیتی برای اضافه کردن اطلاعات مربوط به موقعیت هر کلمه در جمله به بردارهای ورودی استفاده می‌کنند.



۶)

Encoder-only : فقط مولفه encoder را دارند و به عنوان ورودی یک تکست میگیرند و در خروجی مثلا embeddings ها را برمیگردانند.(یا به صورت کلی ورودی را گرفته و به یک representation معنی دار تبدیل میکنند.)

Decoder-only : فقط مولفه Decoder را دارند و هنگامی که یک تکست را به عنوان ورودی دریافت کردند به عنوان خروجی کلمه یا توکن بعدی را به ما میدهند.

Encoder-Decoder : هر دو جزء را دارد و به این صورت است که ورودی به encoder داده میشود و به یک representation با طول ثابت تبدیل میشود و به decoder داده میشود و بعد decoder طبق دیتایی که از encoder گرفته خروجی مناسب را تولید میکند.

به جز نحوه کار، کاربرد های این سه در تسک های مختلف nlp نیز از تفاوت های آنها است.

به طور مثال encoder-only ها برای تسک های text classification , sentiment analysis , clustering و این دست مسائل به کار میرود.

به طور مثال decoder-only ها برای تسک های text generation مناسب هستند.

و به طور مثال encoder-decoder ها برای تسک های QA,speech recognition و این دست مسائل قابل استفاده هستند.

۷)

Extractive : همانطور که از اسم این نوع مشخص است به دنبال استخراج کردن میباشند.
این نوع سیستم های QA برای استخراج جواب از خود متن اصلی هستند و اطلاعات مشخصی را

دارند و به دنبال پاسخگویی به سوالات مشخصی هستند و در کل جواب در خود متن اصلی را میگردند و به عنوان خروجی میدهند.

Abstractive : این نوع به صورت انتزاعی عمل میکند یعنی سیستم QA ما جواب را بر اساس درکی که از متن اصلی داشته است تولید میکند و برای سوالاتی که به دنبال استدلال برای موضوعی مربوط به آن متن هستند مفید هستند.

با توجه به ساز و کار بیان شده این تفاوت ها بین این روش(نوع) وجود دارد:

۱. **نوع پاسخ** : پیش تر بیان کردیم.در extractive به دنبال این هستیم که جوابی را مستقیما از متن اصلی پیدا کنیم ولی در abstractive با توجه به درکی که از متن داشتیم به دنبال دلیل و برهان و پاسخ دهی هستیم.
۲. پیچیدگی : با توجه به اینکه در نوع abstractive باید متن درک شود و بر طبق درک یک پاسخ را با دلیل بیان کنیم منطقا پیچیدگی بیشتری نیز داریم.
۳. دقت : با توجه به اینکه تسکی که در extractive داریم ساده تر است و به دنبال این هستیم تا پاسخی را مستقیما از متن اصلی دریافت کنیم در نتیجه دقت ما ، به احتمال زیاد، بیشتر خواهد بود چون در abstractive باید متن را درک کنیم و سپس از درک خود پاسخی را بدهیم که گاها ممکن است متن اشتباه درک شود یا با دقت کافی درک نشود و پاسخ دهی دقت کافی نداشته باشد.

۸)

Task = Sentiment analysis

Multi head attention : یکی از مکانیزم‌های کلیدی در مدل‌های Transformer است که به آنها اجازه می‌دهد تا به بخش‌های مختلف یک جمله یا پاراگراف توجه کنند و درک عمیق‌تری از روابط بین کلمات و عبارات به دست آورند .این امر به بهبود عملکرد مدل در وظایف مختلف NLP مانند Text classification,feature extraction

QA,summarization,و ... به کار میرود.

