

به نام خدا

بهاره کاوسی نژاد – 99431217

تکلیف سری پنجم NLP

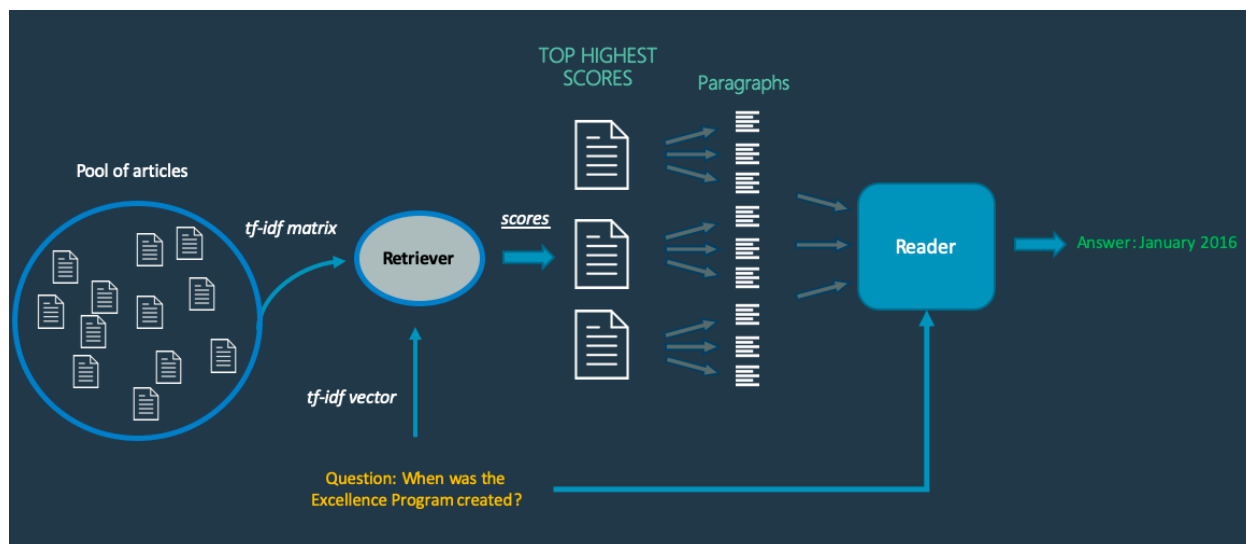
سوالات تئوری

1. دو نوع از سیستم های Question Answering عبارتند از open-domain و close-domain. در مورد هر کدام توضیح دهید و تفاوت آنها را بیان کنید.

- Closed-Domain Question Answering یا به طور مختصر CDQA یک broad name برای پاسخگویی به سوالات فقط از یک دامنه است؛ به عنوان مثال حقوقی، پزشکی، مهندسی و غیره.
 - Open-Domain Question Answering یا به طور مختصر OPQA پاسخگو به سوالات در هر دامنه ای است. به این ترتیب می توان از یک مدل آموزش دیده در مورد هرچیزی سوال پرسید. در OPQA سیستم های مبتنی بر تکنیک های بازبایی اطلاعات، ابتدا یک سند مرتبط با یک سوال را پیدا کرده و سپس از تکنیک های NLP برای استخراج بخش های مرتبط از سند پیروی می کنند.
- یک سوال در اختیار OPQA قرار میگیرد و OPQA باید پاسخی را فرموله کند؛ به عنوان مثال، "آلبرت انیشتین برای چه چیزی جایزه نوبل را برد؟"، پاسخ "قانون اثر فوتوالکتریک" است – در اینجا پاسخ درست قاعده‌تاً مدنظر است.
- در ادامه سه دسته از سوالات وجود دارد که یک سیستم OPQA آموزش دیده باید بتواند به آنها پاسخ دهد (سوالات به ترتیب سختی قرار گرفته اند):
- ابتدایی ترین رفتار این است که بتواند پاسخ سوالی را که مدل در زمان آموزش دیده است به طور صحیح به خاطر بیاورد.
 - یک مدل باید بتواند در زمان آزمون به سوالات جدید پاسخ دهد و از مجموعه پاسخ هایی که در طول آموزش دیده است، پاسخی را انتخاب کند.
 - یک سیستم قوی باید قادر به پاسخگویی به سوالات جدید باشد که پاسخ هایی دارند که در داده های آموزشی موجود نیست.

تفاوت Open-domain و Close-domain:

تفاوت اصلی بین یک سیستم OPQA و CDQA در مجموعه داده ای است که بر روی آن آموزش داده شده است. اگر مدل خود را بر روی مجموعه داده TweetQA آموزش دهید، در نهایت یک QA با دامنه بسته (CDQA) دریافت خواهید کرد که می تواند به سوالات معمولی تویتر پاسخ دهد. برعکس، اگر یک مدل را روی SQuAD آموزش دهید، مدلی خواهید داشت که می تواند تقریباً به هر سوالی پاسخ دهد.



