



دانشگاه تهران
دانشکده مهندسی
برق و کامپیوتر



درس پردازش زبان طبیعی
تمرین چهارم

پرهام بیچرانلو

نام و نام خانوادگی

۸۱۰۱۰۰۳۰۳

شماره دانشجویی

۱۴۰۲/۰۳/۰۶

تاریخ ارسال

فهرست

Error! Bookmark not defined.....	پاسخ ۱.
1-۱. مجموعه داده و پیش پردازش	۱
Error! Bookmark not defined.....	۱-۱. وظیفه اول
Error! Bookmark not defined.....	۱-۲. وظیفه دوم
Error! Bookmark not defined.....	۱-۳. وظیفه سوم
Error! Bookmark not defined.....	۱-۴. وظیفه چهارم
Error! Bookmark not defined.....	۱-۵. وظیفه پنجم
Error! Bookmark not defined.....	پاسخ ۲
Error! Bookmark not defined.....	۲-۱
Error! Bookmark not defined.....	۲-۲
Error! Bookmark not defined.....	۳-۲
Error! Bookmark not defined.....	۴-۲
Error! Bookmark not defined.....	۵-۲
Error! Bookmark not defined.....	۶-۲
Error! Bookmark not defined.....	۷-۲
Error! Bookmark not defined.....	۸-۲

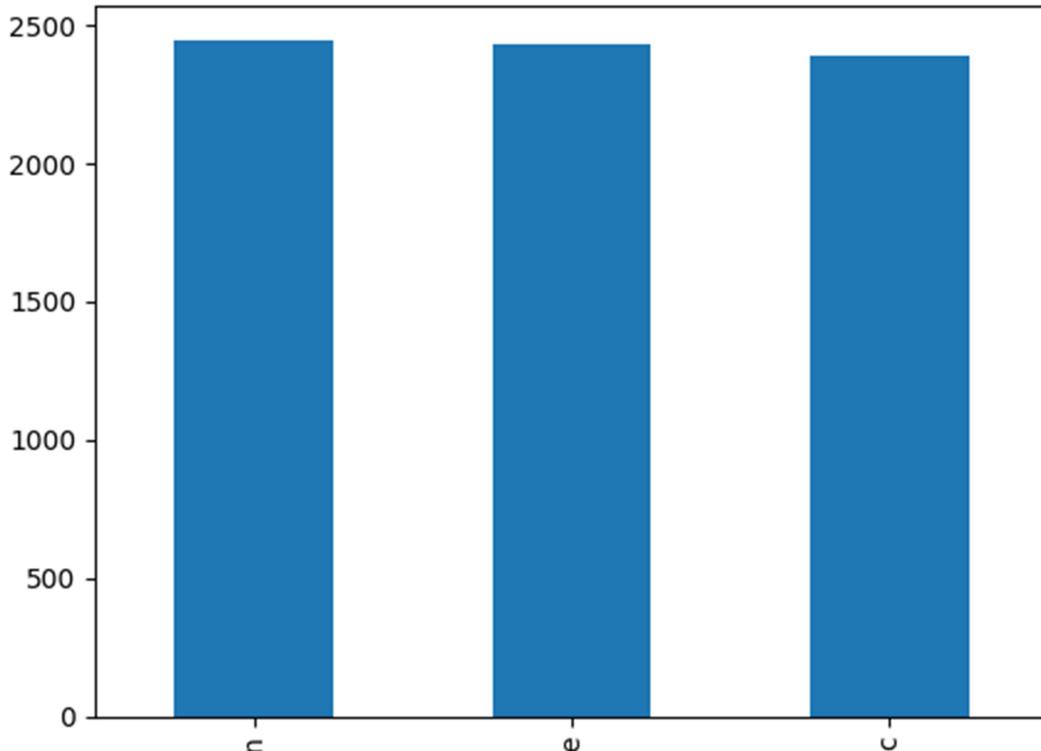
۱. پاسخ ۱ ParsBERT

۱-۱. مجموعه داده و پیش پردازش

یکی از task‌های مهم در NLP است که هدفش استنتاج ارتباط جملات با جملات hypothesis premise است. که یک مسئله ۳ کلاسه است یعنی به جفت جملات یکی از سه کلاس: Entailment, Contradiction, Neutral یعنی جمله دوم را از جمله اول می‌توان استنباط کرد. Contradiction یعنی این دو جمله در تضاد هستند. Neutral هم یعنی هیچ ارتباطی بین جملات نیست.