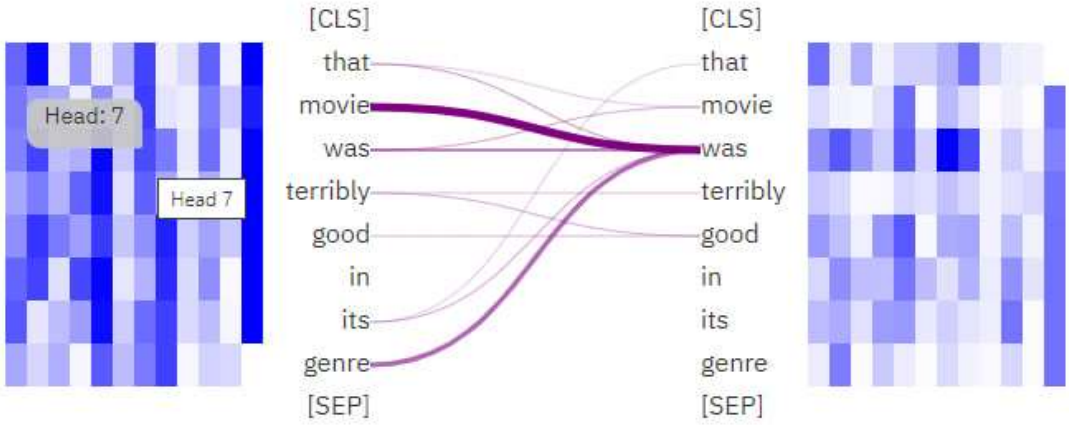
کاربرد های دیگر multi head attention : capture features,rich representation و از این دست میباشد.

در تسک sentiment analysis ، مدل های transformer با استفاده از multiple head attention می‌توانند به بخش‌های مختلف جمله توجه کنند و درک عمیق‌تری از معنای کلی جمله و احساسات نویسنده به دست آورند. این امر به بهبود عملکرد مدل در تشخیص دقیق‌تر احساسات جمله کمک می‌کند.میتوانند به کلمات کلیدی متن توجه بهتری داشته باشند.

در جمله"that movie was terribly good in its genre" ، attention head ها می‌توانند به کلمات کلیدی "terribly" و "good" توجه کنند. این کلمات نشان می‌دهند که نویسنده فیلم را دوست دارد، اما در عین حال فکر می‌کند که فیلم در ژانر خود استثنا است. با در نظر گرفتن لحن جمله و زمینه، مدل می‌تواند تشخیص دهد که احساس کلی نویسنده در مورد فیلم مثبت است.

حال با توجه به موارد بیان شده به بررسی جمله و visualization آن در bert explore میپردازیم.

در کل با داشتن مقادیر مختلف head روابط بین کلمات مختلف با هم بررسی میشوند به طور



مثال

رابطه بین movie,was مورد توجه قرار گرفته است.

یا مثلا با توجه به مثال خودمان ارتباط بین terribly, good که نقش مهمی دارد برجسته میشود.

به صورت کلی اگر بخواهیم نتیجه گیری داشته باشیم؛ داشتن head های مختلف باعث میشود تا روابط بین کلمات مختلف از دیدگاه ها و منظرهای مختلف مورد بررسی قرار بگیرند و باعث میشود bert درک بهتری از کلمات پیدا کند و کلماتی که احساسات مختلفی را بیان میکند بهتر شناسایی و درک کند.

۹) تمام سِل هایی که کامل کردم را قرار میدهم و توضیحات آنرا پایین هر اسکرین میگذارم.

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
```

ابتدا tokernizer مطرح شده در کامنت را لود میکنیم.

```
    # TODO: find start character of answer
    start_char = answer["answer_start"][0]
    # TODO: find end character of answer
    end_char = start_char + len(answer["text"][0])
```

در این قسمت همانطور که در کامنت های سوال از ما خواسته شده بود کاراکتر ابتدا و انتهای پاسخ یا answer را قرار دادیم.

```
from transformers import DefaultDataCollator

data_collator = DefaultDataCollator() # TODO: make an instance
```

یک اینستنس از کلاس معرفی شده یعنی DefaultDataCollator ساختیم.

```
from transformers import AutoModelForQuestionAnswering, TrainingArguments, Trainer