تصویر بالا نحوه عملکرد سیستم های CDQA را نشان می دهد. مجموعه ای از مقالات وجود دارد. هنگامی که یک مدل اطلاعات را برای پاسخ به سوال بازیابی می کند، فقط این مقالات را پردازش می کند. مدل 3-5 مقاله با بالاترین امتیاز را انتخاب می کند، مرتبط ترین پاراگراف ها را از آنجا استخراج می کند و پاسخ را استخراج می کند. ایده اصلی در اینجا این است که دانش مدل (مجموعه داده) بسیار محدود است و همچنین طیف سؤالاتی که مدل می تواند به آنها پاسخ دهد نیز محدود است.

2. ابتدا machine reading comprehension را تعریف کرده و سپس ارتباط آن با Question Answering را توضیح دهید.

Machine Reading Comprehension یک task ساختن سیستمی است که متن را برای پاسخ به سوالات مربوط به آن درک کند. ورودی مدل Reading Comprehension یک سوال و یک context یا passage است. خروجی مدل پاسخی است که از passage ارائه شده است.

Machine Reading Comprehension یا MRC یک task اساسی در textual Question Answering یا QA است که در آن به هر سؤال زمینه مرتبطی داده می شود که از آن می توان پاسخ را استنباط کرد. هدف MRC استخراج پاسخ صحیح از زمینه داده شده یا حتی ایجاد پاسخ پیچیده تر بر اساس زمینه است. MRC وظیفه پل زدن شکاف درک زبان طبیعی بین انسان و ماشین را دارد.

به بیان دیگر، Machine Reading Comprehension یا MRC زیرشاخه Question Answering یا QA است که بر روی نوع خاصی از پاسخگویی به سؤال متمرکز است. نحوه ارتباط این دو را می توان اینگونه بیان کرد:

- Question Answering یا QA یک زمینه گسترده در پردازش زبان طبیعی (NLP) است که با سیستم هایی سروکار دارد که می توانند به سؤالات مطرح شده به زبان طبیعی پاسخ دهند.

- Machine Reading Comprehension یا MRC به طور خاص به یک نوع وظیفه QA اشاره دارد که در آن یک متن و یک سوال مربوط به آن متن به سیستم داده می شود. هدف MRC این است که ماشین به اندازه کافی متن را بخواند و بفهمد که مستقیماً از اطلاعات موجود به سؤال پاسخ دهد.

بنابراین، همه وظایف MRC وظایف QA هستند، اما همه وظایف QA وظایف MRC نیستند. QA می تواند منابع اطلاعاتی دیگری را علاوه بر متن، مانند پایگاه های داده یا نمودارها شامل شود. MRC به طور خاص بر توانایی خواندن و درک متن برای پاسخ به سؤالات متمرکز است.

3. در حوزه QA، دو نوع سوال factoid question و non-factoid question را توضیح دهید.

در QA، کاربر یک سوال زبان طبیعی را وارد می کند و سیستم های QA پاسخی مختصر به سوال کاربر می دهند. سوالات بر اساس طبیعت پاسخشان می توانند factoid یا non-factoid باشد. سؤالات factoid دارای factهای ساده به عنوان پاسخ هستند و یک پاسخ مشخص و objective دارند، و این factها از یک سند بازایی می شوند؛ معمولاً به دنبال پاسخ هایی مانند نام ها، تاریخ ها، مکان ها و تعاریف هستند؛ به عنوان مثال سوال های زیر در دسته factoid قرار می گیرند:

- What is the capital of France?
- When was the first book printed?
- What is the tallest mountain in the world?

در حالی که سؤالات non-factoid معمولاً به عنوان پاسخ دارای اطلاعات readable طولانی تری هستند که ممکن است از اسناد منفرد یا چندگانه باشد. این سوالات فراتر از factهای ساده هستند و برای پاسخ به تحلیل، استدلال یا تفسیر بیشتری نیاز دارند. آنها اغلب شامل نظرات، توضیحات یا قضاوت می شوند؛ به عنوان مثال سوالات زیر در دسته non-factoid قرار می گیرند:

- What is the best way to learn a new language?
- Should I buy a new car?

پاسخ دادن به سؤالات factoid معمولاً برای ماشین ها آسان تر است، زیرا می توان با جست و جو در مقادیر زیادی از داده ها و شناسایی قسمت مربوطه به آن ها پاسخ داد. سوالات غیر واقعی چالش برانگیزتر هستند زیرا به درک عمیق تری از context و زبان نیاز دارند.

جدول زیر نکات بیان شده را به صورت خلاصه نشان می دهد:

Feature	Factoid Question	Non-Factoid Question
Answer Type	Single fact	Explanation, opinion, reasoning
Difficulty for Machines	Easier	More challenging
Example	What is the capital of France?	Should I buy a new car?

