
 <p>دانشگاه صنعتی خواجه نصیرالدین طوسی دانشکده مهندسی برق - گروه مهندسی کنترل</p>	<p>به نام خدا</p> <p>دانشگاه تهران - دانشکاه صنعتی خواجه نصیرالدین طوسی تهران</p> <p>دانشکده مهندسی برق و کامپیوتر</p>	
<p>سامانه‌ی پرسش - پاسخ</p>		

<p>محمد جواد احمدی</p>	<p>نام و نام خانوادگی</p>
<p>۴۰۱۰۰۰۸۶</p>	<p>شماره دانشجویی</p>

فهرست مطالب

۳	پاسخ پرسش اول	۱
۳	پاسخ قسمت ۱ - مدل سازی مسأله	۱.۱
۹	پاسخ قسمت ۲ - پیش پردازش داده ها	۲.۱
۲۶	پاسخ قسمت ۳ - پیاده سازی مدل	۳.۱
۳۹	پاسخ قسمت ۴ - ارزیابی و پس پردازش	۴.۱

فهرست تصاویر

۴	استفاده از برت در دسته‌بندی	۱
۵	مدل زبانی نقاب‌دار	۲
۶	پیش‌بینی جمله‌ی بعدی	۳
۶	مدل زبانی نقاب‌دار	۴
۷	نسبت انواع نگاشت‌های خودتوجه در مدل برت	۵
۷	شباهت کسینوسی میان نگاشت‌های مسطح خودتوجه در شاخه‌های مختلف مدل برت	۶
۸	استفاده از برت برای استخراج ویژگی و تعبیه کلمات	۷
۱۲	نمایش آماری توزیعات مختلف مجموعه داده	۸
۱۹	خروجی برنامه ۵	۹
۲۱	نمایش کد و خروجی	۱۰

پرسش ۱. سامانه‌ی پرسش-پاسخ

۱ پاسخ پرسش اول

توضیح پوشه کدهای سامانه‌ی پرسش-پاسخ

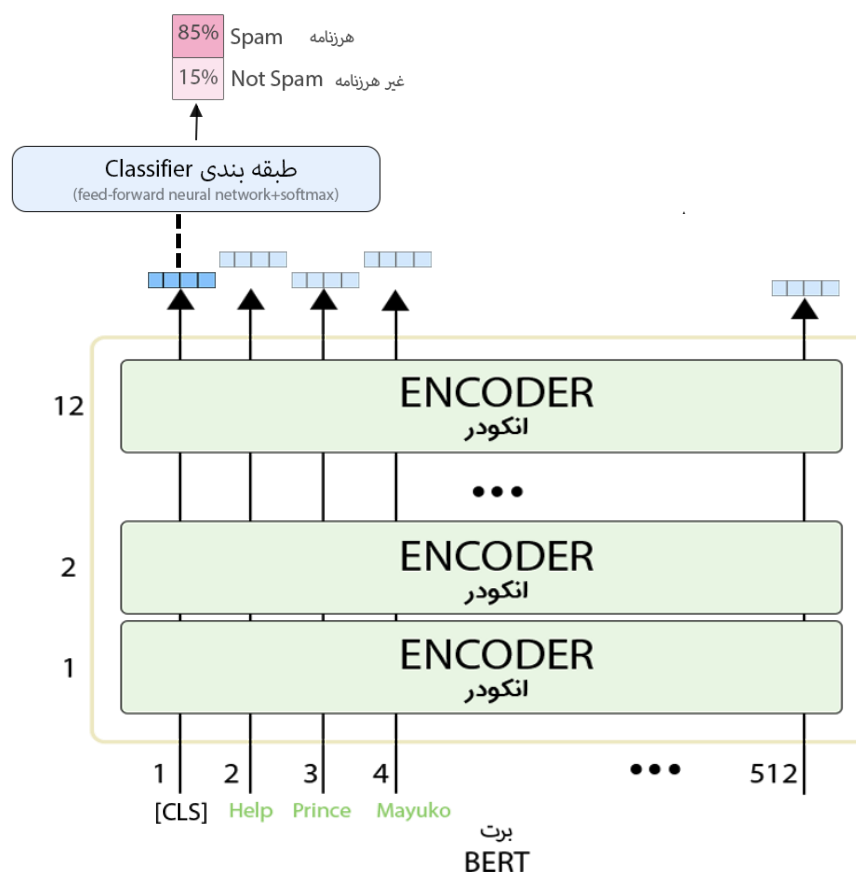
کدهای مربوط به این قسمت، علاوه بر پوشه محلی کدها در این لینک گوگل کولب آورده شده است. مدل‌های ذخیره شده هم از طریق این لینک در دسترس هستند.

۱.۱ پاسخ قسمت ۱ - مدل‌سازی مسأله

BERT^۱ یک مدل زبانی عمیق است که با استفاده از شبکه‌های تبدیل‌کننده^۲ آموزش داده می‌شود. در مرحله پیش‌آموزش، BERT روی یک مجموعه بزرگ از داده‌های متنی آموزش می‌بیند و با تسک‌ها و توابع هدفی مختلف زبانی آشنا می‌شود. معماری BERT از چندین لایه از ترنسفورمرها تشکیل شده است. هر لایه ترنسفورمر دارای یک مکانیزم self-attention و شبکه‌های عصبی پیشروی position-wise است. BERT از رویکرد دوطرفه استفاده می‌کند و با پردازش توالی ورودی به صورت همزمان به جلو و به عقب، اطلاعات متنی مرتبط را به خوبی درک می‌کند. در میان مدل‌های از پیش آموزش دیده، مدل‌های تعبیه کلمات کمک و تغییر محسوسی در دقت شبکه‌ها ایجاد کردند؛ اما اطلاعات کافی نداشتند و همین کمکشان را محدود می‌کرد. در سال ۲۰۱۸ گوگل یک مدل بزرگ BERT با حجم داده فراوان را آموزش داد و آن را برای بهره‌گیری در مسائل متنی در اختیار همگان قرار داد. این مدل در دو اندازه متفاوت آموزش دیده است. این مدل دسته‌ای از انکودهای مدل ترنسفورمر است که آموزش دیده‌اند و هر دو نسخه موجود از آن تعداد زیادی لایه انکودر دارد. مدل پایه آن ۱۲ لایه انکودر یا بلوک ترنسفورمر و ۱۱۰ میلیون پارامتر دارد و مدل بزرگ‌تر ۲۴ لایه انکودر و ۳۴۵ میلیون پارامتر دارد. مدل پایه ۷۶۸ گروه و مدل بزرگ‌تر ۱۰۲۴ گروه در لایه شبکه پیش‌خور خود دارند. تعداد لایه‌های آنتشن هم در مدل پایه ۱۲ و در مدل بزرگ‌تر ۱۶ است. اولین توکن ورودی با یک نام خاص به نام CLS به مدل وارد می‌شود و همانند بخش انکودر مدل ترنسفورمر، این مدل یک توالی از کلمات را در ورودی دریافت می‌کند که در طول لایه‌های انکودر موجود حرکت می‌کنند. ورودی مدل ممکن است یک دنباله شامل حداکثر ۵۱۲ توکن باشد که این دفع قابل تنظیم شدن است. دنباله‌های با توکن SEP از یکدیگر جدا می‌شوند. هر لایه انکودر یک لایه self-attention و یک لایه شبکه پیش‌خور دارد که ورودی‌ها از آن‌ها می‌گذرد و سپس به لایه انکودر بعدی وارد می‌شود. هر موقعیت یک بردار به اندازه گروه‌های لایه پنهان را در خروجی ارائه می‌کند. مثلاً مدل بزرگ‌تر که اندازه لایه پنهان ۷۶۸ دارد، در خروجی بردارهایی به اندازه ۷۶۸ به دست می‌دهد. در مسأله طبقه‌بندی فقط بردار خروجی اول که ورودی آن همان توکن CLS بود اهمیت دارد. این بردار خروجی در مسأله طبقه‌بندی به عنوان ورودی به لایه طبقه‌بندی وارد می‌شود تا نتیجه در خروجی نمایش داده شود. نمایی از این فرآیند در شکل ۱ نمایش داده شده است. اگر دسته‌بندی ما بیش‌تر از دو حالت باشد، تنها کافی است لایه softmax را طوری تغییر دهیم که تعداد خروجی به اندازه کلاس‌های مدنظر ما شود.

^۱Bidirectional Encoder Representations from Transformers

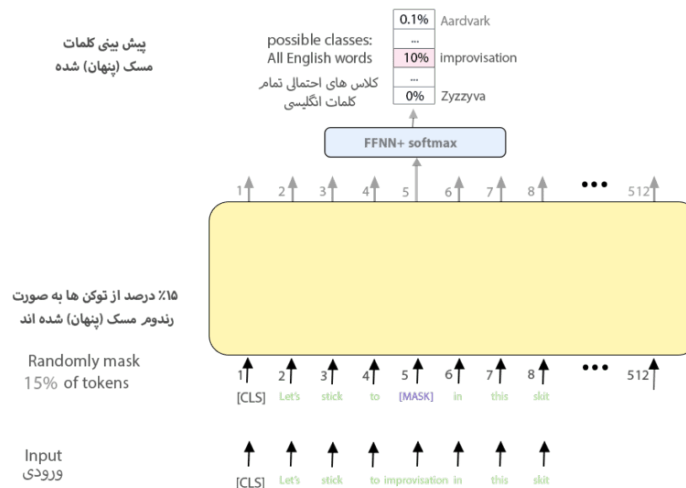
^۲Transformer Networks



شکل ۱: استفاده از برت در دسته‌بندی.

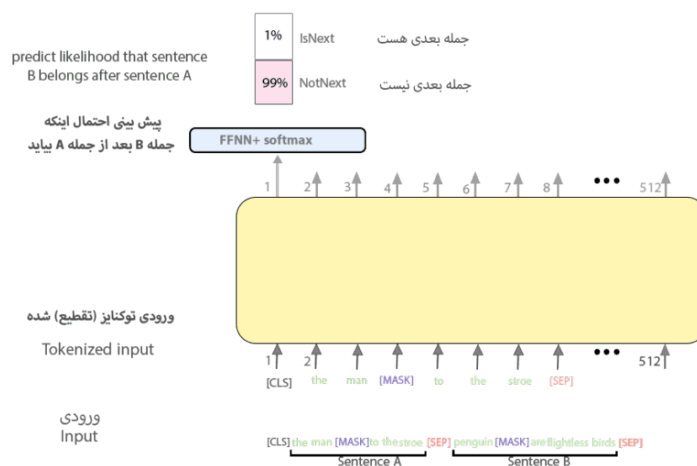
تسک‌های مختلفی با این شبکه قابل انجام است. یکی تسک پیش‌بینی کلمه پنهان است که در این تسک، BERT یاد می‌گیرد که کلمات پنهان در یک جمله را پیش‌بینی کند. در مرحله پیش‌آموزش، حدود ۱۵٪ از توکن‌های ورودی پنهان می‌شوند و BERT آموزش می‌بیند که کلمات پنهان را بر اساس کلمات اطراف حدس بزند. این تسک به BERT کمک می‌کند تا مفاهیم و روابط بین کلمات را درک کند و نمایشی مناسب از آن‌ها ایجاد کند. هدف این تسک این است که تفاوت بین کلمات پیش‌بینی شده و کلمات واقعی پنهان در یک جمله را کمینه کند. برت با استفاده از کلمات پنهان به عنوان ورودی آموزش داده می‌شود و با استفاده از تکنیک‌هایی مانند تابع از دست رفته زبانی (Masked Language Modeling loss) یا تابع انتروپی متقابل بهینه‌سازی می‌شود. تسک قابل انجام دیگر پیش‌بینی جمله بعدی است که در آن مدل BERT آموزش می‌بیند تا تشخیص دهد که آیا یک جمله ادامه منطقی جمله قبلی است یا نه. این تسک روی جفت‌های جملات به عنوان ورودی اعمال می‌شود و برت آموزش می‌بیند تا پیش‌بینی کند که آیا جمله دوم پس از جمله اول در متن اصلی قرار می‌گیرد یا خیر. این تسک به برت کمک می‌کند تا روابط و وابستگی‌های در سطح جمله را درک کند. هدف این تسک است که تفاوت بین برجسب‌های پیش‌بینی شده و برجسب‌های واقعی که نشان می‌دهد آیا جمله دوم پس از جمله اول قرار می‌گیرد یا خیر را کمینه کند. برت با استفاده از جفت جملات به عنوان ورودی آموزش داده می‌شود و با استفاده از تابع خطاهای متقابل دودویی یا توابع هدف مشابه بهینه می‌شود. بنابراین به عنوان جمع‌بندی از دوروش برای آموزش برت استفاده شده است. در مدل زبانی نقاب‌دار ۱۵ درصد لغات متن با توکن MASK جایگزین شده و به ورودی مدل داده می‌شوند. سپس یک لایه طبقه‌بند به اندازه تعداد لغات به همراه یک لایه سافت مکس به خروجی انکودر اضافه می‌شود تا لغات حذف‌شده توسط مدل حدس زده شوند. آموزش این شبکه دوسویه و حساس به کلمات قبلی و بعدی

است. نمایی از این فرآیند در **شکل ۳** نشان داده شده است. هم‌چنین برای این که مدل برت کارایی بهتری در تشخیص ارتباط میان جملات داشته باشد، به مرحله آموزش تسک دیگری نیز اضافه شده که در آن وظیفه مدل این است که یاد بگیرد که دو جمله متوالی هستند یا خیر. در ورودی، دو جمله با توکن SEP از یکدیگر جدا می‌شوند. نمایی از این فرآیند در **شکل ۳** نشان داده شده است. ورودی به BERT شامل جملات و متن است که می‌خواهیم مدل را برای تسک‌های مختلف زبانی آموزش دهیم. برای استفاده از برت، ابتدا جملات و متن مورد نظر را به توکن‌ها تقسیم می‌کنیم. توکن‌ها می‌توانند کلمات، عبارات، علامت‌ها و حروف و نمادهای دیگر باشند. سپس، به هر توکن یک ویژگی و نشانه‌های دیگر نسبت داده می‌شود. یکی از نشانه‌های مهم که به توکن‌ها اختصاص داده می‌شود، "توکن نشانگر جمله" است. برای تشخیص جملات مختلف در ورودی، دو نوع توکن نشانگر جمله استفاده می‌شود: توکن نشانگر جمله آغازین (CLS token) و توکن نشانگر جمله جداگانه (SEP token). توکن نشانگر جمله آغازین به عنوان نماینده‌ی کل جمله استفاده می‌شود و به ابتدای جملات اضافه می‌شود. این توکن برای تسک‌هایی مانند طبقه‌بندی متن یا پرسش و پاسخ به کلمات کلیدی استفاده می‌شود. هم‌چنین، جملات بعد از توکن نشانگر جمله جداگانه تقسیم و جداسازی می‌شوند. این توکن به عنوان نشان‌دهنده انتهای هر جمله در ورودی عمل می‌کند. در واقع علامت‌گذاری مخصوص مانند CLS و SEP به برت کمک می‌کند تا دقیقاً درک کند که هر توکن متعلق به کدام جمله است. به علاوه، به هر توکن یک بردار ویژگی (مانند بردار جاسازی واژگان) اختصاص داده می‌شود که حاوی اطلاعات مربوط به هر توکن است. این بردارها همراه با مکانیزم خودتوجهی و شبکه‌های عصبی پیشروی در لایه‌های ترنسفورمر استفاده می‌شوند تا ارتباطات و وابستگی‌های بین توکن‌ها را درک کنند و نمایشی غنی از متن ایجاد کنند. در نتیجه، ورودی به برت شامل توکن‌ها، توکن نشانگر جمله آغازین و جداگانه، و بردارهای ویژگی مربوط به هر توکن است که برای استفاده در معماری ترنسفورمر و تسک‌های زبانی مورد استفاده قرار می‌گیرد.



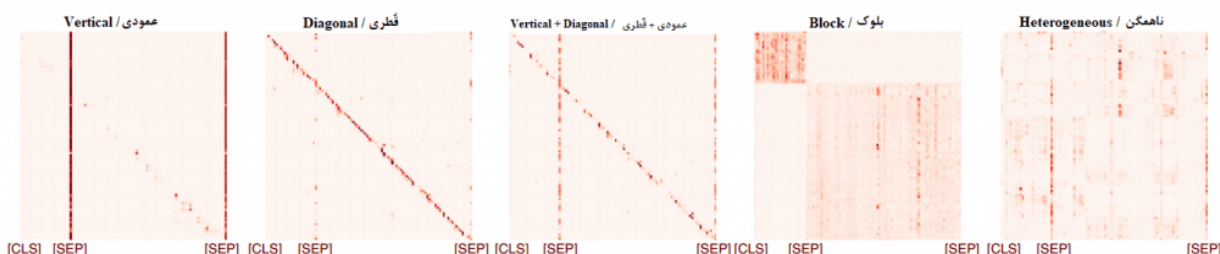
شکل ۲: مدل زبانی نقاب‌دار.

