به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

درس پردازش زبان طبیعی

پاسخ تمرین ۴

نام و نام خانودگی: سامان اسلامی نظری

شماره دانشجویی: ۸۱۰۱۹۹۳۷۵

خرداد ماه ۱۴۰۳

فهرست

پاسخ سوال اول

دادگان

در این پروژه از مجموعه دادگان multi-nli استفاده می‌کنیم. این مجموعه داده برای تسک Natural Language Inference استفاده می‌شود که در آن ۴۳۳ هزار جفت جمله در کنار هم قرار داده شده‌اند؛ این مجموعه داده شامل ۳ نوع لیبل برای هر جفت جمله است: ۰ به معنی اینکه این دو جمله در پی یکدیگر ظاهر می‌شوند، ۱ به معنی خنثی و ۲ به معنی تناقض میان دو جمله داده شده است (یعنی این دو جمله پشت هم ظاهر نمی‌شوند). همچنین این محموعه داده شامل ژانرهای مختلفی از متون صحبت‌شده یا نوشته‌شده را شامل می‌شود.

ستون‌های مختلف دادگان به صورت زیر می‌باشد:

- جفت‌های premise و hypothesis: جملات ورودی هستند که در آن hypothesis پس از premise ظاهر می‌شود.
- جفت‌های premise و hypothesis به صورت پارس‌شده: همان جفت جملات که با استفاده از الگوریتم Stanford PCFG Parser 3.5.2 پارس شده‌اند.
- جفت‌های premise و hypothesis به صورت باینری پارس‌شده: همان جفت جملات که به صورت unlabeled binary-branched پارس شده
- Genre: نشان‌دهنده ژانر جملات آن سطر
- Label: یکی از سه لیبل توضیح داده‌شده

بخش اول – فاین تیون

دو روش سنتی‌تر فاین تیون و روش LORA

یک دسته‌بندی کلی برای روش‌های فاین تیون کردن به شرح زیر می‌باشد:

- اولین روش این است که اکثر وزن‌ها و لایه‌های مدل را ثابت نگه داشته و تنها زیرمجموعه‌ای از آن را مجددا برای تسک جدید آموزش دهیم. به این صورت، تمام چیزهایی که مدل از داده‌های قبلی آموخته را نگه داشته و با استفاده از آموزش‌های جدید، یاد گرفته تا نسبت به تسک جدید بهتر عمل کند. این کار از نظر هزینه و زمان بسیار به صرفه‌تر است چرا که نیازی نداریم کلا مدل را از اول آموزش دهیم و صرفا بخش کوچک‌تری از آن را آموزش می‌دهیم.
- روش دوم این است که مدل را از ابتدا و با داده‌های جدید آموزش دهیم. واضحا این روش نیازمند زمان و قدرت محاسباتی بیش‌تر است. با استناد به این مقاله آنلاین، این کار زمانی مناسب است که داده‌های زیادی برای تسک جدید در اختیار داریم و این داده‌های نسبت به داده‌های قدیمی‌ای که مدل پیش‌تر روی آن آموزش داده شده بود تفاوت فاحشی دارد.

برای شرح LoRA به این مقاله استناد می‌کنیم. LoRA مخفف Low Rank Adaptation است. رنک یک ماتریس نشان‌دهنده تعداد سطرها یا ستون‌هایی است که به صورت خطی از باقی سطرها و ستون‌ها مستقل می‌باشند. ما می‌توانیم سطرها یا ستون‌هایی که به صورت خطی به همدیگر وابسته‌اند را به صورت ضریبی از یک ماتریس کوچک‌تر نمایش دهیم. با استفاده از دانستن رنک یک ماتریس ما می‌توانیم عمل Rank Decomposition را انجام دهیم که در آن یک ماتریس را به ماتریس‌های کوچک‌تر می‌شکنیم؛ دلیل اینکه می‌توانیم این کار را انجام دهیم این است که سطرها یا ستون‌هایی که به صورت خطی به دیگر سطرها و ستون‌ها وابسته‌اند را می‌توانیم با یک سطر یا ستون و یک ضریب نمایش دهیم.

به صورت تجربی مشاهده شده که ماتریس‌های وزن از پیش آموخته‌شده رنک نسبتا کوچکی دارند و می‌توانیم آن‌ها را با تعداد پارامترهای کم‌تری نشان دهیم. فرض کنیم که وزن‌های از پیش آموخته‌شده W_0 باشند و پس از فاین تیون کردن، وزن‌ها به مقدار $W_0 + \triangle W$ می‌رسند. LoRA این فرض را می‌گیرد که اگر W_0 رنک کوچکی دارد پس $\triangle W$ نیز رنک کوچکی خواهد داشت. بنابراین می‌توانیم ماتریس آپدیت را به ماتریس‌های کوچکی تبدیل کنیم (Rank Decomposition). بنابراین LoRA با استفاده از این ایده تعداد پارامترهای مورد نیاز برای فاین تیون کردن مدل را کاهش می‌دهد.