4. مزایا و معایب transformer ها را نسبت به RNN ها بیان کنید.

- معماری: RNN ها مدل های متوالی (sequential) هستند که داده ها را یک عنصر یک عنصر در یک زمان پردازش می کنند و یک حالت پنهان داخلی را حفظ می کنند که در هر مرحله به روز می شود. آنها به شیوه ای مکرر عمل می کنند، جایی که خروجی در هر مرحله به حالت پنهان قبلی و ورودی فعلی بستگی دارد.
- Transformer ها مدل های غیر ترتیبی (non-sequential) هستند که داده ها را به صورت موازی پردازش می کنند. آنها بر مکانیزم های self-attention برای گرفتن وابستگی بین عناصر مختلف در دنباله ورودی تکیه می کنند. Transformer ها اتصالات مکرر یا حالت های پنهان ندارند.
- مدیریت Sequence Length: RNN ها می توانند توالی های با طول متغیر را در حین پردازش متوالی داده ها مدیریت کنند. با این حال، توالی های طولانی می توانند منجر به ناپدید شدن (vanishing) یا exploding gradient ها شوند، که گرفتن وابستگی های طولانی مدت را برای RNN ها چالش برانگیز می کند.
- Transformers ها به دلیل ماهیت پردازش موازی خود می توانند توالی های کوتاه و طولانی را به طور موثر اداره کنند. Self-attention به آنها اجازه می دهد تا وابستگی ها را بدون توجه به طول دنباله capture کنند.
- RNN: Dependency Modeling ها برای مدل سازی sequential dependency ها مناسب هستند. آنها می توانند اطلاعات contextual از گذشته را به دست آورند و آنها را برای کارهایی مانند مدل سازی زبان، تشخیص گفتار و تجزیه و تحلیل احساسات مؤثر کنند.
- Transformers ها در مدل سازی وابستگی بین عناصر، صرف نظر از موقعیت آنها در دنباله، برتری دارند. آنها به ویژه برای کارهایی که شامل وابستگی های طولانی مدت هستند، مانند ترجمه ماشین، document classification و image captioning، قدرتمند هستند.
- اندازه مدل: RNN یک اندازه در درجه اول با تعداد واحدهای تکرارشونده (به عنوان مثال سلول های LSTM یا سلول های GRU) و تعداد پارامترهای درون هر واحد تعیین می شود. RNN ها ساختار فشرده ای دارند زیرا عمدتاً به اتصالات مکرر و ابعاد وضعیت پنهان نسبتاً کوچک متکی هستند. تعداد پارامترها در یک RNN با تعداد واحدهای تکرارشونده و اندازه ورودی و ابعاد حالت پنهان نسبت مستقیم دارد.
- Transformer ها به دلیل معماریشان، معمولاً اندازه مدل های بزرگتری دارند. اجزای اصلی که در اندازه یک مدل Transformer نقش دارند، لایه های self-attention، لایه های feed-forward، و positional encoding هستند. Transformer ها طراحی parallelizableتری دارند که امکان محاسبات کارآمد بر روی GPU یا TPU را فراهم می کند. با این حال، این قابلیت پردازش موازی به قیمت تعداد بیشتری از پارامترها تمام می شود.
- آموزش و موازی سازی: RNN، ما بیشتر آن را در یک رویکرد متوالی آموزش می دهیم، زیرا حالت پنهان به مراحل قبلی متکی است. این امر موازی سازی را چالش برانگیزتر می کند و در نتیجه زمان train کندتر می شود.

از سوی دیگر، ما transformerها را به صورت موازی آموزش می دهیم زیرا آنها داده ها را به طور همزمان پردازش می کنند. این قابلیت موازی سازی سرعت train را افزایش می دهد و استفاده از batch size های بزرگ تر را امکان پذیر می کند که باعث کارآمدتر شدن آموزش می شود.

- تفسیر پذیری RNN ها دارای یک جریان زمانی واضح هستند که تفسیر تصمیمات آنها را آسان تر می کند و درک چگونگی جریان اطلاعات از طریق دنباله را آسان تر می کند.
- Transformerها به مکانیسم های self-attention متکی هستند که ممکن است تفسیر تصمیمات آنها را چالش برانگیزتر کند. با این حال، تکنیک هایی مانند attention visualization می توانند بینشی در مورد تمرکز مدل ارائه دهند.

- Pre-training and Transfer Learning: Pre-train کردن RNN ها به دلیل ماهیت متوالی آنها چالش برانگیزتر است. Transfer Learning معمولاً به وظایف خاص یا حوزه های مرتبط محدود می شود.
- ما می توانیم مدل های Transformer را با استفاده از unsupervised objective مانند language modeling یا masked language modeling در مجموعه های مقیاس بزرگ pre-train کنیم. پس از pre-train کردن، می توانیم مدل را بر روی task های مختلف downstream، fine-tune کنیم و transfer learning موثر را امکان پذیر کنیم.
- 5. در مورد positional encoding و اهمیت آن در مدل های مبتنی بر transformer توضیح دهید.