در مدل زبانی برت الگوهای self-attention یا خودتوجهی مختلفی وجود دارد که در **شکل ۴** نشان داده شده است. در این الگوها توکن‌های نمونه، ورودی برت هستند و رنگ‌ها نیز همان اوزان توجه هستند (رنگ روشن‌تر یعنی وزن کم‌تر). در این الگوها، الگوی عمودی توجه به یک توکن واحد را نشان می‌دهد که به‌طور معمول این توکن با نشان‌گر پایان جمله (بین جمله قبل و بعد) است (توکن SEP) و یا توکن CLS. الگوی قطری هم توجه به کلمات قبلی و بعدی را نشان می‌دهد. الگوی بلوک هم توجهی تقریباً یکنواخت به تمامی توکن‌های یک توالی نشان می‌دهد. الگوی ناهمگن هم به‌لحاظ نظری می‌تواند با هر چیزی مانند روابط



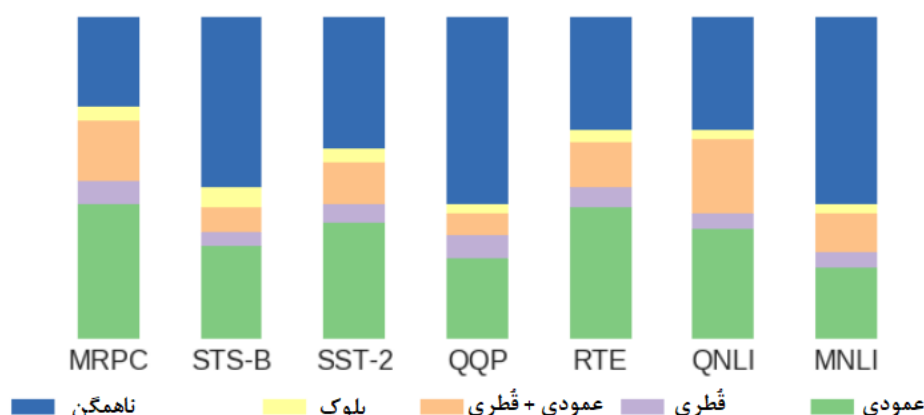
شکل ۳: پیش‌بینی جمله‌ی بعدی.

معنادار میان اجزای توالی ورودی مطابقت داشته باشد. در نمودار آورده‌شده در شکل ۵ نسبت این پنج نوع توجه در مدل برت که



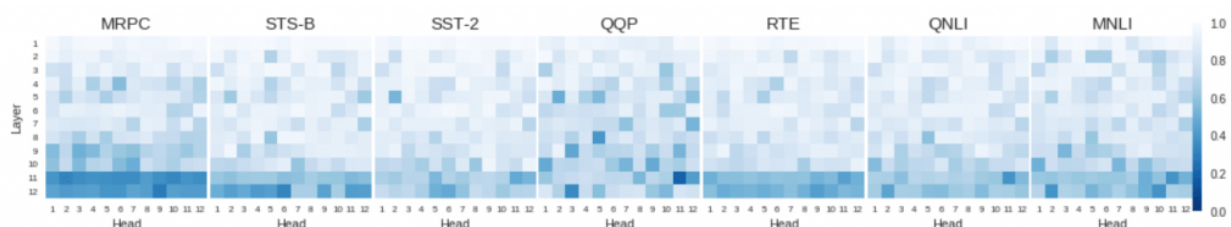
شکل ۴: مدل زبانی نقاب‌دار.

روی GLUE تدقیق شده است نشان داده شده است. هر ستون صد درصد تمامی شاخه‌ها در تمام لایه‌ها را نشان می‌دهد. این نسبت‌ها با توجه به مسأله متغیر هستند؛ اما، در بسیاری موارد این الگوهای معنادار کمتر از نیمی از وزن‌های خودتوجه مدل برت را تشکیل می‌دهند. حداقل یک سوم از شاخه‌های مدل fvz به توکن‌های CLS و SEP توجه می‌کنند که این سبب می‌شود که نتوانیم حجم زیادی از اطلاعات مفید را به بازنمایی‌های لایه بعدی ارسال کنیم. علاوه بر این از این مطب می‌توانیم چنین استنباط کنیم که تعداد پارامترهای مدل بیش از اندازه زیاد است و به همین دلیل است که اخیراً افرادی تلاش کرده‌اند آن را فشرده کنند و البته موفق هم بوده‌اند. با ارائه این توضیحات به دوروش می‌توان از برت استفاده کرد. یکی روش تدقیق یا Fine-tuning است که در آن ورودی مدل فهرستی به طول ۵۱۲ توکن خواهد بود. این توکن‌ها از ۱۲ لایه (در مدل پایه) عبور می‌کنند و در آخر یک بردار با طول ۷۶۸ (در مدل پایه) به عنوان خروجی به دست می‌دهند. این بردار خروجی، می‌تواند ورودی مدل دیگری برای مسأله خودمان مثل هرزنامه یا غیرهرزنامه تشخیص دادن متن باشد. خروجی مدل برت به یک لایه طبقه‌بند (شبکه پیش‌خور و سافت مکس) وارد می‌شود تا احتمال هر کلاس را به دست دهد. هیتمپ آورده‌شده در شکل ۶ شباهت کسینوسی میان ماتریس‌های نگاشت مسطح خودتوجه هر شاخه و لایه، قبل و بعد از انجام تدقیق را نشان می‌دهد. رنگ‌های روشن‌تر نشان‌گر تفاوت کم‌تری در بازنمایی هستند. این نمودارهای مربوط به تدقیق برای تمام مسائل GLUE در سه دوره و گام است. مشاهده می‌شود که اکثر اوزان توجه به میزان اندکی تغییر کرده‌اند و برای بیش‌تر مسائل دو لایه آخر تغییرات زیادی پیدا کرده‌اند. این تغییرات در نوع خاصی از الگوهای



شکل ۵: نسبت انواع نگاشت‌های خودتوجه در مدل برت.

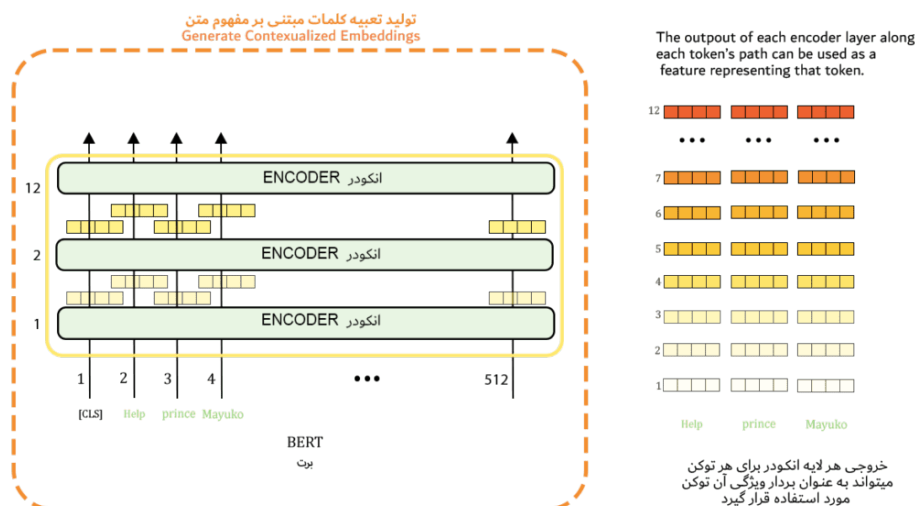
توجه روی نمی‌دهد و در تمامی آن‌ها یکسان است. در عوض، مشخص است که مدل یاد می‌گیرد به الگوهای عمودی توجه بیشتر دقت کند.



شکل ۶: شباهت کسینوسی میان نگاشت‌های مسطح خودتوجه در شاخه‌های مختلف مدل برت.

راهبرد دیگری که از برت می‌توان برای آن استفاده کرد استخراج ویژگی یا Feature Extraction است. در این راهبرد می‌توانیم خروجی لایه‌های میانی آنکودر را به عنوان تعبیه کلمات در نظر بگیریم و مدل خود را با آن‌ها آموزش دهیم. در مدل‌های تعبیه کلمات قبلی مانند GloVe و Word2vec به هر کلمه یک بردار ویژگی اختصاص پیدا می‌کرد. از آن‌جا که در برت قسمت بزرگی از متن همزمان به مدل داده می‌شود و برای تولید بردار هر کلمه، به کلمات قبلی و بعدی نیز توجه می‌شود می‌تواند بهتر و مفیدتر واقع شود. چون ممکن است یک کلمه که مدل‌های تعبیه عادی بردار یکسانی برای آن تولید می‌کنند در جملات مختلف مفاهیم مختلفی را برساند. نمایی از این موضوع در شکل ۷ نشان داده شده است.

با ارائه این توضیحات ساختار پیشنهادی می‌تواند این‌گونه در نظر گرفته شود که ورودی سوالاتی که برای پاسخ استخراج می‌شوند و متن اصلی و پاراگراف متنی‌ای باشد که سوالات از آن استخراج می‌شوند. خروجی مورد انتظار هم پاسخ‌های صحیح به سوالات است. ساختار مدل هم می‌تواند این‌گونه در نظر گرفته شود که یک لایه توکن‌پندی برای تقسیم متن و سوال به توکن‌های جداگانه داشته باشیم. هم چنین برخی توکن‌های ویژه مانند توکن‌های CLS و SEP را باید به متن و سوال اضافه کنیم. سپس توکن‌ها به بردارهای تعبیه‌شده تبدیل می‌شوند و در ادامه لایه‌های ترنسفورمر را خواهیم داشت که شامل بلوک‌های ترنسفورمر هستند که اطلاعات متن و سوال را به طور همزمان پردازش می‌کنند و ارتباطات معنایی و وابستگی‌ها را مدل می‌کنند. برای استخراج پاسخ‌های صحیح، از لایه‌های پیش‌بینی استفاده می‌شود. این لایه‌ها می‌توانند شامل شبکه‌های عصبی پیش‌خور یا لایه‌های توجه خاص باشند. مدل با استفاده از مکانیزم توجه خاصی که روی لایه‌های ترنسفورمر اعمال می‌شود، تلاش می‌کند مکانی را برای



شکل ۷: استفاده از برت برای استخراج ویژگی و تعبیه کلمات.

شروع و پایان موقعیت‌های محتمل پاسخ پیدا کند. معمولاً در مسائل استخراج پاسخ، ائتلاف مدل بر اساس میزان هم‌خوانی بین پاسخ استخراج شده و پاسخ صحیح سنجیده می‌شود. برای محاسبه ائتلاف، می‌توان از تابع هدف مانند معیار تابع هدف تابع هدف متوسط تفاضل مطلق یا تابع هدف تابع هدف متوسط مربعات استفاده کرد. هم‌چنین متریک‌های ارزیابی‌ای که استفاده می‌شوند تا عملکرد مدل را از نظر دقت و کیفیت پاسخ‌های استخراج‌شده اندازه‌گیری کنند عبارتند از: Exact Match (EM) که در صورتی که پاسخ استخراج‌شده به طور دقیق با پاسخ صحیح سوال مطابقت داشته باشد، برابر با ۱ است و در غیر این صورت برابر با ۰ است. F1 Score که این معیار بر اساس میزان هم‌خوانی بین پاسخ استخراج‌شده و پاسخ صحیح سوال محاسبه می‌شود. معیارهای Precision و Recall نیز بر اساس تعداد پاسخ‌های استخراج‌شده صحیح و تعداد کل پاسخ‌های صحیح در داده‌های آزمون محاسبه می‌شوند. Precision نسبت پاسخ‌های درست استخراج‌شده به تعداد کل پاسخ‌های استخراج‌شده و Recall نسبت پاسخ‌های درست استخراج‌شده به تعداد کل پاسخ‌های صحیح است.

بخش مهمی از ساختار در **پاسخ قسمت ۳ - پیاده‌سازی مدل** به تفصیل شرح داده شده است؛ اما به‌عنوان توضیح بیش‌تر ما سه شاخه داده داریم که شامل سوالات، متونی که از آن‌ها سوال استخراج می‌شود و جواب‌ها می‌شوند. خود داده‌ها هم در مجموع به دسته‌های آموزش، اعتبارسنجی و آزمون تقسیم می‌شوند. برای داده‌ها کارهای پیش‌پردازشی مختلفی از جمله توکن‌بندی مناسب انجام می‌دهیم و هر شاخه داده را به تناسب خود آن پیش‌پردازش و آماده می‌کنیم. سپس مدل را ایجاد و مقادیر وزن‌های پیش‌آموزش دیده مدل را از مسیر مشخص شده بارگیری می‌کنیم. سپس یک لایه خطی با ابعاد ورودی ۷۶۸ و خروجی ۲ برای دسته‌بندی (تعیین شروع و پایان) اضافه می‌کنیم. درواقع در بخش داده ما کلیدهایی برای شروع و پایان جملات خواهیم داشت که همین‌ها کلید کار هستند و مشخص می‌کنند شروع و پایان پاسخ کجای متن خواهد بود. بنابراین، این لایه خطی به عنوان یک دسته‌بند در نهایت برای تشخیص شروع و پایان جواب در متن مورد استفاده قرار می‌گیرد. این شروع و پایان را می‌توانیم امتیازدهی کنیم و تابع ائتلاف مناسب برای آن تعیین کنیم تا تابع تابع خطای binary cross-entropy را روی امتیازهای پیش‌بینی شروع و پایان جواب و مقادیر واقعی شروع و پایان جواب محاسبه کند. این تابع خطای باینری برای مسائل دسته‌بندی دودویی استفاده می‌شود. سپس، دو مقدار خطا برای شروع و پایان جواب محاسبه می‌شوند: یکی برای شروع جواب و دیگری برای پایان جواب. در نهایت، مقدار خطای شروع و پایان جواب با هم جمع شده و مقدار نهایی هزینه محاسبه شده برگردانده می‌شود تا میزان ائتلاف را داشته باشیم. باید توجه داشت که می‌توان از تابع BCEWithLogitsLoss استفاده کنیم تا امتیازها را با استفاده

از تابع sigmoid به احتمالات تبدیل کنیم و سپس خطای باینری را برای این احتمالات محاسبه کنیم. برای بحث متریک هم آرایه‌هایی شامل توکن‌های متناظر با محدوده پیش‌بینی و محدوده واقعی شروع و پایان جواب را ایجاد می‌کنیم. این آرایه‌ها با استفاده از np.arange تولید می‌شوند. سپس، تعداد توکن‌های مشترک بین پیش‌بینی و واقعی شمارش می‌شود. برای این منظور، از توابع set.intersection و set.symmetric_difference استفاده می‌شود. با استفاده از تعداد توکن‌های مشترک و تعداد توکن‌هایی که در پیش‌بینی وجود دارند اما در واقعیت نیستند، و تعداد توکن‌هایی که در واقعیت وجود دارند اما در پیش‌بینی نیستند، مقادیر tp (تعداد True Positive)، fp (تعداد False Positive) و fn (تعداد False Negative) محاسبه می‌شوند. در ادامه، دقت، یادآوری و امتیاز F1 محاسبه می‌شوند. دقت نسبت تعداد توکن‌های صحیح پیش‌بینی شده به تعداد کل توکن‌های پیش‌بینی شده است. بازخوانی نسبت تعداد توکن‌های صحیح پیش‌بینی شده به تعداد کل توکن‌های واقعی است. امتیاز F1 نیز میانگین هندسی دقت و یادآوری است. در نهایت، امتیاز F1 محاسبه شده به عنوان خروجی برگردانده می‌شود.

۲.۱ پاسخ قسمت ۲ - پیش‌پردازش داده‌ها

با توجه به ساختار پیشنهادی در مجموعه داده SQuAD فرایند جمع‌آوری داده‌ها در PQuAD مطابق گفته مقاله در سه مرحله انتخاب متن، برچسب‌گذاری جفت سوال-پاسخ و جمع‌آوری پاسخ اضافی انجام شده است. برای استخراج اطلاعات آماری توابع و دستوراتی را تعریف می‌کنیم. ابتدا پس از دریافت داده‌ها از مخزن گیت‌هاب، تابع plot_answer_distribution را تعریف می‌کنیم تا توزیع تعداد پاسخ‌ها برای سوالات در دیتاست و تقسیم‌بندی‌های مختلف آن را مجسم کنیم. وظیفه این تابع این است که با گرفتن دیتاست، نام دیتاست و نام فایل، توزیع تعداد پاسخ‌ها برای سوالات را در قالب نمودار میله‌ای برای دیتاست مورد نظر رسم کند. در این تابع ابتدا یک لیست به نام num_answers برای ذخیره تعداد پاسخ‌ها برای هر سوال ایجاد می‌کنیم؛ سپس، با استفاده از حلقه for تودرتو، برای هر بخش و پاراگراف و سوال، تعداد پاسخ‌ها را استخراج کرده و به لیست num_answers اضافه می‌کنیم. سپس دستوراتی را برای نمایش مناسب اطلاعات و ذخیره‌سازی آن‌ها می‌نویسیم. در ادامه تابع num_answers را برای محاسبه و نمایش آمارهای مربوط به دیتاست نوشته‌ایم که اطلاعات آماری سودمندی از تعداد توکن‌های پاراگراف‌ها، سوالات و پاسخ‌ها به دست می‌دهد و آن‌ها را به صورتی مناسب در خروجی نمایش می‌دهد. با ارائه این توضیحات دستورات به شرح برنامه ۱ است.