model = AutoModelForQuestionAnswering.from_pretrained("distilbert-base-uncased") # TODO: load distilbert-base-uncased model
```

مدل bert را برای QA لود میکنیم.

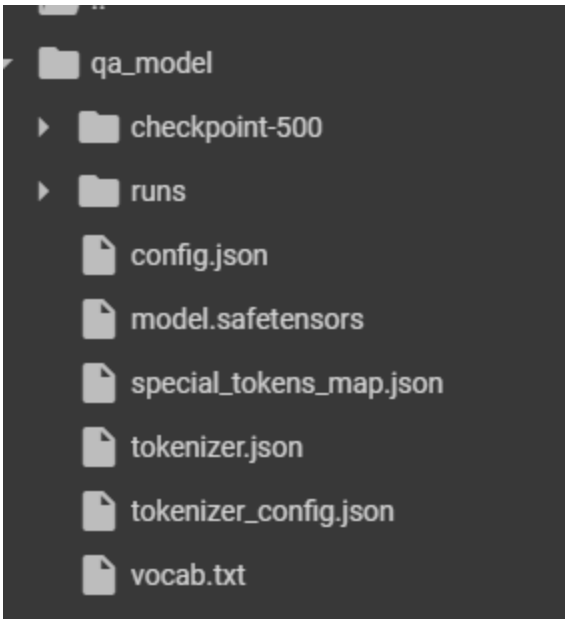
```
training_args = TrainingArguments(
    output_dir="qa_model",
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
)

# TODO: pass the required arguments
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_squad["train"],
    eval_dataset=tokenized_squad["test"],
    tokenizer=tokenizer,
    data_collator=data_collator,
)

trainer.train()
```

پارامترهایی که در کلاس Trainer ساخته شده بودند را به اینستنس trainer که ساختیم پاس میدهیم و در نهایت آبجکتtrainer که از کلاس Trainer میباشد تابعی به نام train دارد آنرا کال میکنیم تا مدل آموزش داده شود.

```
# TODO: save both model and tokenizer
model.save_pretrained("qa_model")
tokenizer.save_pretrained("qa_model")
```

```
('qa_model/tokenizer_config.json',
 'qa_model/special_tokens_map.json',
 'qa_model/vocab.txt',
 'qa_model/added_tokens.json',
 'qa_model/tokenizer.json')
```

در قسمت بعد مدل و توکنایزر را ذخیره میکنیم.



نتایج ذخیره سازی در محیط کولب که شامل فایل ها و کانفیگ های مربوط به مدل و توکنایزر میباشد.

در این قسمت با استفاده از chat gpt یک تکست و یک سوال تولید کردیم.

در این قسمت و با استفاده از pipeline که معرفی شده است و با پاس دادن مدل و توکنایزر خود و البته مشخص کردن نوع تسک میتوانیم به نتیجه دلخواه برسیم.

در رابطه با pipeline نیز طبق داک موجود در hugging face ، ابزاری است که برای تسک های مختلف در حوزه های مختلف از جمله NLP به کار میرود و قابلیت استنتاج یا inference دارد.نحوه کار با آن نیز در مثال های موجود در خود داک مشخص است.

```
[21] from transformers import AutoTokenizer

     tokenizer = AutoTokenizer.from_pretrained("qa_model") # TODO: load your tokenizer
     inputs = tokenizer(question, context, return_tensors="pt")
```

Pass your inputs to the model and return the `logits`:

```
[22] import torch
     from transformers import AutoModelForQuestionAnswering

     model = AutoModelForQuestionAnswering.from_pretrained("qa_model") # TODO: load your model
     with torch.no_grad():
         outputs = model(**inputs) # TODO: pass your inputs to the model
```

Get the highest probability from the model output for the start and end positions:

```
[23] answer_start_index = outputs.start_logits.argmax()
     answer_end_index = outputs.end_logits.argmax()
```

Decode the predicted tokens to get the answer:

```
predict_answer_tokens = inputs.input_ids[0, answer_start_index : answer_end_index + 1]
tokenizer.decode(predict_answer_tokens)
```

```
'12, 742 kilometers'
```

در روش دیگر به صورت دستی توکنایزر و مدلی که در قسمت قبل آموزش داده ایم را فرخوانی میکنیم و مشاهده میکنیم جوابی که در قسمت pipeline گرفتیم و جواب صحیح سوال ما نیز هست را باز هم دریافت کرده ایم.

منابع :

https://medium.com/analytics-vidhya/open-domain-question-answering-series-part-1-introduction-to-reading-comprehension-question-1898c8c9560e

https://medium.com/@mroko001/rnn-vs-lstm-vs-transformers-unraveling-the-secrets-of-sequential-data-processing-c4541c4b09f

https://medium.com/@hunter-j-phillips/positional-encoding-7a93db4109e6

https://vaclavkosar.com/ml/Encoder-only-Decoder-only-vs-Encoder-Decoder-Transfomer

https://www.youtube.com/watch?v=MC3qSrsfWRs