در شبکه Transformer، پردازش embedding vectorها در Transformer attention block برای تبدیل هر بردار به بردار query، key و value به صورت موازی انجام می شود. بنابراین، order information بین کلمات در هیچ کجا مشخص نیست. Order informationها باید مدل سازی شوند - این اطلاعات از طریق positional encodingها مدل سازی می شوند.

Order encodingها بین کلمات در انگلیسی یا هر زبان دیگری اهمیت دارد. برای مثال با توجه به دو جمله:

The man drove the woman to the store.

The woman drove the man to the store.

می توان تصور کرد که با معکوس کردن order information در جمله بالا، ما اساساً در حال معکوس کردن driver جمله هستیم - در این مورد مرد در مقابل زن. بدون order information، تبدیل embedding به contextually rich embedding بی معنی است. بنابراین، positional embedding به همراه word embedding برای پردازش در attention block قرار می گیرد (اضافه می شود).

- 6. در مورد تفاوت encoder-only ها، decoder-only ها و encoder-decoder ها در مدل های مبتنی بر transformer توضیح دهید.

Transformer Modelها در پردازش زبان طبیعی (NLP) رایج هستند، اما بسته به کار، ممکن است از کل معماری یا فقط بخشی از آن استفاده کنید. در اینجا یک تفکیک از transformerهای encoder-only، decoder-only و encoder-decoder ارائه شده است:

- Encoder-Only Transformers

- عملکرد: آنها را به عنوان expert summarizers در نظر بگیرید. آنها یک توالی ورودی (متن، کد، و غیره) را پردازش می کنند و آن را به یک fixed-length representation متراکم می کنند که essence ورودی را به تصویر می کشد.
- موارد استفاده: آنها در کارهایی مانند text classification (مثلاً هرزنامه در مقابل غیر هرزنامه)، clustering similar documents، یا بازیابی اطلاعات (یافتن نتایج جستجوی مرتبط) برتر هستند.

- Decoder-Only Transformers

- عملکرد: یک نویسنده خلاق را تصور کنید. آنها یک دنباله می گیرند و به طور مکرر عنصر بعدی را پیش بینی می کنند و یک دنباله کاملاً جدید را بر اساس ورودی ارائه شده می سازند.
- موارد استفاده: آنها در کارهای تولید متن مانند چت بات ها، نوشتن قالب های متن خلاقانه مختلف (شعر، کد، اسکرپت)، یا خلاصه کردن یک سند بزرگ در یک متن کوتاه و مختصر بسیار عالی عمل می کنند.

- Encoder-Decoder Transformers

- عملکرد: آنها قدرت هر دو encoder و decoder را ترکیب می کنند. Encoder دنباله ورودی را رمزگذاری می کند و decoder از آن نمایش کدگذاری شده برای تولید یک دنباله جدید استفاده می کند. آن را به عنوان مترجمی در نظر بگیرید که یک زبان (encoder) را می فهمد و از این درک برای ایجاد ترجمه دقیق (decoder) در زبان دیگر استفاده می کند.
- موارد استفاده: آنها در کارهایی که نیاز به درک و manipulate کردن زبان دارند، مانند ترجمه ماشین، question answering (پیدا کردن پاسخ سؤال در یک سند)، یا abstractive summarization (ایجاد خلاصه ای که نکات اصلی را در بر می گیرد، اما از کلمات متفاوت استفاده می کند) تسلط دارند.

در جدول زیر نکات بیان شده به صورت خلاصه آمده است:

Feature	Encoder-Only	Decoder-Only	Encoder-Decoder
Function	Summarize	Generate	Understand & Generate
Output	Fixed-length	Sequence	Sequence
Typical Tasks	Classification, Clustering	Text Generation, Summarization	Machine Translation, Q&A

7. یک دسته بندی دیگر برای سیستم های question answering عبارت است از extractive و abstractive. در مورد هر کدام و تفاوت آنها توضیح دهید.

• Abstractive Question Answering

خودتان را در حال پرسیدن از یک کامپیوتر تصور کنید: "پایتخت فرانسه کجاست؟" در یک سناریوی معمولی، کامپیوتر ممکن است اطلاعات را مستقیماً از یک منبع دریافت کند و بگوید: "پایتخت فرانسه پاریس است." این ساده و کاربردی است، اما پاسخ انتزاعی به سؤال آن را یک قدم جلوتر می برد.