Program 1: Code for extracting dataset statistics

```
1 # Clone the PQuAD dataset from GitHub
2 !git clone https://github.com/AUT-NLP/PQuAD.git
3
4 # Load the PQuad dataset
5 train_file_path = '/content/PQuAD/Dataset/Train.json'
6 test_file_path = '/content/PQuAD/Dataset/Test.json'
7 val_file_path = '/content/PQuAD/Dataset/Validation.json'
8
9 import json
10 import matplotlib.pyplot as plt
11 import numpy as np
12
13 # Path to the dataset files
14 train_file_path = '/content/PQuAD/Dataset/Train.json'
15 test_file_path = '/content/PQuAD/Dataset/Test.json'
```

```

16 val_file_path = '/content/PQuAD/Dataset/Validation.json'
17
18 def load_dataset(file_path):
19     with open(file_path, 'r', encoding='utf-8') as file:
20         dataset = json.load(file)
21     return dataset
22
23 def plot_answer_distribution(dataset, dataset_name, file_name):
24     num_answers = []
25     for data in dataset['data']:
26         for paragraph in data['paragraphs']:
27             for qa in paragraph['qas']:
28                 num_answers.append(len(qa['answers']))
29
30     bins = np.arange(max(num_answers) + 2)
31     hist, edges = np.histogram(num_answers, bins=bins)
32
33     fig, ax = plt.subplots()
34     ax.bar(edges[:-1], hist, width=0.8, align='center', color='steelblue')
35
36     # Add count labels in the middle and above each bar
37     for i, count in enumerate(hist):
38         if count > 0:
39             ax.text(edges[i], count + 1, str(int(count)), ha='center', va='bottom')
40
41     plt.xlabel('Number of Answers')
42     plt.ylabel('Number of Questions')
43     plt.title(f'Distribution of Number of Answers for Questions ({dataset_name})')
44     plt.xticks(np.arange(max(num_answers) + 1), [f'{i}-answer' for i in range(max(num_answers) + 1)], rotation=90)
45     plt.legend([dataset_name])
46     plt.savefig(file_name, format='pdf', bbox_inches='tight')
47     plt.show()
48
49 def print_dataset_statistics(dataset, dataset_name):
50     total_questions = 0
51     total_unanswerable_questions = 0
52     total_paragraph_tokens = 0
53     total_question_tokens = 0
54     total_answer_tokens = 0
55
56     for data in dataset['data']:
57         total_questions += len(data['paragraphs'])
58         for paragraph in data['paragraphs']:
59             for qa in paragraph['qas']:

```

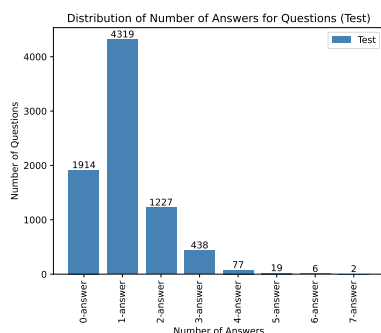
```

60         if qa['is_impossible']:
61             total_unanswerable_questions += 1
62             total_paragraph_tokens += len(paragraph['context'].split())
63             total_question_tokens += len(qa['question'].split())
64             total_answer_tokens += sum([len(answer['text'].split()) for answer in qa['answers
        ']])
65
66     mean_paragraph_tokens = total_paragraph_tokens / total_questions
67     mean_question_tokens = total_question_tokens / total_questions
68     mean_answer_tokens = total_answer_tokens / total_questions
69
70     print(f"Dataset Statistics ({dataset_name}):")
71     print(f"Total Questions: {total_questions}")
72     print(f"Total Unanswerable Questions: {total_unanswerable_questions}")
73     print(f"Mean # of Paragraph Tokens: {mean_paragraph_tokens:.2f}")
74     print(f"Mean # of Question Tokens: {mean_question_tokens:.2f}")
75     print(f"Mean # of Answer Tokens: {mean_answer_tokens:.2f}")
76     print()
77
78 # Load the train dataset
79 train_dataset = load_dataset(train_file_path)
80
81 # Plot the distribution of the number of answers for train questions and save as PDF
82 plot_answer_distribution(train_dataset, 'Train', 'traindistribution.pdf')
83
84 # Print the statistical information of the train dataset
85 print_dataset_statistics(train_dataset, 'Train')
86
87 # Load the test dataset
88 test_dataset = load_dataset(test_file_path)
89
90 # Plot the distribution of the number of answers for test questions and save as PDF
91 plot_answer_distribution(test_dataset, 'Test', 'testdistribution.pdf')
92
93 # Print the statistical information of the test dataset
94 print_dataset_statistics(test_dataset, 'Test')
95
96 # Load the validation dataset
97 val_dataset = load_dataset(val_file_path)
98
99 # Plot the distribution of the number of answers for validation questions and save as PDF
100 plot_answer_distribution(val_dataset, 'Validation', 'validationdistribution.pdf')
101
102 # Print the statistical information of the validation dataset
103 print_dataset_statistics(val_dataset, 'Validation')

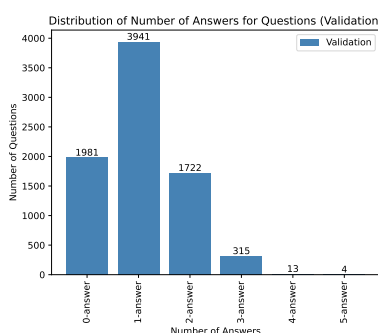
```

نتایج به صورتی است که در شکل ۸ و در زیر آماده است و مشاهده می‌شود که نتایج بر یکی از اشکال مقاله هم منطبق است که این نشان می‌دهد کار به درستی انجام شده:

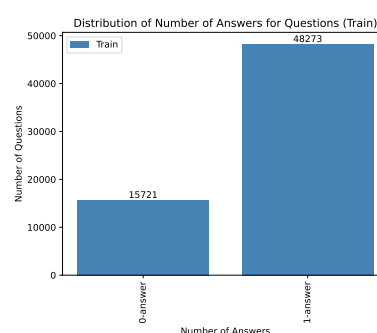
```
1 Dataset Statistics (Train):
2 Total Questions: 8979
3 Total Unanswerable Questions: 15721
4 Mean Num of Paragraph Tokens: 919.56
5 Mean Num of Question Tokens: 73.69
6 Mean Num of Answer Tokens: 28.06
7
8 Dataset Statistics (Validation):
9 Total Questions: 1103
10 Total Unanswerable Questions: 1981
11 Mean Num of Paragraph Tokens: 905.68
12 Mean Num of Question Tokens: 76.47
13 Mean Num of Answer Tokens: 42.83
14
15 Dataset Statistics (Test):
16 Total Questions: 1059
17 Total Unanswerable Questions: 1914
18 Mean Num of Paragraph Tokens: 968.15
19 Mean Num of Question Tokens: 81.91
20 Mean Num of Answer Tokens: 41.80
```



(ج) آزمون



(ب) اعتبارسنجی



(آ) آموزش

شکل ۸: نمایش آماری توزیعات مختلف مجموعه داده.

شماری دیگر از آمارهای این دیتاست در جدول ۱ تا ۴ آورده شده است. برای آماده‌سازی داده‌های ابتدا از آن‌جا که داده‌های در فرمت JSON آماده شده‌اند، تابعی با هدف تبدیل داده‌ها به دیتافریم می‌نویسیم. این تابع فایل JSON را در ورودی دریافت می‌کند، آن را باز می‌کند، داده‌ها را بارگیری می‌کند و لیست‌هایی خالی برای ذخیره مقادیر به صورت مناسب ایجاد می‌کند. سپس، برای هر بخش از داده‌های JSON شامل عنوان (title)، متن (context)، سوال (question)، شروع پاسخ (answer_start) و متن پاسخ (text)، مقادیر مربوطه را استخراج می‌کند. اگر سوالی پاسخ نداشته باشد، مقدار شروع پاسخ را به منفی یک و مقدار متن پاسخ را به رشته خالی تنظیم می‌کند. در نهایت، این مقادیر را

جدول ۱: آمارهای دیتاست

	Train	Validation	Test	Total
Total questions	63994	7976	8002	79972
Unanswerable questions	15721	1981	1914	19616
Mean # of paragraph tokens	125	121	124	125
Mean # of question tokens	10	11	11	10
Mean # of answer tokens	5	6	5	5

جدول ۲: آمارهای دیتاست

Domain	Example	Percentage(%)
Person	Pablo Picasso	22.3
Geographical locations	Caspian Sea	20.1
Science & Tech	Database	6.5
Organization	United Nations	6.0
Sports	Olympic Games	5.8
Fields of specialty	Psychology	5.0
Plants & Animals	Starfish	4.0
Art	Pop music	3.3
Historical Eras	Precambrian	3.2
Books & Movies	Star Wars	2.9
Religious	God	2.7
Events	Nowruz	2.6
Groups	Vikings	2.5
Languages	Middle Persian	2.4
Chemistry & Biology	Hydrogen	2.4
Astronomy	Solar System	2.2
Diseases & Medicines	COVID-19 pandemic	2.0
Objects	Carpet	1.6
Others	Immigration	2.4

به لیست‌های مربوطه اضافه می‌کند. در ادامه، یک دیتافریم خالی ایجاد می‌شود با نام‌های ستون‌های مورد نیاز و مقادیر به درستی تنظیم جای‌گیری می‌شوند. دستوری هم برای حذف سطرهای تکراری از دیتافریم به تابع اضافه کرده‌ایم. درنهایت، در خروجی تابع یک دیتافریم شسته و رفته خروجی داده می‌شود. درنهایت، دستوراتی برای اعمال تابع به داده‌های آموزش و اعتبارسنجی می‌نویسیم و اطلاعات آماری آن‌ها و بخشی از دیتافریم را نمایش می‌دهیم. یک قاب هم برای ترکیب داده‌های آموزش و اعتبارسنجی تعریف می‌کنیم. با این توضیحات، دستورات و بخشی از نتایج در برنامه ۲ آورده شده است.

Program 2: Preparing Data in Preprocessed Dataframe Form

```
1 def json_to_dataframe(file):
2     # Open the JSON file
3     f = open(file, "r")
4     data = json.loads(f.read()) # Load the JSON file
```

جدول ۳: آمارهای دیتاست

POS	NE	Percentage(%)		
		Train	Val	Test
Numeric	Date	10.1	10.2	9.9
	Other numeric	14.3	13.4	14.0
	Person (Individual)	15.3	14.9	17.0
	Location	15.1	12.4	13.6
	Person (Group)	2.7	2.8	3.0
Proper noun phrase	Organization	2.4	2.3	3.0
	Field	1.1	1.1	0.8
	Language	1.2	1.3	1.1
	Other entity	3.4	3.7	3.3
Common noun phrase	-	23.8	27.0	24.2
Adjective phrase	-	2.0	1.9	1.3
Verb phrase	-	4.5	4.9	4.9
Other	-	4.0	4.1	3.8

```

5
6 # Create empty lists to store values
7 ids = []
8 titles = []
9 contexts = []
10 questions = []
11 ans_starts = []
12 texts = []
13
14 # Iterate over the JSON data
15 for i in range(len(data['data'])):
16     title = data['data'][i]['title'] # Extract the 'title' value
17
18     # Iterate over the paragraphs in the JSON data
19     for p in range(len(data['data'][i]['paragraphs'])):
20         context = data['data'][i]['paragraphs'][p]['context'] # Extract the 'context' value
21
22         # Iterate over the questions in the JSON data
23         for q in range(len(data['data'][i]['paragraphs'][p]['qas'])):
24             question = data['data'][i]['paragraphs'][p]['qas'][q]['question'] # Extract the
25             'question' value
26             qid = data['data'][i]['paragraphs'][p]['qas'][q]['id'] # Extract the 'id' value
27
28             # Check if the question has answers
29             if len(data['data'][i]['paragraphs'][p]['qas'][q]['answers']) == 0:
30                 ans_start = -1

```

```

30         text = ''
31
32         # Append the values to the lists
33         titles.append(title)
34         contexts.append(context)
35         questions.append(question)
36         ids.append(qid)
37         ans_starts.append(ans_start)
38         texts.append(text)
39     else:
40         # Iterate over the answers in the JSON data
41         for a in range(len(data['data'][i]['paragraphs'][p]['qas'][q]['answers'])):
42             ans_start = data['data'][i]['paragraphs'][p]['qas'][q]['answers'][a]['
answer_start'] # Extract the 'answer_start' value
43             text = data['data'][i]['paragraphs'][p]['qas'][q]['answers'][a]['text']
44             # Extract the 'text' value
45
46             # Append the values to the lists
47             titles.append(title)
48             contexts.append(context)
49             questions.append(question)
50             ids.append(qid)
51             ans_starts.append(ans_start)
52             texts.append(text)
53
54     # Create an empty DataFrame
55     new_df = pd.DataFrame(columns=['Id', 'title', 'context', 'question', 'ans_start', 'text'])
56
57     # Set the values of the DataFrame columns
58     new_df['Id'] = ids
59     new_df['title'] = titles
60     new_df['context'] = contexts
61     new_df['question'] = questions
62     new_df['ans_start'] = ans_starts
63     new_df['text'] = texts
64
65     # Drop duplicate rows from the DataFrame
66     final_df = new_df.drop_duplicates(keep='first')
67
68     return final_df
69
70 # Convert the train JSON file to a DataFrame
71 df_train = json_to_dataframe(train_file_path)
72

```



```

73 # Get the number of rows in the train DataFrame
74 train_rows = df_train.shape[0]
75 print('Size of the train DataFrame before concatenation is {}'.format(train_rows))
76
77 # Convert the test JSON file to a DataFrame
78 df_test = json_to_dataframe(test_file_path)
79
80 # Convert the validation JSON file to a DataFrame
81 df_validation = json_to_dataframe(val_file_path)
82
83 # Concatenate the train and validation DataFrames
84 # frames = [df_train, df_validation]
85 # df_train = pd.concat(frames)
86
87 # Get the number of rows in the concatenated train DataFrame
88 train_rows = df_train.shape[0]
89 print('Size of the train DataFrame after concatenation is {}'.format(train_rows))
90
91 # Display the first few rows of the train DataFrame
92 df_train.head()

```

در ادامه تابع دیگری تعریف می‌کنیم که با استفاده از اطلاعات پاسخ‌ها، شروع پاسخ و متن‌های مربوطه، اندیس پایان پاسخ‌ها را محاسبه و به هر پاسخ اضافه می‌کند. این تابع سه ورودی `answers_text`، `answers_start` و `contexts` را دریافت می‌کند. سپس یک لیست خالی به نام `new_answers` ایجاد می‌شود تا پاسخ‌های تغییر یافته را در آن ذخیره کنیم. یک حلقه برای هر جفت پاسخ، شروع پاسخ و متن مربوطه ایجاد می‌کنیم. از توابع `zip` و `tqdm` برای همزمان سازی لیست‌ها و نمایش نوار پیشرفت در حلقه استفاده می‌شود. سپس نیم‌فاصله ابتدایی را از متن پاسخ حذف می‌کنیم. این عمل به وسیله تابع `re.sub` انجام می‌شود و الگوی `"^\\u200c"` (نیم‌فاصله) را با رشته خالی جایگزین می‌کند. اگر طول متن باقیمانده از حذف نیم‌فاصله ابتدایی یک واحد کمتر از طول پاسخ اصلی باشد، یعنی پاسخ اصلی یک حرف کمتر از پاسخ قبلی دارد و به `start_shift` یک واحد اضافه می‌شود. این مورد برای مواجهه با مشکل نیم‌فاصله ابتدایی در پاسخ‌ها است. سپس نیم‌فاصله انتهایی را هم از متن حذف می‌کنیم؛ هم‌چنین فاصله ابتدایی و انتهایی را. اندیس شروع پاسخ را با مقدار `start_shift` تغییر می‌دهیم. این کار به منظور تطابق درست اندیس شروع پاسخ با پاسخ اصلی است. در ادامه اندیس پایان پاسخ را براساس طول متن و شروع پاسخ محاسبه می‌کند. اگر بخش متنی از متن اصلی که با استفاده از اندیس شروع و پایان محاسبه شده است با متن پاسخ برابر باشد، اندیس پایان پاسخ را برابر با `end_idx` تنظیم می‌کنیم. در غیر این صورت، اگر پاسخ با ۱ یا ۲ توکن از محدوده درست خارج شده باشد، اندیس شروع و پایان پاسخ را تنظیم مجدد می‌کند. این کار به منظور اصلاح اندیس‌های پاسخ در صورتی است که پاسخ یک یا دو توکن عقب یا جلوتر از محدوده درست باشد. پاسخ تغییر یافته را به لیست `new_answers` اضافه می‌کنیم. پاسخ شامل متن تغییر یافته، اندیس شروع و اندیس پایان است. با این توضیحات، دستورات و بخشی از نتایج در برنامه ۳ آورده شده است.

Program 3: Index Setting and Correction

```

1 def set_and_correct_index(answers_text, answers_start, contexts):
2     new_answers = []
3
4     # Loop through each answer-context pair

```

```

5     for answer_text, answer_start, context in tqdm(zip(answers_text, answers_start, contexts)):
6         start_shift = 0
7
8         # Remove start half-spaces from the answer text
9         text = re.sub("^\\u200c", "", answer_text)
10
11        # Check if the length of the text is one less than the length of the original answer
12        if len(list(text)) == (len(list(answer_text)) - 1):
13            start_shift += 1
14
15        # Remove end half-spaces from the text
16        text = re.sub("\\u200c$", "", text)
17
18        # Remove leading and trailing whitespaces from the text
19        text = re.sub("^\\s+", '', text)
20        text = re.sub("\\s+$", '', text)
21
22        # Adjust the answer_start index by the start_shift value
23        answer_start += start_shift
24
25        # Calculate the end index of the answer
26        end_idx = answer_start + len(text)
27
28        # Check if the answer is correct
29        if context[answer_start:end_idx] == text:
30            # If the answer is correct, set the answer_end index to end_idx
31            answer_end = end_idx
32        else:
33            # If the answer is off by 1-2 tokens, adjust the answer_start and answer_end indices
34            for n in [1, 2]:
35                if context[answer_start - n:end_idx - n] == text:
36                    answer_start = answer_start - n
37                    answer_end = end_idx - n
38
39            # Append the modified answer to the new_answers list
40            new_answers.append({'text': text, 'answer_start': answer_start, 'answer_end': answer_end
41                                })
42
43    return new_answers

```

در ادامه با استفاده از دیتاست ورودی و تابع آماده‌سازی داده‌ها ستون‌های مورد نیاز را استخراج کرده و پردازش پاسخ‌ها را انجام می‌دهیم. این تابع جدید ستون `ans_start` را از دیتاست استخراج می‌کند و به فرمت لیست تبدیل می‌کند. این ستون شامل اندیس شروع پاسخ‌ها است. ستون `text` را از دیتاست استخراج می‌کند و به فرمت لیست تبدیل می‌کند. این ستون شامل متن پاسخ‌ها است. ستون `context` را از دیتاست استخراج می‌کند و به فرمت لیست تبدیل می‌کند. این ستون شامل متن‌های مربوط به سوالات است. ستون `question` را از دیتاست استخراج می‌کند و به فرمت لیست تبدیل می‌کند. این

ستون شامل سوالات است. بعد از انجام این کار تابع قبلی تعریف شده در برنامه ۳ فراخوانی می‌شود تا پاسخ‌ها را پردازش کند و پاسخ‌های تغییر یافته را برگرداند. این پاسخ‌ها شامل متن پاسخ، اندیس شروع و اندیس پایان پاسخ است. این تابع در نهایت یک دیکشنری را برمی‌گرداند که شامل سوالات، متن متن‌ها و پاسخ‌های تغییر یافته است. این دیکشنری داده‌های آماده‌شده را نمایندگی می‌کند و در ادامه می‌تواند برای استفاده در مراحل بعدی پردازش استفاده شود. با این توضیحات، دستورات و بخشی از نتایج در برنامه ۴ آورده شده است.