روش‌های فاین تیونینگ مبتنی بر Prompt

به صورت کلی ما می‌توانیم به جای اینکه مدل را دوباره از اول آموزش دهیم، به متن ورودی آن یک سری متون از پیش تعیین‌شده اضافه کنیم تا مدل را به سمت تسک خاصی هدایت کنیم. برای این کار ما دو روش hard و soft را داریم:

- روش hard prompting: در این روش ما یک تمپلیت درست می‌کنیم و با استفاده از این تمپلیت و جا دادن ورودی دلخواه در آن، مدل را به سمت پاسخ درست‌تر و بهتر هدایت می‌کنیم. از آنجایی که تمپلیت‌ها به صورت دستی مهندسی می‌شوند، گاها پیدا کردن یک تمپلیت مناسب و در کل یافتن بهترین تمپلیت می‌تواند کار دشواری باشد.
- روش soft prompting: در این روش ما مدل اصلی را دست نزده و صرفا پارامترهای توکنی که جدیدا اضافه کردیم را آموزش می‌دهیم. در این حالت این پارامترها برای هر تسک جدید آموزش داده شده و یاد می‌گیرند که برای آن تسک به خصوص بهترین پرامپت‌ها را برای ورودی اصلی ایجاد کنند، به طوری که مدل اصلی به سمت جواب درست‌تر هدایت شود. بدین صورت ما نیازی نداریم که مانند روش قبلی پرامپت‌ها را به صورت دستی اضافه کنیم؛ یک مدل با پارامترهای کم‌تر نسبت به مدل اصلی این کار را برای ما بر عهده خواهد گرفت.

بخش دوم – آموزش مدل

۱. برای این بخش از مدل RobertaForSequenceClassification استفاده می‌کنیم. این مدل همان Roberta Large است که روی آن یک هد به منظور انجام تسک Sequence Classification قرار گرفته است. به دلیل محدودیت در منابع سخت‌افزاری، از کانفیگ زیر برای معماری این مدل استفاده کردیم:

بیشینه اندازه position embeddings: ۵۱۲

تعداد attention head-ها: ۱۲

تعداد hidden layer-ها: ۶

باقی موارد مشابه مدل اصلی Roberta می‌باشند. همچنین تعداد پارامترهای قابل آموزش نیز برابر ۸۲۱۱۹۹۳۹ می‌باشد. هایپرپارامترهای انتخاب‌شده نیز به صورت زیر می‌باشند:

Epoch: ۰.۰۰۰۲

تعداد epoch-ها: ۱۰

Weight decay: ۰.۰۱

اندازه batch-ها: ۶۴

زمان صرف‌شده جهت آموزش ۲۴۳۶ ثانیه بود. جدول accuracy در انتهای هر epoch از آموزش به صورت زیر می‌باشد:

[6140/6140 40:35, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy
1	1.088800	1.059160	0.400612
2	1.063700	1.053924	0.424057
3	1.053800	1.049788	0.446483
4	1.042000	1.059306	0.436290
5	1.019000	1.076406	0.420999
6	1.014500	1.043109	0.449541
7	1.004400	1.060505	0.436290
8	1.002300	1.048106	0.450561
9	0.992100	1.045917	0.451580
10	0.985700	1.049965	0.443425

۲. برای این مرحله نیز از معماری قبلی استفاده کردیم. دقت کنید که به دلیل آنکه خود مدل اصلی Roberta را بارگذاری نکردیم، وزن‌های موجود اشتباه بوده و در اصل آموزش‌داده نشده‌اند؛ بنابراین فاین تیون کردن این مدل نیز تا حدودی بیهوده به نظر می‌رسد؛ با این حال با این روش می‌توان تمام موارد خواسته‌شده را گزارش و مقایسه کرد، به جز accuracy دو مدل. تعداد پارامترهای قابل آموزش در این روش برابر ۷۴۰۳۳۵

بود. همچنین مدت زمان مورد نیاز برای آموزش ۲۴۲۵ ثانیه بود که تفاوت چندانی با روش قبلی نداشت. جدول accuracy در انتهای هر epoch به صورت زیر می‌باشد:

[6140/6140 40:24, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy
1	1.118700	1.099187	0.347604
2	1.096300	1.064775	0.403670
3	1.071400	1.058549	0.433231
4	1.061600	1.060972	0.423038
5	1.043800	1.072510	0.422018
6	1.036100	1.051604	0.436290
7	1.027200	1.051772	0.438328
8	1.024800	1.054530	0.442406
9	1.011900	1.051389	0.442406
10	1.007300	1.051572	0.441386

بخش سوم – چرا LoRA؟

از مزایای این روش می‌توان به موارد زیر اشاره کرد:

- کاهش منابع مورد استفاده: از آن جایی که اکثر مدل‌های ترنسفرمر رنک پایینی دارند، می‌توان آن‌ها را در فضاهای کوچک‌تری نمایش داد. در این حالت LoRA از این موضوع استفاده کرده و پارامترهای مدل را در فضای کوچک‌تری ذخیره می‌کند.
- آموزش مدل سریع‌تر می‌شود.

به جز روش‌هایی مثل آموزش دادن کل مدل از ابتدا، روش‌های دیگری نیز وجود دارند؛ یک راه feature-based می‌باشد. در این حالت ما مدل ترنسفرمری که از پیش آموزش داده شده را دست نمی‌زنیم و به جای آن، روی embedding-های خروجی از آن، یک مدل جدید آموزش می‌دهیم. روش دیگر in context prompting است. Zero-shot و few-shot از انواع آن می‌باشند که در بخش‌های بعدی به آن‌ها می‌پردازیم. در اینجا ما یک مدل از پیش آموزش داده‌شده را در نظر می‌گیریم و با استفاده از prompt-ها سعی می‌کنیم آن را به سمتی هدایت کنیم تا تسک مورد نظرمان را انجام دهد.