مدل های انتزاعی فقط اطلاعات را نمی گیرند. در عوض، آن ها در متن کاوش می کنند، معنا را درک می کنند و سپس پاسخی ایجاد می کنند که ممکن است کپی دقیقی از آنچه قبلاً وجود دارد نباشد. مانند مکالمه با دوستی است که نه تنها حقایق را ارائه می دهد، بلکه یک نظر شخصی و کمی flair به پاسخ اضافه می کند.

دوباره به سوال پایتخت فرانسه توجه کنید. یک مدل انتزاعی ممکن است با پاسخی مانند "پاریس، قلب پر جنب و جوش فرانسه، دارای تاریخ و میراث فرهنگی غنی" باشد. توجه کنید که چگونه این یک تکرار کلمه به کلمه نیست. در عوض، کامپیوتر پاسخی مختصر و گویا را فرموله کرده است و به اطلاعاتی که ارائه می دهد، رنگی از انسانیت اضافه می کند.

➤ مزایا:

1. پاسخ های Human-Like: Abstractive QA پاسخی را به شیوه ای شبیه به انسان ایجاد می کند، و پاسخی را تولید می کند که فراتر از mere extraction است و می تواند یک خلاصه مختصر یا توضیح بازنویسی شده ارائه دهد.

2. انعطاف پذیری: مدل های abstractive می توانند طیف گسترده ای از سؤالات و زمینه ها را مدیریت کنند، و آنها را با سناریوهای مختلف بدون نیاز به extensive fine-tuning برای هر مورد استفاده، سازگار می سازد.

➤ معایب:

1. دشواری در train: آموزش مدل های abstractive به دلیل نیاز به مجموعه داده های بزرگ و معماری های پیچیده می تواند چالش برانگیزتر باشد. دستیابی به عملکرد خوب اغلب به منابع محاسباتی قابل توجهی نیاز دارد.

2. پتانسیل برای تولید اطلاعات نادرست: آزادی در تولید پاسخ ها ممکن است منجر به تولید پاسخی شود که واقعاً نادرست هستند یا از زمینه اصلی منحرف می شوند.

• Extractive Question Answering

یک highlighting pen که قبلاً از آن استفاده می کردیم را تصور کنید. Extractive QA شامل انتخاب مرتبط ترین بخش های متن (معمولاً یک متن یا سند) است که مستقیماً به سؤال پاسخ می دهد. پاسخ گزیده ای از محتوای اصلی است. روش های extractive اساساً اطلاعات را بدون بازنویسی «استخراج» می کنند.

در ادامه مثال قبلی، یک مدل extractive ممکن است کلمات دقیق «پاریس پایتخت فرانسه است» را شناسایی و خروجی دهد.

➤ مزایا:

1. حفظ متن: Extractive QA مستقیماً بخش‌هایی از متن اصلی را انتخاب و ارائه می‌کند و اطمینان می‌دهد

که زمینه و جزئیات اطلاعات حفظ می‌شود. این می‌تواند برای حفظ دقت مفید باشد.

2. سادگی: آموزش و اجرای مدل‌های extractive در مقایسه با مدل‌های abstractive ساده‌تر است. آنها

ممکن است برای عملکرد موثر به قدرت محاسباتی و داده کمتری نیاز داشته باشند.

➤ معایب:

1. خلاقیت محدود: Extractive QA توسط اطلاعات موجود در متن محدود می‌شود. نمی‌تواند پاسخ‌هایی

را ایجاد کند که فراتر از جمله‌بندی دقیق منبع باشد، و توانایی آن را برای ارائه پاسخ‌های خلاقانه یا

ظریف‌تر محدود می‌کند.

2. دشواری استنتاج: مدل‌های extractive ممکن است با سؤالاتی که نیاز به استنباط یا ترکیب اطلاعات از

بخش‌های مختلف متن دارند، مشکل داشته باشند، زیرا بر انتخاب محتوای موجود متمرکز هستند.

چه زمانی از هر رویکرد استفاده کنیم؟

➤ از abstractive QA در موارد زیر استفاده کنید:

- زمانی که پاسخ‌هایی می‌خواهید که بیشتر شبیه یک توضیح انسانی باشد.
- خلاصه کردن اطلاعات بسیار مهم است.
- خلاقیت در ایجاد پاسخ مهم است.

➤ از Extractive QA زمانی استفاده کنید که:

- دقت و تطابق دقیق پاسخ‌ها بسیار مهم است.
- یک سوال ساده با پاسخ واضح در متن ارائه شده دارید.
- بهره‌وری منابع و زمان پاسخ سریع‌تر اولویت هستند.

8. با استفاده از ابزار BERT Explorer مدل bert-base-cased را انتخاب کنید. یک جمله ورودی نمونه (باید جمله را

خودتان بنویسید) و یک task ارائه دهید و توضیح دهید که چرا داشتن multiple head attention مفید است.