Program 4: Preparing Dataset

```
1 def prepare_data(dataset):
2     # Extract necessary columns from the dataset
3     answer_start = dataset['ans_start'].tolist()
4     text = dataset['text'].tolist()
5     questions = dataset['question'].tolist()
6     contexts = dataset['context'].tolist()
7
8     # Call the add_end_index function to process answers
9     answers = add_end_index(text, answer_start, contexts)
10
11     # Return a dictionary with the prepared data
12     return {
13         'question': questions,
14         'context': contexts,
15         'answers': answers
16     }
```

سپس دستوراتی می‌نویسیم تا تابع آماده‌شده در برنامه ۴ را با استفاده از دیتافریم `df_train` فراخوانی کند تا داده‌های آماده‌شده را برای مجموعه آموزش ایجاد و در متغیر `train_dataset` ذخیره کند. همین کار را برای داده‌های اعتبارسنجی هم انجام می‌دهیم. سپس یک نمونه را نمایش می‌دهیم. کد و نتایج به شرح زیر است:

Program 5: Applying Preparing Dataset

```
1 # Prepare the training dataset
2 train_dataset = prepare_data(df_train)
3
4 # Prepare the validation dataset
5 val_dataset = prepare_data(df_validation)
6
7 # Prepare the test dataset
8 test_dataset = prepare_data(df_test)
9
10 # Access the second answer in the training dataset
11 answer = train_dataset['answers'][10]
12 train_dataset['answers'][10]
```

در ادامه دستوراتی می‌نویسیم که در توابعی که با استفاده از یک توکنایزر (مانند توکنایزر برت) اطلاعات دیتاست را پردازش می‌کند، استفاده می‌شوند. `new_context`، `new_question` و `new_answer` لیست‌هایی خالی هستند که برای ذخیره‌سازی

```
63994it [00:00, 148179.51it/s]
10379it [00:00, 134221.35it/s]
10417it [00:00, 115438.75it/s]
{'text': 'پنج عنوان قهرمانی در چمپیونشیپ و دو قهرمانی در جام حذفی', 'answer_start': 288, 'answer_end': 343}
```

شکل ۹: خروجی برنامه ۵.

داده‌های توکنیزه شده جدید استفاده می‌شوند. اندازه کل داده‌ها در متغیر `len_data` ذخیره می‌شود. یک حلقه تکرار روی هر نمونه داده اجرا می‌شود و متن پاسخ، متن متناظر با سوال و متن سوال را توکنیزه می‌کند. تعداد کل توکن‌ها محاسبه می‌شود و سپس بررسی می‌شود که تعداد کل توکن‌ها در محدوده مورد نظر قرار دارد یا خیر. در اینجا محدوده تعیین شده برای تعداد توکن‌ها بین $(\text{min_len} - 3)$ و $(\text{max_len} - 3)$ است. عدد ۳ نشان دهنده سه توکن ویژه است: یک `CLS` و دو `SEP`. اگر تعداد توکن‌ها در محدوده مورد نظر قرار داشته باشد و عبارت `[UNK]` در پاسخ وجود نداشته باشد، داده جدید را به لیست‌های جدید اضافه می‌کنیم. تعداد داده‌هایی که حاوی تعداد توکن‌های دلخواه بوده و عبارت `[UNK]` ندارند را چاپ می‌کنیم. هم‌چنین درصد داده‌هایی که حاوی تعداد توکن‌های دلخواه بوده و عبارت `[UNK]` ندارند را نیز چاپ می‌کنیم. با انجام این کار یک دیتاست جدید با داده‌های تمیز، توکنیزه و پیش‌پردازش شده ایجاد می‌شود که آن را در متغیر `new_train_dataset` ذخیره می‌کنیم. این دیتاست شامل فهرست سوال‌ها، فهرست متن‌های متناظر با سوال و فهرست پاسخ‌ها است. با این توضیحات، دستورات و بخشی از نتایج در برنامه ۶ آورده شده است.

Program 6: Make Tokenized and Preprocessed Data

```
1 from transformers import AutoConfig, AutoTokenizer
2
3 config = AutoConfig.from_pretrained("m3hrdadfi/albert-fa-base-v2")
4 tokenizer = AutoTokenizer.from_pretrained("m3hrdadfi/albert-fa-base-v2")
5
6 from transformers import AutoConfig, AutoTokenizer, AutoModel
7
8 config = AutoConfig.from_pretrained("HooshvareLab/bert-base-parsbert-uncased")
9 tokenizer = AutoTokenizer.from_pretrained("HooshvareLab/bert-base-parsbert-uncased")
10
11 # Create empty lists to store the new tokenized data
12 new_context, new_question, new_answer = [[] for _ in range(3)]
13
14 # Get the total size of the data
15 len_data = len(train_dataset['answers'])
16 print('Total size of data is {}'.format(len_data))
17
18 # Iterate through each data instance
19 for i in range(len_data):
20     # Tokenize the answer, context, and question
21     tokenized_answer = tokenizer.tokenize(train_dataset['answers'][i]['text'])
22     tokenized_context = tokenizer.tokenize(train_dataset['context'][i])
23     tokenized_question = tokenizer.tokenize(train_dataset['question'][i])
```

```

24
25 # Calculate the total number of tokens
26 num = len(tokenized_context) + len(tokenized_question)
27
28 # Check if the total number of tokens is within the desired range
29 if num > (min_len - 3) and num <= (max_len - 3): # 3 for three special tokens: 1 for [CLS]
    and 2 for [SEP]
30     if '[UNK]' not in tokenized_answer:
31         # Append the tokenized data to the new lists
32         new_context.append(train_dataset['context'][i])
33         new_question.append(train_dataset['question'][i])
34         new_answer.append({
35             'text': train_dataset['answers'][i]['text'],
36             'answer_start': train_dataset['answers'][i]['answer_start'],
37             'answer_end': train_dataset['answers'][i]['answer_end']
38         })
39
40 # Print the number of data without [UNK] and containing 128-256 tokens
41 print('Number of data without [UNK] and containing 128-256 tokens is {}'.format(len(new_context))
    )
42
43 # Print the percentage of data without [UNK] and containing 128-256 tokens
44 print('Percentage of data without [UNK] and containing 128-256 tokens is {}'.format(100 * len(
    new_context) / len_data))
45
46 # Create a new training dataset with the filtered data
47 new_train_dataset = {
48     'question': new_question,
49     'context': new_context,
50     'answers': new_answer
51 }

```

همین فرآیند را برای سایر دسته‌های داده‌ای تکرار می‌کنیم و نتایج به شرح زیر است:

```

1 ::Train::
2 Total size of data is 63994
3 Number of data without [UNK] and containing 128-256 tokens is 51797
4 Percentage of data without [UNK] and containing 128-256 tokens is 80.94040066256211
5
6 ::Val::
7 Total size of data is 10379
8 Number of data without [UNK] and containing 128-256 tokens is 8258
9 Percentage of data without [UNK] and containing 128-256 tokens is 79.56450525098757
10
11 ::Test::
12 Total size of data is 10417

```

13 Number of data without [UNK] and containing 128-256 tokens is 8367
14 Percentage of data without [UNK] and containing 128-256 tokens is 80.32062973984833

در ادامه دستوراتی برای ایجاد دیتافریم از داده‌های دیتاست اعتبارسنجی و آموزش نوشته می‌شود و طول و نمایشی از بخشی از این داده‌ها نشان داده می‌شود. این دستورات و نتایج آن‌ها در شکل ۱۰ نشان داده شده است.

```
# Create a DataFrame for the test dataset
test_df = pd.DataFrame.from_dict(new_test_dataset)

# Create a DataFrame for the validation dataset
validation_df = pd.DataFrame.from_dict(new_val_dataset)

# Create a DataFrame for the training dataset
train_df = pd.DataFrame.from_dict(new_train_dataset)

# Print the length of the training dataset
print("Length of the training dataset: {}".format(len(train_df)))

# Display the first few rows of the training dataset
print("Training dataset:")
train_df.head()
```

Length of the training dataset: 51797
Training dataset:

	question	context	answers
0	لیگ برتر انگلستان موفق به کسب چند عنوان قهرمان	Arsenal Foo... باشگاه فوتبال آرسنال (به انگلیسی)	{'text': '۱۲', 'answer_start': 173, 'answer_en...
1	بیشترین بازی بدون پیایی متعلق به کدام باش	Arsenal Foo... باشگاه فوتبال آرسنال (به انگلیسی)	{'text': 'باشگاه فوتبال انگلیسی', 'answer_star...
2	باشگاه فوتبال آرسنال موفق به کسب چند عنوان قهر	Arsenal Foo... باشگاه فوتبال آرسنال (به انگلیسی)	{'text': '۱۳', 'answer_start': 119, 'answer_en...
3	باشگاه فوتبال آرسنال چند عنوان قهرمانی در جام	Arsenal Foo... باشگاه فوتبال آرسنال (به انگلیسی)	{'text': '۱۴', 'answer_start': 214, 'answer_en...
4	باشگاه فوتبال آرسنال چند قهرمانی در جام اتحادی	Arsenal Foo... باشگاه فوتبال آرسنال (به انگلیسی)	{'text': 'تو قهرمانی', 'answer_start': 249, 'a...

```
# Print the length of the validation dataset
print("Length of the validation dataset: {}".format(len(validation_df)))

# Display the first few rows of the validation dataset
print("Validation dataset:")
validation_df.head()
```

Length of the validation dataset: 8258
Validation dataset:

	question	context	answers
0	آژانس بین‌المللی انرژی اتمی برای حفاظت از بازی	...با توجه به خشونت‌های اخیر که در ریونوژانیزو و	{'text': 'برای جلوگیری از وقوع حملات تروریستی', 'answer_start': 10...
1	آژانس بین‌المللی انرژی اتمی برای حفاظت از بازی	...با توجه به خشونت‌های اخیر که در ریونوژانیزو و	{'text': 'امکانات لازم را در اختیار این کشور ق', 'answer_start': 10...
2	آژانس بین‌المللی انرژی اتمی امکانات لازم	...با توجه به خشونت‌های اخیر که در ریونوژانیزو و	{'text': 'برای جلوگیری از وقوع حملات تروریستی', 'answer_start': 10...
3	آژانس بین‌المللی انرژی اتمی امکانات لازم	...با توجه به خشونت‌های اخیر که در ریونوژانیزو و	{'text': 'برای جلوگیری از وقوع حملات تروریستی', 'answer_start': 10...
4	طبق آمار اوایل ماه دسامبر 2015 چند نفر به تب د	...مشکل دیگر برزیل مقابله با پشه‌ها و برخی بیماری	{'text': '۱۰,۵۸ میلیون نفر', 'answer_start': 10...

```
# Print the length of the test dataset
print("Length of the test dataset: {}".format(len(test_df)))

# Display the first few rows of the test dataset
print("Test dataset:")
test_df.head()
```

Length of the test dataset: 8367
Test dataset:

	question	context	answers
0	چند اسم از اسماء قرآن در خود قرآن آمده‌است؟	...به نوشته محمد صادقی تهرانی، اسماء قرآن، از جم	{'text': 'حدود چهل اسم', 'answer_start': 582, ...
1	چند اسم از اسماء قرآن در خود قرآن آمده‌است؟	...به نوشته محمد صادقی تهرانی، اسماء قرآن، از جم	{'text': 'حدود چهل اسم در قرآن آمده', 'answer...
2	چند اسم از اسماء قرآن در خود قرآن آمده‌است؟	...به نوشته محمد صادقی تهرانی، اسماء قرآن، از جم	{'text': 'چهل اسم', 'answer_start': 587, 'answ...
3	از جمله امتیازات قرآن چیست؟	...به نوشته محمد صادقی تهرانی، اسماء قرآن، از جم	{'text': 'این است که چه اسم‌های خود قرآن و چه', 'answer...
4	از جمله امتیازات قرآن چیست؟	...به نوشته محمد صادقی تهرانی، اسماء قرآن، از جم	{'text': 'چه اسم‌هایی که', 'answer_start': 587, 'answ...

شکل ۱۰: نمایش کد و خروجی.

برای جمع‌بندی ادامه دستورات را می‌نویسیم. این دستورات دیتاست‌ها را در دسته‌بندی‌های مختلفی که باید آماده و ذخیره می‌کند و داده‌ها را با استفاده از توکنایزر ساخته‌شده توسط دستورات توکن‌ساز (بنابه مدل انتخابی) توکنیزه می‌کند و توکن‌ها را به صورت پدینگ‌شده با طول مشخص‌شده (max_len) تولید می‌کند. هم‌چنین تابعی برای برگرداندن شماره توکن متناظر با

شماره کاراکتر در متن می‌نویسیم و تابعی که به ازای هر سوال در داده، شماره توکن‌هایی که با شروع و پایان پاسخ متناظرند را به توکن‌های ورودی (encodings) اضافه می‌کند. شماره توکن‌های شروع و پایان پاسخ هم ذخیره می‌شوند. به‌عنوان توضیحات دقیق‌تر، در این دستورات ابتدا ستون‌های مربوط به بخش‌های مورد نیاز سوال، متن و پاسخ از دیکشنری dataset، استخراج و به یک لیست تبدیل می‌شوند. این لیست شامل سوالات، پاسخ‌ها و متن‌های موجود در دیتاست است. بنابراین تابع دوم آماده‌سازی داده‌ها یک دیکشنری را برمی‌گرداند که شامل داده‌های آماده‌سازی شده و کلیدهای مربوطه است. در ادامه با استفاده از توکن‌ساز tokenizer تعریف‌شده در قبل، قسمت‌های مختلف توکن‌بندی می‌شوند. توکن‌سازی شامل truncation، افزودن پدینگ با طول بیشینه و برگرداندن تنسورها به فرمت مشخص‌شده است. لازم به ذکر است که truncation در مواقعی که متن ورودی بیشتر از حداکثر طول مشخص‌شده توسط توکن‌ساز است اعمال می‌شود. تراشها معمولاً برای کوتاه‌تر کردن متن استفاده می‌شوند تا به حداکثر طول مشخص‌شده برسد. اگر truncation=True باشد، تراشها فعال خواهند بود و در صورتی که متن ورودی بیشتر از max_len باشد، توکن‌ساز تعداد توکن‌های اضافی را حذف خواهد کرد تا به طول مشخص‌شده برسد. در این حالت، ممکن است بخشی از متن از دست برود تا طول مشخص‌شده را رعایت کند. درنهایت نتیجه توکن‌بندی در متغیرهای مربوط به هر دسته قرار می‌گیرد. در ادامه تابعی نوشته شده که یک اندیس کاراکتر را به اندیس توکن متناظر تبدیل می‌کند. این زمانی کاربردی است که ما نیاز داریم بدانیم توکنی که با یک کاراکتر خاص در متن مطابقت دارد، کدام توکن است. در واقع تابع char_idx_to_token_idx اندیس یک کاراکتر را به اندیس توکن متناظر در جمله تبدیل می‌کند. برای این کار، ابتدا جمله را به توکن‌ها تبدیل می‌کند و سپس با استفاده از لیستی که نشان‌دهنده این است که هر کاراکتر یک فاصله یا نیم‌فاصله است، اندیس توکن متناظر با کاراکتر مورد نظر را محاسبه می‌کند. برای این کار ابتدا یک لیست به نام char ایجاد می‌کند که برای هر کاراکتر در جمله مشخص می‌کند که آیا کاراکتر فاصله یا نیم‌فاصله است. این لیست به صورت دودویی است، به این معنی که عنصری با مقدار ۰ برابر فاصله را نشان می‌دهد و عنصری با مقدار ۱ برابر نیم‌فاصله را نشان می‌دهد. سپس جمله را با استفاده از توکن‌ساز به توکن‌ها تبدیل می‌کند و نتیجه را در متغیر tokens ذخیره می‌کند. سپس اندیس اولیه را به مقدار char_idx تنظیم می‌کند. یک متغیر کمکی به نام counter برای شمارش تعداد کاراکترها تعریف می‌کند. متغیر کمکی به نام flag_continue برای مشخص کردن ادامه جستجوی اندیس توکن استفاده می‌شود. این متغیر به ابتدا به مقدار True مقداردهی می‌شود. یک متغیر کمکی به نام token_index که مقدار اولیه آن برابر ۰ است، تعریف می‌شود تا اندیس توکن متناظر را ذخیره کند. با استفاده از دو حلقه for، اندیس توکن متناظر با کاراکتر مورد نظر را پیدا می‌کند. حلقه بیرونی تمام توکن‌ها را پیمایش می‌کند و حلقه داخلی هر کاراکتر در هر توکن را بررسی می‌کند. در صورتی که مقدار counter برابر با char_idx شود، به معنی یافتن کاراکتر مورد نظر، متغیر flag_continue را به False تغییر می‌دهد و مقدار token_index را با اندیس توکن محاسبه‌شده تنظیم می‌کند. در نهایت، اندیس توکن محاسبه‌شده را برمی‌گرداند. به این ترتیب، این تابع با استفاده از توکن‌سازی جمله و لیست char می‌تواند اندیس توکن متناظر با یک کاراکتر مشخص را در جمله تعیین کند. تابع add_token_positions هم اطلاعات مربوط به موقعیت توکن‌های شروع و پایان پاسخ را به encodings اضافه می‌کند. برای هر سوال در لیست پاسخ‌ها، اندیس توکن‌های شروع و پایان پاسخ را برای آن محاسبه کرده و در لیست‌های مجزا ذخیره می‌کند. این تابع ابتدا دو لیست به نام target_start_list و target_end_list ایجاد می‌کند که هر کدام حاوی لیستی از صفر و یک هستند و اندیس‌هایی را که توکن شروع و پایان پاسخ را نشان می‌دهند، نگهداری می‌کنند. با استفاده از حلقه for، برای هر سوال در لیست پاسخ‌ها عملیات‌های انجام می‌شود. دو لیست target_start و target_end که هر کدام از طول max_len هستند از صفر پر می‌شوند. اندیس شروع و پایان پاسخ از دیکشنری پاسخ‌ها دریافت می‌شود. بررسی می‌شود که آیا اندیس‌های شروع و پایان پاسخ داخل محدوده طول متن قرار دارند یا خیر. در صورتی که سوال قابل پاسخ باشد (شروع و پایان پاسخ معتبر باشد) اندیس توکن‌های شروع و پایان پاسخ با استفاده از تابع char_idx_to_token_idx محاسبه می‌شود. مقدار یک در اندیس توکن محاسبه‌شده برای target_start و target_end تنظیم می‌شود و آن‌ها به target_start_list و target_end_list اضافه می‌شوند. در صورتی که سوال قابل پاسخ نباشد

هم مقدار یک در اندیس اولیه (۰) برای target_start و target_end تنظیم می‌شود و آن‌ها به target_start_list و target_end_list اضافه می‌شوند. در پایان حلقه for، اطلاعات محاسبه‌شده را با استفاده از متد update به encodings اضافه می‌کنیم. در نهایت بخش‌های مختلف دسته‌های مختلف داده‌ای به ترتیب فراخوانی می‌شوند تا موقعیت توکن‌ها برای شروع و پایان پاسخ به encodings هر یک اضافه شود.

Program 7: Make Final Tokenized and Preprocessed Data

```

1 def prepare_data_2(dataset):
2     # Extract the questions, contexts, and answers from the dataset
3     questions = dataset['question'].tolist()
4     contexts = dataset['context'].tolist()
5     answers = dataset['answers'].tolist()
6
7     # Return a dictionary containing the prepared data
8     return {
9         'question': questions,
10        'context': contexts,
11        'answers': answers
12    }
13
14 # Prepare the training dataset
15 train_dataset = prepare_data_2(train_df)
16
17 # Prepare the validation dataset
18 val_dataset = prepare_data_2(validation_df)
19
20
21 # Tokenize the training data using the tokenizer
22 train_data = tokenizer(train_dataset['context'], train_dataset['question'],
23                        truncation=False, padding='max_length',
24                        max_length=max_len, return_tensors='pt')
25
26 # Tokenize the validation data using the tokenizer
27 validation_data = tokenizer(val_dataset['context'], val_dataset['question'],
28                             truncation=False, padding='max_length',
29                             max_length=max_len, return_tensors='pt')
30
31 # Tokenize the test data using the tokenizer
32 test_data = tokenizer(test_dataset['context'], test_dataset['question'],
33                      truncation=False, padding='max_length',
34                      max_length=max_len, return_tensors='pt')
35
36 # Function to convert character index to token index
37 def char_idx_to_token_idx(tokenizer, char_idx, sentence):
38     # Create a list of binary values indicating whether each character is a space or a half-space
39     char = [0 if sentence[i] == ' ' or sentence[i] == ' ' else 1 for i in range(len(sentence))]

```



```

40     tokens = tokenizer.tokenize(sentence)
41     index = char_idx
42
43     # Adjust the index to consider half-spaces
44     for i in range(index):
45         if char[i] != 1:
46             index -= 1
47
48     counter = 0
49     flag_continue = True
50     token_index = 0
51
52     # Find the token index corresponding to the character index
53     for i in range(len(tokens)):
54         if tokens[i].startswith('##'):
55             tokens[i] = tokens[i][2:]
56         for j in range(len(tokens[i])):
57             counter += 1
58             if counter == index:
59                 flag_continue = False
60                 token_index = i
61                 break
62         if not flag_continue:
63             break
64
65     return token_index
66
67 # Function to add token positions for answer start and end
68 def add_token_positions(tokenizer, encodings, answers, contexts):
69     target_start_list = []
70     target_end_list = []
71     for i in tqdm(range(len(answers))):
72         target_start = [0] * max_len
73         target_end = [0] * max_len
74         start_idx = answers[i]['answer_start']
75         end_idx = answers[i]['answer_end']
76
77         if start_idx <= len(contexts[i]) and end_idx <= len(contexts[i]):
78             # Answerable question
79             if start_idx != -1 and end_idx != -1:
80                 start_token_idx = char_idx_to_token_idx(tokenizer, start_idx, contexts[i]) + 1
81                 end_token_idx = char_idx_to_token_idx(tokenizer, end_idx, contexts[i]) + 1
82                 target_start[start_token_idx] = 1
83                 target_end[end_token_idx] = 1
84                 target_start_list.append(target_start)

```

```
85         target_end_list.append(target_end)
86
87         # Unanswerable question
88         else:
89             target_start[0] = 1
90             target_end[0] = 1
91             target_start_list.append(target_start)
92             target_end_list.append(target_end)
93         else:
94             continue
95
96         # Update the encodings with the target start and end lists
97         encodings.update({'targets_start': target_start_list, 'targets_end': target_end_list})
98
99         # Add token positions to the train_data encodings
100 add_token_positions(tokenizer, train_data, train_dataset['answers'], train_dataset['context'])
101
102         # Add token positions to the validation_data encodings
103 add_token_positions(tokenizer, validation_data, val_dataset['answers'], val_dataset['context'])
104
105         # Add token positions to the test_data encodings
106 add_token_positions(tokenizer, test_data, test_dataset['answers'], test_dataset['context'])
```

توضیح نحوه تغییر مدل

توضیحات داده شده در پاسخ قسمت ۳ - پیاده سازی مدل و پاسخ قسمت ۴ - ارزیابی و پس پردازش به صورت کلی هستند و برای سوییچ کردن بین دو مدل پارس برت و آلبرت تنها قسمت های مربوط به تعاریف (مدل، توکنایزر و غیره) و لینک اتصال مربوطه تغییر پیدا کرده است. در استفاده از مدل آلبرت با توجه به توضیحات صفحه هاگینگ فیس باید از نصب مواردی اطمینان حاصل کرد.

۳.۱ پاسخ قسمت ۳ - پیاده سازی مدل

در ادامه قسمت قبل، ابتدا یک کلاس سازنده ی داده های سفارشی برای استفاده در آموزش مدل شبکه عصبی تعریف می کنیم. در این کلاس، سه تابع برای مقداردهی اولیه، دریافت یک آیتم از مجموعه داده، و محاسبه طول مجموعه داده تعریف شده است. در تابع `init`، مقداردهی اولیه انجام می شود و دیکشنری داده ها را دریافت می کند و آن را در متغیر `self.data` ذخیره می کند. از متد `getitem` برای دریافت یک آیتم از مجموعه داده استفاده می شود. با دریافت یک شاخص (`idx`)، داده مربوط به آن شاخص را از مجموعه داده برمی گرداند. این داده به صورت یک دیکشنری از تنسورها برگردانده می شود که کلیدهای آن برابر با کلیدهای موجود در متغیر `self.data` است و مقادیرشان تنسورهای متناظر هستند. متد `len` هم طول مجموعه داده را برمی گرداند. برای این منظور، طول داده موجود در کلید `'input_ids'` از متغیر `self.data` را برمی گرداند. در ادامه، ابتدا یک نمونه از این کلاس برای مجموعه داده های آموزش ایجاد می شود و سپس طول این مجموعه داده چاپ می شود. سپس یک بار دیگر برای مجموعه داده های اعتبارسنجی نیز این عملیات صورت می گیرد و طول مجموعه داده اعتبارسنجی نیز چاپ می شود. در نهایت، دو لودر داده برای آموزش و اعتبارسنجی ایجاد می شوند که برای تقسیم داده ها به دسته ها و مخلوط کردن آن ها استفاده می شوند. با این توضیحات، دستورات و بخشی از نتایج در برنامه ۸ آورده شده است.

Program 8: Make Custom Datasets and Dataloaders

```
1 # Custom Dataset class
2 class CustomDataset(torch.utils.data.Dataset):
3     def __init__(self, data):
4         """
5         Initialize the CustomDataset.
6
7         Args:
8             data (dict): The data dictionary containing input tensors.
9         """
10        self.data = data
11
12    def __getitem__(self, idx):
13        """
14        Get an item from the dataset.
15
16        Args:
17            idx (int): The index of the item.
18        """
```

```

19     Returns:
20         dict: A dictionary containing input tensors.
21     """
22     return {key: torch.tensor(val[idx]) for key, val in self.data.items()}
23
24     def __len__(self):
25         """
26         Get the length of the dataset.
27
28         Returns:
29             int: The length of the dataset.
30         """
31         return len(self.data['input_ids'])
32
33
34 # Create a custom dataset for training
35 train_datas = CustomDataset(train_data)
36 print("Length of training dataset: {}".format(len(train_datas)))
37
38 # Create a data loader for training
39 train_loader = torch.utils.data.DataLoader(train_datas, batch_size=32, shuffle=True)
40
41 # Create a custom dataset for validation
42 validation_datas = CustomDataset(validation_data)
43 print("Length of validation dataset: {}".format(len(validation_datas)))
44
45 # Create a data loader for validation
46 validation_loader = torch.utils.data.DataLoader(validation_datas, batch_size=32, shuffle=False)
47
48 # Create a custom dataset for test
49 test_datas = CustomDataset(test_data)
50 print("Length of test dataset: {}".format(len(test_datas)))
51
52 # Create a data loader for test
53 test_loader = torch.utils.data.DataLoader(test_datas, batch_size=32, shuffle=False)
54 -----
55 Length of training dataset: 51797
56 Length of validation dataset: 8258
57 Length of test dataset: 8367

```

پس از آماده‌سازی کامل داده‌ها نوبت به تعریف مدل و ساختار آموزش می‌رسد. برای این هدف یک کلاس به نام QAModel تعریف می‌کنیم که از کلاس nn.Module ارث‌بری می‌کند. این کلاس مدلی را برای پرسش و پاسخ ایجاد می‌کند و مبتنی بر مدل برت (پارس‌برت) است. در این کلاس ابتدا مسیر و مشخصات مدل تعریف می‌شود. این کلاس دارای دو متد init و forward است. در متد init، ابتدا متغیر MODEL_NAME_OR_PATH با مسیر و مشخصات مربوط به مدل تعریف شده است. سپس متد سازنده کلاس فراخوانی می‌شود. در این متد، مدل با استفاده از from_pretrained ایجاد و مقادیر وزن‌های پیش‌آموزش

دیده مدل از مسیر مشخص شده بارگیری می‌شوند. سپس یک لایه خطی با ابعاد ورودی ۷۶۸ و خروجی ۲ برای دسته‌بندی (تعیین شروع و پایان) اضافه می‌شود. این لایه خطی به عنوان یک دسته‌بند در نهایت برای تشخیص شروع و پایان جواب در متن مورد استفاده قرار می‌گیرد. در متد forward، ورودی‌های input_ids، attention_mask و token_type_ids به مدل ارسال می‌شوند و سپس خروجی‌های sequence_output و pooled_output از مدل دریافت می‌شوند. sequence_output نشان‌دهنده خروجی هر توکن ورودی در متن است و pooled_output نشان‌دهنده خلاصه‌برداری از کل متن است. در ادامه لایه خطی اضافه‌شده به خروجی مدل اعمال می‌شود و خروجی‌ها با شکستن تابع $\text{split}(1, \text{dim}=-1)$ به دو بخش تقسیم می‌شوند: start_logits و end_logits. این بخش‌ها نشان‌دهنده امتیازهای شروع و پایان جواب در متن هستند. در نهایت، امتیازهای شروع و پایان به صورت مجزا به دست می‌آیند و خروجی تابع forward به عنوان امتیازهای شروع و پایان جواب برگردانده می‌شود. در ادامه تابع loss_fn به عنوان تابع محاسبه تابع هزینه برای مدل سوال و پاسخ استفاده می‌شود. این تابع چهار ورودی دارد: start_logits که امتیازهای پیش‌بینی شروع جواب به شکل ماتریس با ابعاد (batch_size, num_tokens) است. این ماتریس نشان می‌دهد هر توکن ممکن است شروع جواب باشد یا نه. end_logits که امتیازهای پیش‌بینی پایان جواب به شکل ماتریس با همان ابعاد است. این ماتریس نشان می‌دهد هر توکن ممکن است پایان جواب باشد یا نه. start_targets که موقعیت‌های واقعی شروع جواب برای مقادیر ورودی به شکل ماتریس با ابعاد (batch_size, num_tokens) است. end_targets که موقعیت‌های واقعی پایان جواب برای مقادیر ورودی به شکل ماتریس با ابعاد گفته‌شده است. این تابع مقدار هزینه را محاسبه می‌کند و یک مقدار تنسور به عنوان خروجی برمی‌گرداند. برای محاسبه اتلاف و هزینه، ابتدا از nn.BCEWithLogitsLoss استفاده می‌شود تا تابع خطای binary cross-entropy را روی امتیازهای پیش‌بینی شروع و پایان جواب و مقادیر واقعی شروع و پایان جواب محاسبه کند. این تابع خطای باینری برای مسائل دسته‌بندی دودویی استفاده می‌شود. سپس، دو مقدار خطا برای شروع و پایان جواب محاسبه می‌شوند: start_loss برای شروع جواب و end_loss برای پایان جواب. در نهایت، مقدار خطای شروع و پایان جواب با هم جمع شده و مقدار نهایی هزینه محاسبه شده را برمی‌گرداند. باید توجه داشت که تابع BCEWithLogitsLoss ابتدا امتیازها را با استفاده از تابع sigmoid به احتمالات تبدیل می‌کند و سپس خطای باینری را برای این احتمالات محاسبه می‌کند. همچنین، ورودی start_targets و end_targets به صورت عدد صحیح نیستند بلکه با استفاده از float به اعداد اعشاری تبدیل می‌شوند تا با خروجی sigmoid همخوانی داشته باشند. تابع evaluate_f1 هم برای محاسبه امتیاز F1 براساس مقادیر پیش‌بینی شروع و پایان جواب و مقادیر واقعی شروع و پایان جواب استفاده می‌شود. این تابع دارای چهار ورودی است: start_pred که موقعیت پیش‌بینی شروع جواب است. start_target که موقعیت واقعی شروع جواب است. end_pred که موقعیت پیش‌بینی پایان جواب است. end_target که موقعیت واقعی پایان جواب است. این تابع ابتدا آرایه‌هایی شامل توکن‌های متناظر با محدوده پیش‌بینی و محدوده واقعی شروع و پایان جواب را ایجاد می‌کند. این آرایه‌ها با استفاده از np.arange تولید می‌شوند. سپس، تعداد توکن‌های مشترک بین پیش‌بینی و واقعی شمارش می‌شود. برای این منظور، از توابع set.intersection و set.symmetric_difference استفاده می‌شود. با استفاده از تعداد توکن‌های مشترک و تعداد توکن‌هایی که در پیش‌بینی وجود دارند اما در واقعیت نیستند، و تعداد توکن‌هایی که در واقعیت وجود دارند اما در پیش‌بینی نیستند، مقادیر tp (تعداد True Positive)، fp (تعداد False Positive) و fn (تعداد False Negative) محاسبه می‌شوند. در ادامه، دقت، یادآوری و امتیاز F1 محاسبه می‌شوند. دقت نسبت تعداد توکن‌های صحیح پیش‌بینی شده به تعداد کل توکن‌های پیش‌بینی شده است. بازخوانی نسبت تعداد توکن‌های صحیح پیش‌بینی شده به تعداد کل توکن‌های واقعی است. امتیاز F1 نیز میانگین هندسی دقت و یادآوری است. در نهایت، امتیاز F1 محاسبه شده به عنوان خروجی برگردانده می‌شود. در ادامه تابع generate_indexes برای تولید شاخص‌های شروع و پایان برای محدوده‌های پیش‌بینی شده استفاده می‌شود. این تابع دارای چهار ورودی است: start_logits که امتیازهای پیش‌بینی شروع به شکل یک آرایه numpy است. end_logits که امتیازهای پیش‌بینی پایان است. N که تعداد شماره‌های شروع و پایان برتر برای بررسی.