ابزار ارائه شده یک Explorable BERT Model است که به کاربران اجازه می‌دهد attention weight لایه‌های

مختلف را در مدل BERT تجسم کنند. در اصل، این ابزاری است که به شما کمک می‌کند تا بفهمید یک مدل BERT

چگونه کار می‌کند.

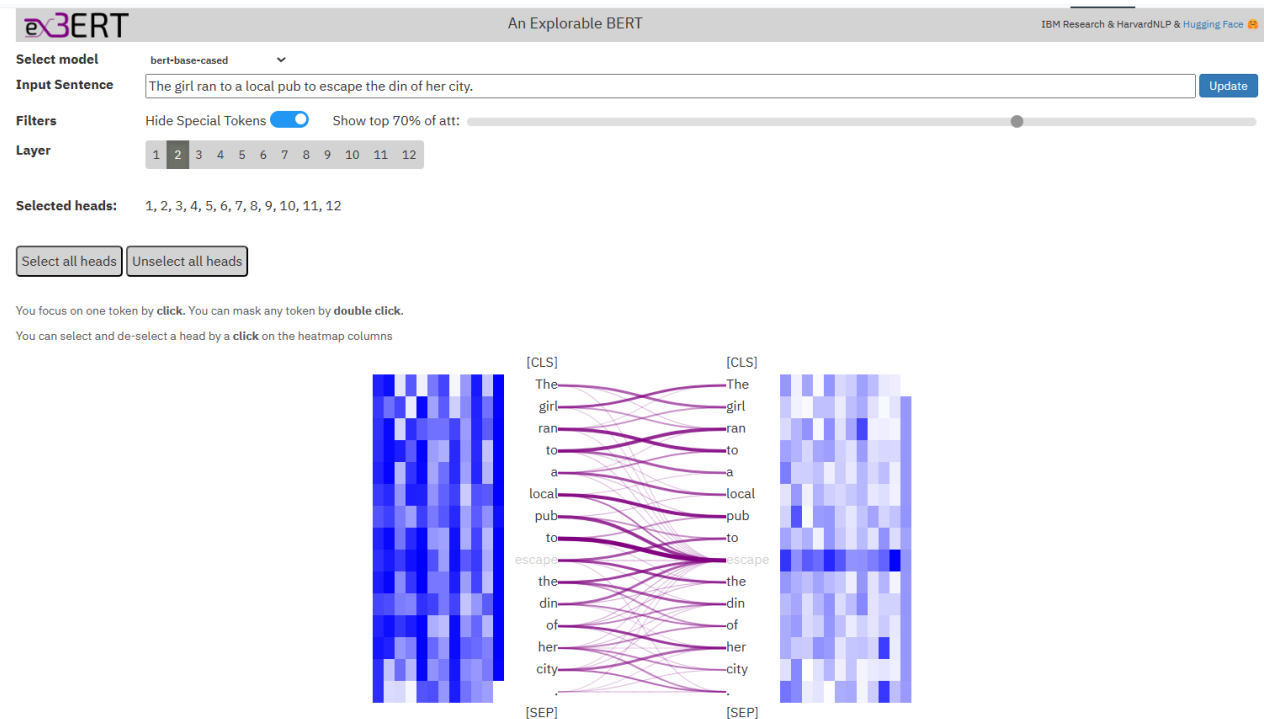
BERT مخفف Bidirectional Encoder Representations from Transformers، یک تکنیک قدرتمند برای

taskهای پردازش زبان طبیعی (NLP) است. کلمات را در رابطه با سایر کلمات یک جمله پردازش می‌کند و به آن

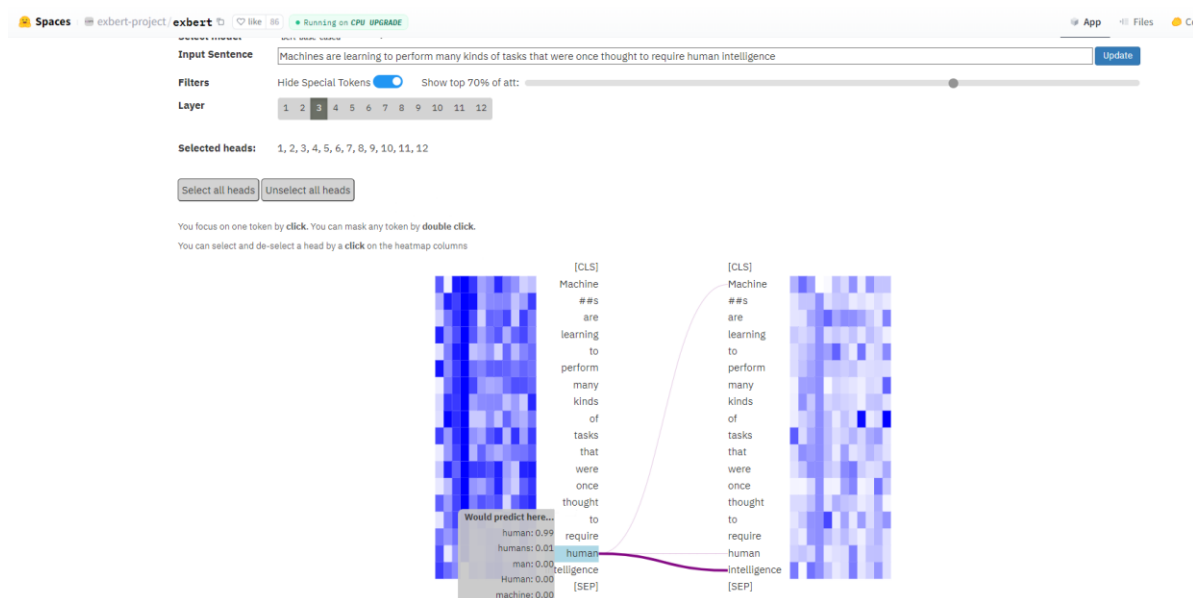
اجازه می‌دهد وابستگی‌های long-range را دریافت کند. با این حال، نحوه رسیدن BERT به خروجی خود می‌تواند