`max_index_list` که لیستی از بزرگ‌ترین شاخص‌ها برای هر مثال است. این تابع ابتدا امتیازهای شروع و پایان را در متغیرهای `output_start` و `output_end` ذخیره می‌کند. سپس، ابعاد آرایه `output_start` را ذخیره می‌کند. بعد از آن، لیست‌های `list_start` و `list_end` را برای هر مثال ایجاد می‌کند. در هر مثال، شاخص‌های شروع و پایان را ایجاد می‌کند و احتمالات شروع و پایان را از `output_start` و `output_end` استخراج می‌کند. سپس، لیست‌های `list_start` و `list_end` را به شکل دیکشنری‌ها تبدیل کرده و به متغیرهای `list_start` و `list_end` اضافه می‌کند. در ادامه، لیست‌های `sorted_start_list` و `sorted_end_list` را برای هر مثال ایجاد می‌کند. این لیست‌ها شامل احتمالات شروع و پایان به ترتیب نزولی مرتب شده بر اساس احتمالات در `list_start` و `list_end` هستند. برای مرتب‌سازی، از تابع `sorted` با استفاده از `lambda` برای تعیین مقایسه‌گر استفاده می‌شود. در مرحله بعد، دو لیست جدید به نام `final_start_idx` و `final_end_idx` ایجاد می‌شوند. سپس برای هر مثال، متغیرهای `start_idx`، `end_idx` و `prob` را با مقدار صفر مقداردهی اولیه می‌کند. سپس با استفاده از دو حلقه تو در تو، اندیس‌های شروع و پایانی که جمع امتیاز آن‌ها بیشترین مقدار را دارد و شروع کوچکتری از پایان دارد و همچنین اندیس پایان کوچکتری از ماکزیمم اندیس ممکن برای مثال است، را پیدا می‌کند و آن‌ها را در متغیرهای `start_idx`، `end_idx` ذخیره می‌کند. در هر مرحله، اگر مجموع امتیاز شروع و پایان جدید بیشتر از `prob` باشد، مقدار `prob` و اندیس‌های شروع و پایان را به‌روزرسانی می‌کند. در انتها، اندیس‌های شروع و پایان را به لیست‌های `final_start_idx` و `final_end_idx` اضافه می‌کند و زوجی از این لیست‌ها را به‌عنوان زوج شاخص‌های شروع و پایان نهایی در خروجی تابع برمی‌گرداند. در انتها تابع ارزیابی را تعریف می‌کنیم که برای ارزیابی مدل با محاسبه امتیاز F1 استفاده می‌شود. این تابع شش ورودی دارد: `start_logits` که امتیازهای پیش‌بینی شروع به شکل یک آرایه با ابعاد `((batch_size, num_tokens))` است. این آرایه نشان می‌دهد هر توکن ممکن است شروع جواب باشد یا نه. `end_logits` که امتیازهای پیش‌بینی پایان به شکل یک آرایه است. این آرایه نشان می‌دهد هر توکن ممکن است پایان جواب باشد یا نه. `N` که تعداد بالایی اندیس‌های شروع و پایان برای محاسبه در نظر گرفته شده است. درواقع تعداد شماره‌های شروع و پایان برتر برای بررسی. `max_index_list` که لیستی از اندیس‌های حداکثر برای هر مثال است. `target_start` لیستی از اندیس‌های شروع هدف است. `target_end` لیستی از اندیس‌های پایان هدف است. این تابع ابتدا با استفاده از تابع `generate_indexes` اندیس‌های شروع و پایان نهایی برای بخش‌های پیش‌بینی شده را محاسبه می‌کند و در متغیرهای `final_start_idx` و `final_end_idx` ذخیره می‌کند. سپس یک لیست به نام `f1` ایجاد می‌شود و برای هر مثال، از تابع `evaluate_f1` استفاده می‌شود تا امتیاز F1 برای اندیس‌های شروع و پایان نهایی و اندیس‌های شروع و پایان هدف محاسبه شود. این امتیازها به لیست `f1` اضافه می‌شوند. در انتها، میانگین امتیازهای F1 محاسبه شده را با استفاده از تابع `np.mean` محاسبه کرده و به عنوان خروجی تابع برمی‌گردانیم. با ارائه این توضیحات مفصل دستورات مربوطه در برنامه ۹ آمده است.

Program 9: QAModel, Loss, Metrics, Evaluation, ...

```

1 MODEL_NAME_OR_PATH = 'HooshvareLab/bert-base-parsbert-uncased'
2
3 class QAModel(nn.Module):
4     def __init__(self):
5         super(QAModel, self).__init__()
6
7         # Initialize the BERT model
8         self.bert = BertModel.from_pretrained(MODEL_NAME_OR_PATH, return_dict=False)
9         # Add a linear layer for classification
10        self.classifier = nn.Linear(768, 2)
11
```

```

12 def forward(self, input_ids, attention_mask, token_type_ids):
13     # Pass the input through the BERT model
14     sequence_output, pooled_output = self.bert(
15         input_ids=input_ids,
16         attention_mask=attention_mask,
17         token_type_ids=token_type_ids)
18
19     # Apply linear layer to the BERT output
20     # Shape: (batch_size, num_tokens, 768)
21     logits = self.classifier(sequence_output)
22     # Shape: (batch_size, num_tokens, 2)
23
24     # Split the logits into start and end logits
25     start_logits, end_logits = logits.split(1, dim=-1)
26     start_logits = start_logits.squeeze(-1)
27     end_logits = end_logits.squeeze(-1)
28     # Shape: (batch_size, num_tokens), (batch_size, num_tokens)
29
30     return start_logits, end_logits
31
32
33 def loss_fn(start_logits, end_logits, start_targets, end_targets):
34     """
35     Compute the loss function given the predicted start and end logits and the target start and
36     end positions.
37
38     Args:
39         start_logits (torch.Tensor): Predicted start logits of shape (batch_size, num_tokens).
40         end_logits (torch.Tensor): Predicted end logits of shape (batch_size, num_tokens).
41         start_targets (torch.Tensor): Target start positions of shape (batch_size, num_tokens).
42         end_targets (torch.Tensor): Target end positions of shape (batch_size, num_tokens).
43
44     Returns:
45         torch.Tensor: Loss value.
46     """
47     # Compute the binary cross-entropy loss for start and end logits
48     start_loss = nn.BCEWithLogitsLoss()(start_logits, start_targets.float())
49     end_loss = nn.BCEWithLogitsLoss()(end_logits, end_targets.float())
50
51     # Return the sum of the two losses
52     return start_loss + end_loss
53
54 def evaluate_f1(start_pred, start_target, end_pred, end_target):
55     """
56     Compute the F1 score given the predicted start and end positions and the target start and end

```

```

        positions.
56
57     Args:
58         start_pred (int): Predicted start position.
59         start_target (int): Target start position.
60         end_pred (int): Predicted end position.
61         end_target (int): Target end position.
62
63     Returns:
64         float: F1 score.
65     """
66     # Generate arrays of tokens for prediction and target spans
67     pred = np.arange(start_pred, end_pred + 1)
68     tar = np.arange(start_target, end_target + 1)
69
70     # Compute the number of tokens shared between prediction and target
71     tp_list = list(set.intersection(*map(set, [pred, tar])))
72
73     # Compute the number of tokens in prediction not in target
74     fp_list = list(set(pred).symmetric_difference(set(tp_list)))
75
76     # Compute the number of tokens in target not in prediction
77     fn_list = list(set(tar).symmetric_difference(set(tp_list)))
78
79     tp, fp, fn = len(tp_list), len(fp_list), len(fn_list)
80
81     # Compute precision, recall, and F1 score
82     if (tp + fp) != 0:
83         precision = tp / (tp + fp)
84     else:
85         precision = 0
86
87     if (tp + fn) != 0:
88         recall = tp / (tp + fn)
89     else:
90         recall = 0
91
92     if (precision + recall) != 0:
93         f1 = (2 * precision * recall) / (precision + recall)
94     else:
95         f1 = 0
96
97     return f1
98
99

```



```

100 def generate_indexes(start_logits, end_logits, N, max_index_list):
101     """
102     Generate the start and end indexes for the predicted spans.
103
104     Args:
105         start_logits (numpy.ndarray): Predicted start logits of shape (batch_size, num_tokens).
106         end_logits (numpy.ndarray): Predicted end logits of shape (batch_size, num_tokens).
107         N (int): Number of top start and end indexes to consider.
108         max_index_list (list): List of maximum indexes for each example.
109
110     Returns:
111         tuple: Final start and end indexes for the predicted spans.
112     """
113     output_start = start_logits
114     output_end = end_logits
115     dimension = output_start.shape[1]
116
117     list_start, list_end = [], []
118     for n in range(output_start.shape[0]):
119         start_indexes = np.arange(output_start.shape[1])
120         start_probs = output_start[n]
121         list_start.append(dict(zip(start_indexes, start_probs)))
122
123         end_indexes = np.arange(output_end.shape[1])
124         end_probs = output_end[n]
125         list_end.append(dict(zip(end_indexes, end_probs)))
126
127     sorted_start_list, sorted_end_list = [], []
128     for j in range(len(list_start)):
129         sort_start_probs = sorted(list_start[j].items(), key=lambda x: x[1], reverse=True)
130         sort_end_probs = sorted(list_end[j].items(), key=lambda x: x[1], reverse=True)
131         sorted_start_list.append(sort_start_probs)
132         sorted_end_list.append(sort_end_probs)
133
134     final_start_idx, final_end_idx = [], []
135
136     for c in range(len(list_start)):
137         start_idx, end_idx, prob = 0, 0, 0
138         for a in range(N):
139             for b in range(N):
140                 if (sorted_start_list[c][a][1] + sorted_end_list[c][b][1]) > prob:
141                     if (sorted_start_list[c][a][0] <= sorted_end_list[c][b][0]) and (
142                         sorted_end_list[c][b][0] < max_index_list[c]):
143                         prob = sorted_start_list[c][a][1] + sorted_end_list[c][b][1]
144                         start_idx = sorted_start_list[c][a][0]

```

```

145         end_idx = sorted_end_list[c][b][0]
146         final_start_idx.append(start_idx)
147         final_end_idx.append(end_idx)
148
149     return final_start_idx, final_end_idx
150
151
152 def evaluate_model(start_logits, end_logits, N, max_index_list, target_start, target_end):
153     """
154     Evaluate the model by computing the F1 score.
155
156     Args:
157         start_logits (numpy.ndarray): Predicted start logits of shape (batch_size, num_tokens).
158         end_logits (numpy.ndarray): Predicted end logits of shape (batch_size, num_tokens).
159         N (int): Number of top start and end indexes to consider.
160         max_index_list (list): List of maximum indexes for each example.
161         target_start (list): List of target start positions.
162         target_end (list): List of target end positions.
163
164     Returns:
165         float: Mean F1 score.
166     """
167     final_start_idx, final_end_idx = generate_indexes(start_logits, end_logits, N, max_index_list
168     )
169     f1 = []
170     for i in range(len(final_start_idx)):
171         f1.append(evaluate_f1(final_start_idx[i], target_start[i], final_end_idx[i], target_end[i]
172         ))
173
174     return np.mean(f1)

```

در ادامه برای آماده‌سازی و تنظیمات مربوط به آموزش یک مدل پرسش و پاسخ دستوراتی را می‌نویسیم. ابتدا با استفاده از `torch.cuda.empty_cache` از حافظه GPU استفاده کرده و آن را پاک می‌کنیم. این کار برای آزاد کردن حافظه GPU مفید است. سپس دستگاه مورد استفاده برای آموزش را تعیین می‌کنیم. دستور به گونه‌ای نوشته شده که اگر GPU در دسترس باشد، دستگاه را روی GPU تنظیم می‌کند، در غیر این صورت از CPU استفاده می‌کند. در ادامه دستوراتی نوشته و تنظیمات مربوط به مدل پارس‌برت را از مسیر مربوطه بارگیری می‌کنیم. این تنظیمات شامل اطلاعاتی مانند تعداد لایه‌ها، اندازه حجم نهان، تعداد سر و غیره است. سپس یک نمونه از کلاس مدل پرسش و پاسخ معرفی شده در بالا را با استفاده از تنظیمات بارگیری شده ایجاد می‌کنیم. این مدل برای آموزش و پیش‌بینی پرسش و پاسخ استفاده می‌شود. در ادامه مدل را به دستگاه مورد استفاده (GPU یا CPU) منتقل می‌کنیم و با استفاده از `model.train` مدل را در حالت آموزش قرار می‌دهیم. این به مدل می‌گوید که در حین آموزش اطلاعات به روزرسانی شوند. در نهایت یک نمونه از بهینه‌ساز AdamW را با استفاده از پارامترهای قابل آموزش مدل و نرخ یادگیری مشخص ایجاد می‌کنیم. این بهینه‌ساز برای به‌روزرسانی وزن‌ها در حین آموزش استفاده می‌شود. دستورات مذکور به شرح زیر است:

```

1 from transformers import AutoConfig, AutoTokenizer

```

```

2
3 # Clear GPU cache
4 torch.cuda.empty_cache()
5
6 # Determine the device (GPU or CPU)
7 device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
8
9 # Load the configuration for the BERT model
10 config = AutoConfig.from_pretrained("HooshvareLab/bert-base-parsbert-uncased")
11
12 # Create an instance of the QAModel2 using the loaded configuration
13 model = QAModel2(config)
14
15 # Move the model to the appropriate device (GPU or CPU)
16 model.to(device)
17
18 # Set the model to training mode
19 model.train()
20
21 # Define the optimizer
22 optim = AdamW(model.parameters(), lr=5e-5)

```

در نهایت دستوراتی را برای آموزش مدل می‌نویسیم. ابتدا تعداد دوره‌های آموزش را با متغیر `n_epochs` تعیین می‌کنیم. سپس برای انجام درست محاسبات لازم تعداد دسته‌های آموزش و اعتبارسنجی را مشخص می‌کنیم. سپس با تعریف یک دستور یک نمونه از تابع `softmax` را ایجاد می‌کنیم که برای محاسبه احتمالات نرمال‌شده در خروجی استفاده می‌شود. بعد از اعمال `softmax`، مجموع احتمالات هر ردیف برابر با ۱ خواهد شد. سپس با قراردادن مقدار اولیه بهترین تلفات اعتبارسنجی برابر با بی‌نهایت، این متغیر را برای استفاده در ذخیره‌کردن مدلی که دارای کمترین تلفات اعتبارسنجی است آماده‌سازی می‌کنیم. در ادامه لیست‌هایی خالی برای محاسبات متریک‌ها و اتلاف‌ها ایجاد می‌کنیم. در ادامه حلقه مربوط به آموزش را خواهیم داشت. ابتدا یک شیء `tqdm` برای نمایش نوار پیشرفت در حلقه آموزش ایجاد می‌کنیم. سپس متغیرهایی برای جمع تلفات آموزش و اعتبارسنجی در هر دوره تعریف می‌کنیم و آن‌ها را به صفر مقداردهی اولیه می‌کنیم. در ادامه دستور گرادین‌ها را صفر می‌کنیم و داده‌ها را به دستگاه منتقل می‌کنیم (شروع). سپس داده‌های ورودی را به دسته‌هایی که بارگذاری شده است، منتقل می‌کنیم و خروجی شبکه را با استفاده از دسته‌های ورودی محاسبه می‌کنیم. در ادامه تلفات را با استفاده از خروجی‌های مدل و برچسب‌های مورد نظر محاسبه می‌کنیم. و تلفات محاسبه‌شده در هر گام را به تلفات آموزش کنونی اضافه می‌کنیم. گرادین‌ها را هم محاسبه کرده و اوزان را به تناسب به‌روز می‌کنیم. در ادامه با دسترسی به معنای عدم محاسبه گرادین بخش اعتبارسنجی را آغاز می‌کنیم. محاسبات این بخش مشابه بخش قبل است اما دیگر به‌روزرسانی نداریم. در ادامه احتمالات نرمال‌شده را با استفاده از تابع `softmax` محاسبه می‌کنیم. تلفات را هم با استفاده از خروجی‌های مدل و برچسب‌های مورد نظر در اعتبارسنجی محاسبه می‌کنیم و سپس امتیازهای تطبیق دقیق و `F1` را به دست می‌آوریم. دستورهای نوشته شده که برچسب‌های شروع و پایان واقعی را استخراج می‌کنند. دستوری هم برای ایجاد تانسور برچسب‌ها نوشته شده. مکان‌های شروع و پایان پاسخ را هم با استفاده از خروجی‌های مدل، تعداد بیشینه نقاط مورد نظر و نقاط جداکننده محاسبه می‌کنیم. در ادامه تانسور پیش‌بینی‌ها را ایجاد می‌کنیم و امتیازهای محاسبه‌شده را به لیست مربوطه‌اش می‌افزاییم. `input_ids` متغیر داده‌های ورودی متن را نشان می‌دهد. این داده‌ها به شکل یک بردار شامل شناسه‌های توکن‌های متن است. `attention_mask` متغیر ماسک توجه را نشان می‌دهد. این

ماسک برای تعیین کدام بخش‌های دنباله متنی باید توجه بیشتری شود و کدام بخش‌ها باید نادیده گرفته شوند استفاده می‌شود. `token_type_ids` متغیر شناسه‌های نوع توکن را نشان می‌دهد. این متغیر برای تمایز دادن بین دو بخش مختلف دنباله، به عنوان مثال جمله اول و جمله دوم در مدل‌های ترجمه ماشینی استفاده می‌شود. `y_start` متغیر شروع مطلب مورد نظر در دنباله متنی را نشان می‌دهد. این داده برای آموزش مدل سوال-پاسخ استفاده می‌شود. `y_end` هم متغیر پایان مطلب مورد نظر در دنباله متنی را نشان می‌دهد. این داده نیز برای آموزش مدل سوال-پاسخ استفاده می‌شود. برای محاسبات شاخص `exact match`، `y_start` و `y_end` برچسب‌های واقعی شروع و پایان جواب مربوط به داده فعلی هستند. ابتدا با استفاده از `np.nonzero` موقعیت‌های غیرصفر برچسب‌ها را دریافت کرده و به آرایه‌های `start_label` و `end_label` تبدیل می‌کنیم. سپس با استفاده از تابع `generate_indexes`، موقعیت شروع و پایان جواب پیش‌بینی شده را با توجه به احتمالات محاسبه‌شده (`start_pred` و `end_pred`) و نشانه‌گذاری [SEP] در داده ورودی محاسبه می‌کنیم. نتیجه به عنوان `start_log` و `end_log` به دست می‌آید. سپس این آرایه‌ها را به داده‌های نمونه برچسب تبدیل می‌کنیم که شامل دو ستون متناظر با شروع و پایان جواب است. این داده‌ها را با `tensor_label` ترکیب می‌کنیم. در نهایت، با استفاده از یک حلقه `for`، `exact match` را برای هر نمونه محاسبه می‌کنیم. اگر شروع و پایان جواب پیش‌بینی شده با شروع و پایان برچسب واقعی برابر باشد، یکی به مقدار آن اضافه می‌شود (در غیر این صورت صفر). در نهایت، میانگین این شاخص برای هر دسته نمونه محاسبه شده و به لیست مربوط به آن اضافه می‌شود. با ارائه این توضیحات دستورات مربوطه به شرح برنامه ۱۰ است.