مبهم باشد. اینجاست که ExBERT وارد می‌شود.

ExBERT به شما امکان می دهد جمله ای را وارد کنید و ببینید که چگونه مدل بر قسمت های مختلف جمله تمرکز می کند تا کلمه بعدی را پیش بینی کند. با visualize کردن attention weight ها، می توانید بینش هایی در مورد فرآیند استدلال مدل کسب کنید و نقاط قوت و ضعف آن را بهتر درک کنید. به عنوان مثال، جمله پیشفرض که به مدل داده می شود، نتایج زیر را به همراه دارد:



در مثال دیگری که در نظر گرفتیم، مشاهده می شود که کلمه human با در نظر گرفتن Machine و intelligence تعیین می شود.



تسک NER را در نظر میگیریم؛ Multi-head attention نقش مهمی در task های Named Entity Recognition یا NER در مدل های مبتنی بر transformer ایفا می کند و به مدل اجازه می دهد جنبه های مختلف روابط بین کلمات را در یک جمله ثبت کند. در اینجا مزایای NER آمده است:

- یادگیری روابط متنوع: برخلاف standard attention، multi-head attention چندین "head" ایجاد می کند که به طور مستقل جمله ورودی را تجزیه و تحلیل می کند. هر head به جنبه های مختلف روابط کلمه، مانند part-of-speech tags یا ارتباطات معنایی توجه می کند. این تجزیه و تحلیل جامع به مدل کمک می کند تا سرنخ های مختلفی را برای شناسایی موجودیت بدست آورد.
- Improved Feature Representation: با ترکیب خروجی های تمام attention head ها، مدل نمایش غنی تری از هر کلمه به دست می آورد. این نمایش غنی شده، اطلاعاتی را نه تنها در مورد خود کلمه، بلکه در مورد بافت آن در رابطه با کلمات دیگر در موضوع های مختلف ارائه شده توسط head های متعدد، در بر می گیرد.
- تشخیص مرز موجودیت: multi-head attention می تواند به ویژه در وظایف NER که شامل شناسایی نقاط شروع و پایان موجودیت های نام گذاری شده است (به عنوان مثال، شناسایی "نیویورک سیتی" به عنوان یک موجودیت مکان) مفید باشد. مکانیسم attention به مدل اجازه می دهد تا روی کلمات درون یک موجودیت بالقوه تمرکز کند و روابط بین آنها را بیاموزد و به تشخیص دقیق مرز کمک کند.
- طبقه بندی نوع موجودیت: multi-head attention نیز می تواند در طبقه بندی نوع موجودیت نام گذاری شده (به عنوان مثال، شخص، سازمان، مکان) کمک کند. با توجه به جنبه های مختلف کلمات اطراف، مدل می تواند الگوهایی را بیاموزد که بین انواع موجودیت ها تفاوت قائل می شود.

برخی از کاربردهای خاص multi-head attention در مدل های NER عبارتند از:

- شبکه های Supervised Multi-Head Self-Attention: در اینجا، هر head بر روی یک نوع موجودیت خاص تمرکز می کند و به مدل اجازه می دهد تا همبستگی بین کلمات مربوط به آن نوع خاص را بیاموزد. این می تواند شناسایی موجودیت های nested را بهبود بخشد، جایی که یک نوع موجودیت را می توان به صورت contained با دیگری قرار داد.
- Multi-Head Adjacent Attention: این رویکرد اطلاعات موقعیتی را در کنار attention به منظور جلب اهمیت کلمات همسایه در یک موجودیت بالقوه ترکیب می کند. این برای وظایف NER بسیار مهم است زیرا موجودیت ها اغلب با کلمات متوالی در یک جمله تشکیل می شوند.