Program 10: Training Code

```
1 import matplotlib.pyplot as plt
2
3 # Set the number of epochs
4 n_epochs = 5
5
6 # Get the number of batches in the training and validation sets
7 n_train_batches = len(train_loader)
8 n_validation_batches = len(validation_loader)
9
10 # Initialize softmax function
11 softmax = torch.nn.Softmax(dim=1)
12
13 # Initialize best validation loss
14 best_valid_loss = float('inf')
15
16 # Initialize lists to store loss, EM, and F1 scores for each epoch
17 train_losses, valid_losses = [], []
18 exact_match_scores, f1_scores = [], []
19
20 # Main training loop
21 for epoch in range(n_epochs):
22
23     loop = tqdm(train_loader)
24     train_running_loss, validation_running_loss = 0.0, 0.0
25     exact_match = []
26     f1 = []
```

```

27
28 # Training phase
29 for batch in loop:
30
31     # Zero the gradients
32     optim.zero_grad()
33
34     # Move data to device
35     input_ids = batch['input_ids'].to(device)
36     attention_mask = batch['attention_mask'].to(device)
37     token_type_ids = batch['token_type_ids'].to(device)
38     y_start = batch['targets_start'].to(device)
39     y_end = batch['targets_end'].to(device)
40
41     # Forward pass
42     out_start, out_end = model(input_ids, attention_mask=attention_mask, token_type_ids=
token_type_ids)
43
44     # Compute loss
45     loss = loss_fn(out_start, out_end, y_start, y_end)
46     train_running_loss += loss
47
48     # Backward pass
49     loss.backward()
50
51     # Update weights
52     optim.step()
53
54     loop.set_description(f'Epoch {epoch+1} - Training')
55
56 # Validation phase
57 with torch.no_grad():
58     loop2 = tqdm(validation_loader)
59     for content in loop2:
60
61         # Move data to device
62         input_ids = content['input_ids'].to(device)
63         temp_ids = input_ids.cpu().data.numpy().tolist()
64         max_ind = [temp_ids[i].index(4) if 4 in temp_ids[i] else -1 for i in range(len(
temp_ids))] # index of first [sep]
65         attention_mask = content['attention_mask'].to(device)
66         token_type_ids = content['token_type_ids'].to(device)
67         y_start = content['targets_start'].to(device)
68         y_end = content['targets_end'].to(device)
69

```

```

70         # Forward pass
71         out_start, out_end = model(input_ids, attention_mask=attention_mask, token_type_ids=
token_type_ids)
72         start_pred = softmax(out_start)
73         end_pred = softmax(out_end)
74
75         # Compute loss
76         loss2 = loss_fn(out_start, out_end, y_start, y_end)
77         validation_running_loss += loss2
78
79         # Compute EM and F1 scores
80         start_label = np.nonzero(y_start).cpu().data.numpy()
81         end_label = np.nonzero(y_end).cpu().data.numpy()
82         tensor_label = torch.stack((torch.tensor(start_label[:, 1]), torch.tensor(end_label
[:, 1])), -1)
83         start_log, end_log = generate_indexes(start_pred, end_pred, N, max_ind)
84         start_log = np.array(start_log)
85         end_log = np.array(end_log)
86         tensor_pred = torch.stack((torch.tensor(start_log), torch.tensor(end_log)), -1)
87         ex_ma = sum([1 if (tensor_pred[i][0] == tensor_label[i][0] and tensor_pred[i][1] ==
tensor_label[i][1]) else 0 for i in range(0, len(tensor_pred))]) / len(tensor_pred)
88         exact_match.append(ex_ma)
89         f1.append(evaluate_model(start_pred.cpu().data.numpy(), end_pred.cpu().data.numpy(),
N, max_ind, start_label[:, 1], end_label[:, 1]))
90         loop2.set_description(f'Epoch {epoch+1} - Validation')
91
92         # Compute average losses and scores
93         train_loss = train_running_loss / n_train_batches
94         valid_loss = validation_running_loss / n_validation_batches
95
96         # Append losses to the lists
97         train_losses.append(train_loss)
98         valid_losses.append(valid_loss)
99
100        # Update best validation loss and save model weights
101        if valid_loss < best_valid_loss:
102            best_valid_loss = valid_loss
103            torch.save(model.state_dict(), '/content/drive/MyDrive/HW5/Q2/Models/Model1.pt')
104            print('Model Saved in Google Drive!')
105
106        # Compute and append average EM and F1 scores
107        exact_match_score = 100 * np.mean(exact_match)
108        f1_score = 100 * np.mean(f1)
109        exact_match_scores.append(exact_match_score)
110        f1_scores.append(f1_score)

```

```

111
112     # Print epoch statistics
113     print('Epoch {}: Train Loss: {:.5f}, Valid Loss: {:.5f}, Exact Match: {:.2f}%, F1: {:.2f}%'.
114           format(epoch + 1,
115
116                 train_loss,
117
118                 valid_loss,
119
120                 exact_match_score,
121
122                 f1_score))
123 # Plotting the results
124 epochs = range(1, n_epochs + 1)
125
126 # Plotting training and validation losses
127 plt.figure()
128 plt.plot(epochs, train_losses, label='Train')
129 plt.plot(epochs, valid_losses, label='Validation')
130 plt.xlabel('Epochs')
131 plt.ylabel('Loss')
132 plt.title('Training and Validation Loss')
133 plt.legend()
134 plt.savefig('loss_plot.pdf')
135 plt.show()
136
137 # Plotting Exact Match scores
138 plt.figure()
139 plt.plot(epochs, exact_match_scores)
140 plt.xlabel('Epochs')
141 plt.ylabel('Exact Match (%)')
142 plt.title('Exact Match Score')
143 plt.savefig('exact_match_plot.pdf')
144 plt.show()
145
146 # Plotting F1 scores
147 plt.figure()
148 plt.plot(epochs, f1_scores)
149 plt.xlabel('Epochs')
150 plt.ylabel('F1 Score (%)')
151 plt.title('F1 Score')
152 plt.savefig('f1_score_plot.pdf')
153 plt.show()

```

نتیجه آموزش پارس‌برت با نرخ یادگیری 0.00005 و 0.0005 در برنامه ۱۱ آورده شده است:

Program 11: Training Results

```

1 LR = 0.00005:
2 Epoch 1 - Training:      100%|| 1619/1619 [35:16<00:00,  1.31s/it]
3 Epoch 1 - Validation:    100%|| 259/259 [10:53<00:00,  2.52s/it]
4 Model Saved in Google Drive!
5 Epoch 1: Train Loss: 0.01552, Valid Loss: 0.01891, Exact Match: 32.72%, F1: 70.76%
6 Epoch 2 - Training:      100%|| 1619/1619 [35:16<00:00,  1.31s/it]
7 Epoch 2 - Validation:    100%|| 259/259 [10:41<00:00,  2.48s/it]
8 Epoch 2: Train Loss: 0.01209, Valid Loss: 0.01946, Exact Match: 34.74%, F1: 72.29%
9 Epoch 3 - Training:      100%|| 1619/1619 [35:17<00:00,  1.31s/it]
10 Epoch 3 - Validation:    100%|| 259/259 [10:46<00:00,  2.50s/it]
11 Epoch 3: Train Loss: 0.00930, Valid Loss: 0.02048, Exact Match: 34.31%, F1: 71.04%
12 Epoch 4 - Training:      100%|| 1619/1619 [35:17<00:00,  1.31s/it]
13 Epoch 4 - Validation:    100%|| 259/259 [10:56<00:00,  2.53s/it]
14 Epoch 4: Train Loss: 0.00718, Valid Loss: 0.02417, Exact Match: 34.58%, F1: 72.34%
15 Epoch 5 - Training:      100%|| 1619/1619 [35:17<00:00,  1.31s/it]
16 Epoch 5 - Validation:    100%|| 259/259 [10:56<00:00,  2.53s/it]
17 Epoch 5: Train Loss: 0.00576, Valid Loss: 0.02588, Exact Match: 33.40%, F1: 71.34%
18
19 LR = 0.0005:

```

نتیجه آموزش آلبرت با نرخ یادگیری 0.00005 و 0.0005 در برنامه ۱۲ آورده شده است:

Program 12: Training Results

```

1 Epoch 1 - Training:      100%|| 1724/1724 [41:45<00:00,  1.45s/it]
2 Epoch 1 - Validation:    100%|| 283/283 [12:17<00:00,  2.61s/it]
3 Model Saved in Google Drive!
4 Epoch 1: Train Loss: 0.04189, Valid Loss: 0.04076, Exact Match: 19.16%, F1: 19.16%
5 Epoch 2 - Training:      100%|| 1724/1724 [41:45<00:00,  1.45s/it]
6 Epoch 2 - Validation:    100%|| 283/283 [12:10<00:00,  2.58s/it]
7 Model Saved in Google Drive!
8 Epoch 2: Train Loss: 0.03870, Valid Loss: 0.04065, Exact Match: 19.16%, F1: 19.16%

```

۴.۱ پاسخ قسمت ۴ - ارزیابی و پس پردازش

در ادامه دستورات آورده شده در پاسخ قسمت ۳ - پیاده سازی مدل و برای بخش ارزیابی و پس پردازش ابتدا تابع `evaluate_model` را تعریف می کنیم که برای ارزیابی مدل با محاسبه امتیاز F1 استفاده می شود. این تابع شش ورودی دارد: `start_logits` که امتیازهای پیش بینی شروع به شکل یک آرایه با ابعاد `((batch_size, num_tokens))` است. این آرایه نشان می دهد هر توکن ممکن است شروع جواب باشد یا نه. `end_logits` که امتیازهای پیش بینی پایان به شکل یک آرایه است. این آرایه نشان می دهد هر توکن ممکن است پایان جواب باشد یا نه. `N` که تعداد بالایی اندیس های شروع و پایان برای محاسبه در نظر گرفته شده است. درواقع تعداد شماره های شروع و پایان برتر برای بررسی. `max_index_list` که لیستی از اندیس های حداکثر برای هر مثال است. `target_start` لیستی از اندیس های شروع هدف است. `target_end` لیستی از اندیس های پایان هدف است. این

تابع ابتدا با استفاده از تابع `generate_indexes` اندیس‌های شروع و پایان نهایی برای بخش‌های پیش‌بینی شده را محاسبه می‌کند و در متغیرهای `final_start_idx` و `final_end_idx` ذخیره می‌کند. سپس یک لیست به نام `f1` ایجاد می‌شود و برای هر مثال، از تابع `evaluate_f1` استفاده می‌شود تا امتیاز `F1` برای اندیس‌های شروع و پایان نهایی و اندیس‌های شروع و پایان هدف محاسبه شود. این امتیازها به لیست `f1` اضافه می‌شوند. در انتها، میانگین امتیازهای `F1` محاسبه شده را با استفاده از تابع `np.mean` محاسبه کرده و به عنوان خروجی تابع برمی‌گردانیم. در ادامه دستوراتی را برای ارزیابی عملکرد مدل بر روی داده‌های تست می‌نویسیم. ابتدا با دستور `model.eval` مدل را در حالت ارزیابی قرار می‌دهیم. این کار معمولاً شامل غیرفعال کردن برخی عملیات‌هایی است که در حالت آموزش مدل استفاده می‌شوند. حالت ارزیابی به معنای استفاده از مدل برای پیش‌بینی و ارزیابی بدون به‌روزرسانی وزن‌ها است. در ادامه و با دستور `torch.no_grad with` محدودده‌ای را مشخص می‌کنیم که در آن گرادیان‌ها غیرفعال هستند. این کار منجر به کاهش مصرف حافظه و سرعت بالاتر اجرای کد در فرآیند ارزیابی می‌شود. داده‌های ورودی و برجسب‌های متناظر را از دسته‌های داده دریافت می‌کنیم و آن‌ها را به دستگاه مورد نیاز برای محاسبات (GPU) انتقال می‌دهیم. در ادامه دستوراتی را برای انجام عملیات پیش‌بینی با استفاده از مدل روی داده‌های ورودی از مدل استفاده می‌کنیم. این عملیات شامل انتشار جلو است که خروجی شروع و پایان پاسخ را محاسبه می‌کند. با استفاده از تابع `خطا`، مقدار خطا برای دسته فعلی محاسبه می‌شود و به مجموع خطا اضافه می‌شود. سپس با استفاده از برجسب‌ها و پیش‌بینی‌های مدل، معیارهای ارزیابی مانند `exact match` و `f1 score` برای دسته‌های فعلی محاسبه می‌شود و در لیست مربوطه ذخیره می‌شود. با استفاده از مقادیر محاسبه شده، میانگین خطا و میانگین معیارها برای تمام دسته‌های تست محاسبه می‌شود و مقادیر مربوطه در خروجی نمایش داده می‌شوند. با ارائه این توضیحات مفصل دستورات مربوطه در برنامه ۱۳ آمده است.

Program 13: Test & Evaluation Code

```
1 def evaluate_model(start_logits, end_logits, N, max_index_list, target_start, target_end):
2     """
3     Evaluate the model by computing the F1 score.
4
5     Args:
6         start_logits (numpy.ndarray): Predicted start logits of shape (batch_size, num_tokens).
7         end_logits (numpy.ndarray): Predicted end logits of shape (batch_size, num_tokens).
8         N (int): Number of top start and end indexes to consider.
9         max_index_list (list): List of maximum indexes for each example.
10        target_start (list): List of target start positions.
11        target_end (list): List of target end positions.
12
13    Returns:
14        float: Mean F1 score.
15    """
16    final_start_idx, final_end_idx = generate_indexes(start_logits, end_logits, N, max_index_list
17    )
18    f1 = []
19    for i in range(len(final_start_idx)):
20        f1.append(evaluate_f1(final_start_idx[i], target_start[i], final_end_idx[i], target_end[i]
21        ))
22    return np.mean(f1)
23 import matplotlib.pyplot as plt
```

```

23
24 # Softmax function for probability calculation
25 softmax = torch.nn.Softmax(dim=1)
26
27 # Set the model in evaluation mode
28 model.eval()
29
30 # Initialize variables for loss and evaluation metrics
31 n_test_batches = len(test_loader)
32 loss = 0
33 exact_match = []
34 f1_scores = []
35
36 # Iterate over the test data
37 with torch.no_grad():
38     loop = tqdm(test_loader)
39     for content in loop:
40         # Move input tensors to the appropriate device
41         input_ids = content['input_ids'].to(device)
42         temp_ids = input_ids.cpu().data.numpy().tolist()
43         max_ind = [temp_ids[i].index(4) for i in range(0, len(temp_ids))] # index of first [SEP]
44         attention_mask = content['attention_mask'].to(device)
45         token_type_ids = content['token_type_ids'].to(device)
46         y_start = content['targets_start'].to(device)
47         y_end = content['targets_end'].to(device)
48
49         # Forward pass
50         out_start, out_end = model(input_ids, attention_mask=attention_mask, token_type_ids=
token_type_ids)
51         start_pred = softmax(out_start)
52         end_pred = softmax(out_end)
53
54         # Calculate the loss
55         loss_batch = loss_fn(out_start, out_end, y_start, y_end)
56         loss += loss_batch.item()
57
58         # Convert labels and predictions to numpy arrays
59         start_label = np.nonzero(y_start).cpu().data.numpy()
60         end_label = np.nonzero(y_end).cpu().data.numpy()
61         tensor_label = torch.stack((torch.tensor(start_label[:, 1]), torch.tensor(end_label[:,
1])), -1)
62         start_log, end_log = generate_indexes(start_pred, end_pred, N, max_ind)
63         start_log = np.array(start_log)
64         end_log = np.array(end_log)
65         tensor_pred = torch.stack((torch.tensor(start_log), torch.tensor(end_log)), -1)

```

```

66
67     # Calculate exact match and F1 scores
68     ex_ma = sum([1 if (tensor_pred[i][0] == tensor_label[i][0] and tensor_pred[i][1] ==
        tensor_label[i][1]) else 0
69                 for i in range(0, len(tensor_pred))]) / len(tensor_pred)
70     exact_match.append(ex_ma)
71
72     f1_score = evaluate_model(start_pred.cpu().data.numpy(), end_pred.cpu().data.numpy(),
73                             N, max_ind, start_label[:, 1], end_label[:, 1])
74     f1_scores.append(f1_score)
75
76     # Calculate average loss and metrics
77     valid_loss = loss / n_test_batches
78     em_accuracy = 100 * np.mean(exact_match)
79     f1_mean = 100 * np.mean(f1_scores)
80     print('Test Loss: {:.4f}, Exact Match: {:.2f}%, F1 Score: {:.2f}%'.format(valid_loss,
        em_accuracy, f1_mean))

```

نتیجه ارزیابی پارس‌برت با نرخ یادگیری 0.00005 و 0.0005 در برنامه ۱۴ آورده شده است:

Program 14: Test Results

```

1 LR = 0.00005:
2 100%|| 262/262 [10:45<00:00, 2.46s/it]Test Loss: 0.0265, Exact Match: 34.08%, F1 Score: 71.68%
3
4 LR = 0.0005:

```

نتیجه ارزیابی آلبرت با نرخ یادگیری 0.00005 و 0.0005 در برنامه ۱۵ آورده شده است:

Program 15: Test Results

```

1
2 100%|| 276/276 [12:17<00:00, 2.67s/it]Test Loss: 0.0416, Exact Match: 18.59%, F1 Score: 19.24%