به طور کلی، Multi-head attention ابزار قدرتمندی برای مدل های NER ارائه می کند که آنها را قادر می سازد تا روابط پیچیده بین کلمات را بیاموزند که منجر به تشخیص و طبقه بندی دقیق تر موجودیت می شود.

9. در فایل Q9.ipynb قسمت های خواسته شده را تکمیل کنید.

توضیحات قسمت های کامل شده:

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased") # TODO: load
distilbert-base-uncased tokenizer
```

distilbert-base-uncased tokenizer را با استفاده از تابع AutoTokenizer موجود در transformers، load می کنیم.

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

کاراکتر اول و آخر پاسخ را با استفاده از داده answer به دست می آوریم.

```
from transformers import DefaultDataCollator

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

ساختن یک instance به صورت صدا زدن نام کلاس انجام می شود.

```
from transformers import AutoModelForQuestionAnswering, TrainingArguments,
Trainer

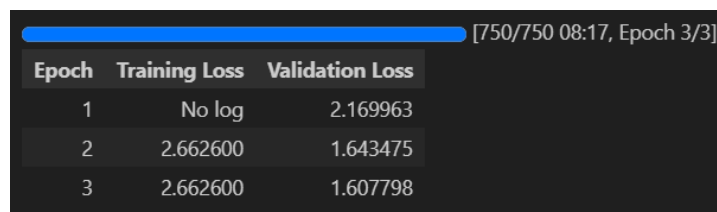
model = AutoModelForQuestionAnswering.from_pretrained("distilbert-base-uncased")
# TODO: load distilbert-base-uncased model
```

مانند اولین TODO، load کردن را انجام می دهیم.

```
# 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,
)
```

ورودی های مورد نیاز را بر اساس داده هایی که از پیش در کد مشخص کردیم به Trainer می دهیم. داده های test، 20 درصد کل داده ها و داده های train، 80 درصد کل داده ها را تشکیل می دهند.

پس از train کردن با trainer.train()، Training Loss و Validation Loss به شکل زیر خواهند بود:



Epoch	Training Loss	Validation Loss
1	No log	2.169963
2	2.662600	1.643475
3	2.662600	1.607798

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

# Save the tokenizer
tokenizer.save_pretrained("qa_tokenizer")
```

مدل و tokenizer که train شد را با نام های qa_model و qa_tokenizer ذخیره می کنیم.

```
question = "Where did Sara go?" # TODO: write a question
context = "Sara was bored, so she went to the park." # TODO: write a context for
your question
```

سوال و context را مشخص می کنیم.

```
from transformers import pipeline

question_answerer = pipeline("question-answering", model="qa_model",
tokenizer="qa_tokenizer") # TODO: call QA pipeline
question_answerer(question=question, context=context)
```

یک pipeline را instantiate می کنیم و مدل و tokenizer خود را به آن می دهیم و سپس آن را صدا می کنیم. به عنوان ورودی سوال و context ای که در قسمت قبل مشخص شد را می دهیم.

```
from transformers import AutoTokenizer

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

با استفاده از تابع AutoTokenizer، tokenizer خود را که با نام qa_tokenizer ذخیره کرده بودیم، load می کنیم و ورودی (سوال و context) را tokenize می کنیم.

```
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
```

با استفاده از تابع AutoModelForQuestionAnswering، مدل خود را که با نام qa_model ذخیره کرده بودیم، load می کنیم. ورودی های tokenize شده را به مدل می دهیم. دو علامت ستاره پیش از inputs برای unpack کردن ورودی هستند و محتوای unpack شده ورودی را به صورت argument های جداگانه به مدل می دهند.

نتیجه پاسخ سوال:

```
'park'
```

- <https://whites.agency/blog/open-domain-question-answering-introduction-to-the-topic/>
- <https://hyperskill.org/learn/step/27243>
- <https://medium.com/analytics-vidhya/open-domain-question-answering-series-part-1-introduction-to-reading-comprehension-question-1898c8c9560e>
- <https://medium.com/analytics-vidhya/open-domain-question-answering-series-part-1-introduction-to-reading-comprehension-question-1898c8c9560e>
- <https://www.mdpi.com/2076-3417/9/18/3698>
- <https://conservancy.umn.edu/items/90239401-18bf-4956-86c1-502b2a2ad328>
- <https://www.baeldung.com/cs/rnns-transformers-nlp>
- <https://www.linkedin.com/pulse/deep-dive-positional-encodings-transformer-neural-network-ajay-taneja/>
- <https://www.linkedin.com/pulse/transformer-architectures-dummies-part-2-decoder-only-bhaskar-t-hj9xc/>
- <https://blog.stackademic.com/mastering-nlp-simplified-guide-to-abstractive-vs-802e5e6c0a26>
- <https://huggingface.co/spaces/exbert-project/exbert>