```

نتیجه کلی تست در ؟؟ نشان داده شده است. قطعاً با تغییر مضاعف پارامترها و افزایش تعداد دوره‌ها می‌توانستیم به نتایج بهتری هم دست پیدا کنیم؛ ولی از آن‌جا که تمرین‌ها را یک‌نفره انجام می‌دهم و وقت کم‌تری دارم موفق به این مهم نشدم. مثلاً در بحث نرخ یادگیری عدد 0.0005 هم امتحان شد که نتایج بسیار بدتر می‌شد (امتیاز F1 حدود ۹ درصد). اما به‌عنوان راهی دیگر برای مدل آلبرت که نتایجش چنگی به دل نمی‌زد، از یکی از مطالب آماده‌شده بر بستر اینترنت استفاده کرده‌ام و ضمن فهم آن و افزودن کامنت آن را هم تست کردم. دستورات مربوطه در برنامه ۱۶ آورده شده است. لازم به ذکر است که قسمت‌هایی از این کد را تغییر دادم که مشخصاً و دقیقاً روی مجموعه داده صورت سوال کارایی داشته باشد و جواب دهد. این کدها ابتدا یک لودر برای بارگیری دیتاست‌ها فراهم می‌کند. برای این کار ابتدا کلاس DatasetLoader تعریف شده است که شامل متدها و متغیرهای مختلف است. این کلاس دارای یک متد سازنده است که مقادیر مورد نیاز را برای توکن‌سازی و بارگیری دیتاست دریافت می‌کند و به متدهای مربوطه ارجاع می‌دهد تا دیتاست مورد نظر را بارگیری کنند. در متد سازنده، مقدار dataset انتخاب شده بر اساس ورودی dataset تعیین می‌شود. سپس، با استفاده از دیکشنری dataset_to_loader، لودر مربوطه بر اساس دیتاست انتخاب شده فراخوانی می‌شود و دیتاست بارگیری می‌شود. متد extract_entries وظیفه استخراج داده‌ها از فایل JSON و تبدیل آن‌ها به یک دیتافریم در پانداس را دارد. این متد بر اساس یک نمونه از داده و محدودیت تعداد داده‌های استخراج شده، ورودی‌ها

را استخراج می‌کند و به صورت دیکشنری ذخیره می‌کند. متدهایی هم برای دریافت دیتاست‌های مختلف تعریف شده است که تمرکز ما روی PQuAD است. این متدها فایل‌های JSON مربوطه را بارگیری کرده و با استفاده از متد `extract_entries` داده‌های مربوط به هر بخش، `train` و `validation` (test) را استخراج و به دیتافریم تبدیل می‌کنند. سپس دیتافریم را به عنوان یک مجموعه داده در دیکشنری `dataset` ذخیره می‌کنند. متد `preprocess_function` وظیفه پیش‌پردازش داده‌ها را برای توکن‌سازی و آماده‌سازی داده‌های ورودی برای آموزش مدل تعیین می‌کند. این متد ورودی‌های مورد نیاز را از داده‌ها استخراج کرده و آن‌ها را توکن‌سازی می‌کند. سپس شروع و پایان موقعیت پاسخ‌ها را مشخص می‌کند و به عنوان ورودی نهایی برای آموزش مدل بازگردانده می‌شود. در ادامه کلاس `TrainerQA` برای آموزش و ارزیابی مدل سوال و پاسخ استفاده می‌شود. در مرحله اول، متد `init` توکن‌سازی، بارگیری دیتاست و بارگیری مدل را انجام می‌دهد. این متد از یک نقطه شروع (`model_checkpoint`) و دیتاست مورد استفاده (`dataset`) برای تنظیم مدل و بارگیری دیتاست استفاده می‌کند. متد `train` برای آموزش مدل استفاده می‌شود. ابتدا یک `data_collator` به عنوان مجموعه داده‌های آموزش راه‌اندازی می‌شود، سپس پارامترهای آموزش مانند تعداد دوره‌ها و نرخ یادگیری تنظیم می‌شوند. سپس `Trainer` با استفاده از مدل، پارامترهای آموزش، دیتاست‌های آموزش و ارزیابی و توکن‌سازی‌کننده راه‌اندازی می‌شود و مدل آموزش داده می‌شود. متد `evaluate` هم برای ارزیابی مدل استفاده می‌شود. ابتدا یک `pipeline` سوال و پاسخ با استفاده از مدل و توکن‌سازی‌کننده ساخته می‌شود. سپس سوال‌ها و متن‌ها از دیتاست ارزیابی استخراج می‌شوند و با استفاده از `pipeline` پاسخ‌ها تولید می‌شوند. سپس نتایج ارزیابی با استفاده از معیار `SQuAD` محاسبه می‌شوند و چاپ می‌شوند. متد `push_to_hub` برای آپلود مدل و توکن‌سازی‌کننده به `Hugging Face Model Hub` استفاده می‌شود.

Program 16: Training, Test & Evaluation Code (ALBERT - Method 2)

```
1 ! nvidia-smi
2 !pip install transformers[torch] # Install the transformers package with PyTorch support
3 !pip install accelerate -U # Install the accelerate package
4 ! pip install transformers datasets
5 ! pip install huggingface_hub
6 ! pip install sentencepiece
7 # ! gdown 1SLko03mD7bpV8B7UpcDeuPHIOF3bFLPp
8 ! unzip Dataset.zip
9 from datasets import load_dataset, load_metric
10 from transformers import AutoTokenizer, DefaultDataCollator, AutoModelForQuestionAnswering
11 from transformers import TrainingArguments, Trainer, create_optimizer, pipeline
12 import json
13 import pandas as pd
14 import datasets
15 from tqdm.auto import tqdm
16 from matplotlib import pyplot as plt
17 from huggingface_hub import notebook_login
18 notebook_login()
19
20 # Clone the PQuAD dataset from GitHub
21 !git clone https://github.com/AUT-NLP/PQuAD.git
22
23 # Load the PQuAD dataset
24 train_file_path = '/content/PQuAD/Dataset/Train.json'
```

```

25 test_file_path = '/content/PQuAD/Dataset/Test.json'
26 val_file_path = '/content/PQuAD/Dataset/Validation.json'
27
28 DATASETS = {
29     "pquad": 0, # Dataset key for pquad
30     "persian_qa": 1, # Dataset key for PersianQA
31     "parsquad": 2, # Dataset key for ParSQuAD
32     "pquad_and_persian_qa": 3 # Dataset key for both pquad and PersianQA
33 }
34
35 class DatasetLoader:
36     def __init__(self, dataset, tokenizer):
37         self.tokenizer = tokenizer # Tokenizer for preprocessing
38         self.dataset = datasets.DatasetDict() # Dictionary to store the datasets
39         dataset_to_loader = {
40             DATASETS["pquad"]: self.__load_pquad_public, # Load the pquad_public dataset
41             DATASETS["persian_qa"]: self.__load_persian_qa, # Load the PersianQA dataset
42             DATASETS["parsquad"]: self.__load_parsquad, # Load the ParSQuAD dataset
43             DATASETS["pquad_and_persian_qa"]: self.__load_pquad_and_persian_qa # Load both
44             pquad_public and PersianQA datasets
45         }
46         self.dataset = dataset_to_loader[dataset]() # Call the corresponding loader based on the
47             selected dataset
48         self.tokenized_dataset = self.dataset.map(self.preprocess_function, # Apply the
49             preprocess_function to tokenize and preprocess the dataset
50             batched=True,
51             remove_columns=self.dataset["train"].
52             column_names)
53
54     def __extract_entries(self, data, limit=200000000):
55         df_list = [] # List to store the extracted entries
56         c = 0 # Counter to keep track of the number of entries processed
57         length_distribution = [] # List to store the length of contexts
58         for d in tqdm(data['data'], desc="Converting json to dataset"): # Iterate over the data
59             entries
60             for p in d['paragraphs']: # Iterate over the paragraphs in the data entry
61                 length_distribution.append(len(p['context'].split())) # Record the length of the
62                 context
63                 for qas in p['qas']: # Iterate over the questions and answers in the paragraph
64                     c += 1 # Increment the counter
65                     if c > limit:
66                         return df_list # Return the extracted entries if the limit is reached
67                     if qas["is_impossible"]:
68                         continue # Skip the entry if it is impossible to answer
69                     df_list.append({

```

```

64         "id": str(qas['id']),
65         "title": d['title'],
66         "context": p['context'],
67         "question": qas['question'],
68         "answers": {"text": "", "answer_start": 0}
69     })
70     else:
71         for answer in qas['answers']: # Iterate over the answers
72             df_list.append({
73                 "id": str(qas['id']),
74                 "title": d['title'],
75                 "context": p['context'],
76                 "question": qas['question'],
77                 "answers": {"text": answer["text"], "answer_start": answer["
answer_start"]}}
78             })
79     plt.hist(length_distribution) # Plot the length distribution of contexts
80     plt.title("Length Distribution")
81     plt.show()
82     return df_list
83
84 def __load_pquad_public(self):
85     self.dataset = datasets.DatasetDict() # Clear the dataset dictionary
86     for part in ["train", "validation", "test"]:
87         with open(f"PQuAD/Dataset/{part}.json", 'r', encoding='utf-8') as f:
88             data = json.load(f) # Load the data from the JSON file
89             df_list = self.__extract_entries(data) # Extract entries from the data
90             self.dataset[part] = datasets.Dataset.from_pandas(pd.DataFrame.from_dict(df_list)) #
Convert the extracted entries to a dataset
91     return self.dataset
92
93 def __load_persian_qa(self):
94     self.dataset = datasets.DatasetDict() # Clear the dataset dictionary
95     part_to_path = {
96         "train": f"Dataset/PersianQA/pqa_train.json",
97         "validation": f"Dataset/PersianQA/pqa_test.json",
98         "test": f"Dataset/pquad_public/test_samples.json"
99     }
100     for part in ["train", "validation", "test"]:
101         with open(part_to_path[part], 'r', encoding='utf-8') as f:
102             data = json.load(f) # Load the data from the JSON file
103             df_list = self.__extract_entries(data) # Extract entries from the data
104             self.dataset[part] = datasets.Dataset.from_pandas(pd.DataFrame.from_dict(df_list)) #
Convert the extracted entries to a dataset
105     return self.dataset

```

```

106
107 def __load_parsquad(self):
108     self.dataset = datasets.DatasetDict() # Clear the dataset dictionary
109     part_to_path = {
110         "train": f"Dataset/ParSQuAD/ParSQuAD-manual-train.json",
111         "validation": f"Dataset/ParSQuAD/ParSQuAD-manual-dev.json",
112         "test": f"Dataset/pquad_public/test_samples.json"
113     }
114     for part in ["train", "validation", "test"]:
115         with open(part_to_path[part], 'r', encoding='utf-8') as f:
116             data = json.load(f) # Load the data from the JSON file
117             df_list = self.__extract_entries(data) # Extract entries from the data
118             self.dataset[part] = datasets.Dataset.from_pandas(pd.DataFrame.from_dict(df_list)) #
119             Convert the extracted entries to a dataset
120
121     return self.dataset
122
123 def __load_pquad_and_persian_qa(self):
124     self.dataset = datasets.DatasetDict() # Clear the dataset dictionary
125     part_to_path = {
126         "train": f"Dataset/PersianQA/pqa_train.json",
127         "validation": f"Dataset/PersianQA/pqa_test.json",
128         "test": f"Dataset/pquad_public/test_samples.json"
129     }
130     for part in ["train", "validation", "test"]:
131         with open(f"PQuAD/Dataset/{part}.json", 'r', encoding='utf-8') as f:
132             data = json.load(f) # Load the data from the JSON file
133             df_list = self.__extract_entries(data) # Extract entries from the data
134             if part != "test":
135                 with open(part_to_path[part], 'r', encoding='utf-8') as f:
136                     data = json.load(f) # Load the additional data from the JSON file
137                     df_list.extend(self.__extract_entries(data)) # Extend the extracted entries with
138                     additional entries
139             self.dataset[part] = datasets.Dataset.from_pandas(pd.DataFrame.from_dict(df_list)) #
140             Convert the extracted entries to a dataset
141
142     return self.dataset
143
144 def preprocess_function(self, examples):
145     questions = [q.strip() for q in examples["question"]] # Extract the questions
146     inputs = self.tokenizer(
147         questions,
148         examples["context"],
149         max_length=400, # Maximum length of the tokenized inputs
150         # max_length=250,
151         truncation="only_second", # Truncate the second sequence (context)
152         # return_overflowing_tokens=True,

```

```

148         return_offsets_mapping=True, # Return the character offsets of the tokens
149     #         padding="max_length",
150 )
151
152     offset_mapping = inputs.pop("offset_mapping") # Extract the offset mappings
153     answers = examples["answers"] # Extract the answers
154     start_positions = [] # List to store the start positions of the answers
155     end_positions = [] # List to store the end positions of the answers
156
157     for i, offset in enumerate(offset_mapping): # Iterate over the offset mappings
158         answer = answers[i] # Get the answer
159         start_char = answer["answer_start"] # Get the start character position of the answer
160         end_char = answer["answer_start"] + len(answer["text"]) # Get the end character
161         position of the answer
162         sequence_ids = inputs.sequence_ids(i) # Get the sequence IDs of the input tokens
163
164         # Find the start and end of the context
165         idx = 0
166         while sequence_ids[idx] != 1: # Find the start of the context
167             idx += 1
168         context_start = idx
169         while sequence_ids[idx] == 1: # Find the end of the context
170             idx += 1
171         context_end = idx - 1
172
173         # If the answer is not fully inside the context, label it (0, 0)
174         if offset[context_start][0] > end_char or offset[context_end][1] < start_char:
175             start_positions.append(0)
176             end_positions.append(0)
177         else:
178             # Otherwise it's the start and end token positions
179             idx = context_start
180             while idx <= context_end and offset[idx][0] <= start_char:
181                 idx += 1
182             start_positions.append(idx - 1)
183
184             idx = context_end
185             while idx >= context_start and offset[idx][1] >= end_char:
186                 idx -= 1
187             end_positions.append(idx + 1)
188
189     inputs["start_positions"] = start_positions # Add the start positions to the inputs
190     inputs["end_positions"] = end_positions # Add the end positions to the inputs
191     return inputs

```



```

192
193 class TrainerQA:
194     def __init__(self, model_checkpoint, dataset):
195         print("### Loading Tokenizer ###")
196         self.tokenizer = AutoTokenizer.from_pretrained(model_checkpoint) # Load the tokenizer
197         print("### Loading Dataset ###")
198         self.dataset_loader = DatasetLoader(dataset, self.tokenizer) # Load the dataset
199         print(self.dataset_loader.dataset) # Print the loaded dataset
200         print(self.dataset_loader.tokenized_dataset) # Print the tokenized dataset
201         print("### Loading Model ###")
202         self.model = AutoModelForQuestionAnswering.from_pretrained(model_checkpoint) # Load the
model
203
204     def train(self, num_train_epochs=3, learning_rate=2e-5):
205         print("### Training Model ###")
206         data_collator = DefaultDataCollator() # Data collator for handling the training data
207         training_args = TrainingArguments(
208             output_dir="./results",
209             evaluation_strategy="epoch",
210             learning_rate=learning_rate,
211             per_device_train_batch_size=16,
212             per_device_eval_batch_size=16,
213             num_train_epochs=num_train_epochs,
214             weight_decay=0.01,
215             group_by_length=True,
216             logging_steps=20
217         )
218         self.trainer = Trainer(
219             model=self.model,
220             args=training_args,
221             train_dataset=self.dataset_loader.tokenized_dataset["train"],
222             eval_dataset=self.dataset_loader.tokenized_dataset["validation"],
223             tokenizer=self.tokenizer,
224             # data_collator=data_collator,
225         ) # Initialize the Trainer with the model, training arguments, datasets, and tokenizer
226         self.trainer.train() # Train the model
227
228     def evaluate(self):
229         print("### Evaluating Model ###")
230         qa_model = pipeline("question-answering", model=self.model, tokenizer=self.tokenizer,
device=0) # Create a question-answering pipeline with the model and tokenizer
231         questions = self.dataset_loader.dataset["test"]["question"] # Get the questions from the
test dataset
232         contexts = self.dataset_loader.dataset["test"]["context"] # Get the contexts from the
test dataset

```

```

233     preds = qa_model(question=questions, context=contexts, device="cuda") # Generate
    predictions using the question-answering pipeline
234     print(questions[0]) # Print the first question
235     print(contexts[0]) # Print the first context
236     print(preds[0]) # Print the prediction for the first question-context pair
237
238     metric = load_metric("squad") # Load the SQuAD metric
239     last_id = -1
240     predictions, references = [], []
241     for i, answers in tqdm(enumerate(self.dataset_loader.dataset["test"]["answers"])):
242         if len(answers["text"]) < 1:
243             continue
244         id = self.dataset_loader.dataset["test"][i]["id"]
245         if id != last_id:
246             predictions.append({
247                 "id": id,
248                 "prediction_text": preds[i]["answer"].strip()
249             })
250             references.append({
251                 "id": id,
252                 "answers": []
253             })
254             last_id = id
255             references[-1]["answers"].append(answers)
256 #     predictions = [{"id": i, "prediction_text": pred["answer"].strip()} for i, pred in
    enumerate(preds) if len(trainer.dataset_loader.dataset["test"][i]["answers"]["text"]) > 0]
257 #     references=[{"id": i, "answers": [answers]} for i, answers in enumerate(trainer.
    dataset_loader.dataset["test"]["answers"]) if len(answers["text"]) > 0]
258     results = metric.compute(predictions=predictions, references=references) # Compute the
    SQuAD metric using the predictions and references
259     print(results) # Print the evaluation results
260     return results
261
262     def push_to_hub(self, name):
263         self.model.push_to_hub(name) # Push the model to the Hugging Face Model Hub
264         self.tokenizer.push_to_hub(name) # Push the tokenizer to the Hugging Face Model Hub

```

نتایج آلبرت در زیر آورده شده و بصورت کلب تمام نتایج در جدول ۴ نشان داده شده است.

Epoch	Training Loss	Validation Loss
1	0.852800	1.075374
2	0.686400	1.111044
3	0.455700	1.233197

{ 'exact_match': 56.48817345597897, 'f1': 77.83653745721735 }

جدول ۴: نتیجه‌ی ارزیابی مدل‌ها روی داده‌های آزمون

Model	EM	F1
ParsBERT	35.45	71.68
ALBERT	18.59, ... Best: 56.48	19.34, ... Best: 77.83