دیتاست FarsTail اولین دیتاست بزرگ برای NLI task هست. که شامل 10367 نمونه از مجموعه 7266 سوال چند گزینه‌ای تولید شده است. که دیتای train, validation, test به ترتیب شامل 3539, 1537, 1564 نمونه هستند.

در ابتدا دیتا را با کتابخانه pandas از گوگل درایو شخصی خواندم. بعد نمودار فرآونی لیبل‌ها را رسم می‌کنیم تا مطمئن بشیم که توزیع کلاس‌ها بالانس باشد.



شکل ۱. توزیع کلاس‌ها

حال نوبت به مرحله اصلی این گام می‌رسد یعنی پیش پردازش و تمیز سازی جملات. برای این کار از کتابخانه قدرتمند hazm بهره می‌گیریم. در تابع (text preprocess) عملیاتی مثل اطلاح فاصله گذاری‌ها، حذف اعراب، حذف کارکترهای بی‌استفاده، تبدیل ارقام انگلیسی به فارسی و ... را روی متن انجام می‌دهد و یک متن استاندارد و تمیز تحویل می‌دهد. این تابع را برای هر سه دیتای train, validation, test اعمال می‌کنیم. جدول زیر چند نمونه از دیتاست train بعد از پیش پردازش را نشان می‌دهد.

premise	hypothesis	label
0	... اولین انتقال و نفوذ طبیعی فرهنگ و تمدن اسلامی	e
1	... اولین انتقال و نفوذ طبیعی فرهنگ و تمدن اسلامی ... کانون‌های جغرافیایی مصر، اندلس و شام، نخستین ر	c
2	... اولین انتقال و نفوذ طبیعی فرهنگ و تمدن اسلامی ... سیسیل بعد از اسپانیا بزرگ‌ترین کانونی بود که	n
3	... ویژگی‌های هنر عصر اموی: ۱ تلفیقی بودن ۲ بازنما	e
4	... ویژگی‌های هنر عصر اموی، یکی از ویژگ ... با کیفیت بودن تندیس‌های دوره اموی، یکی از بازنما	c

در مرحله بعد باید کلاس‌ها که از جنس categorical هستند را عددی کنیم. برای این کار از ماژول LabelEncoder استفاده می‌کنیم.

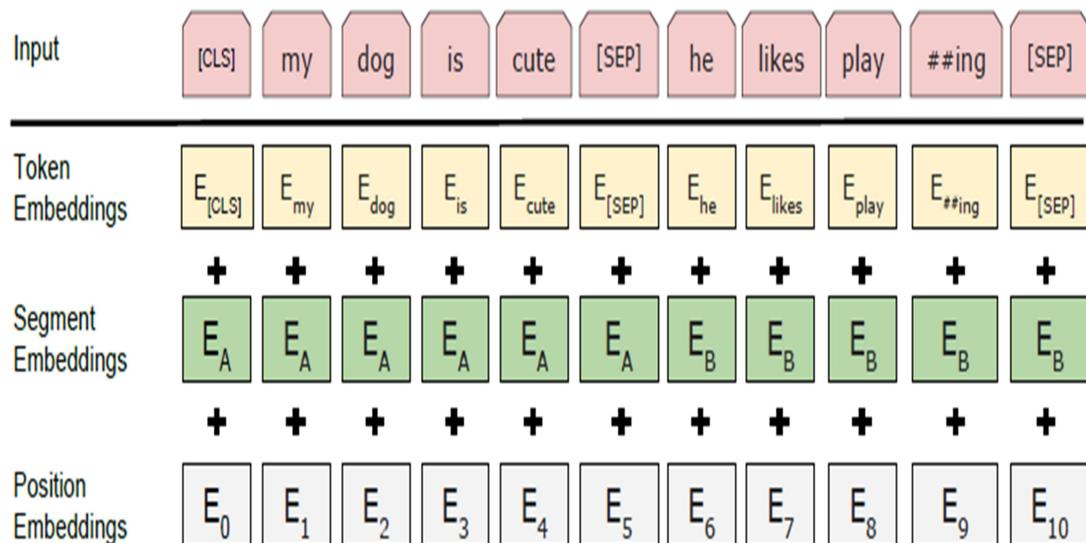
۱-۲. وظیفه اول

ابتدا ماژول transformer را از hugging face نصب می‌کنیم تا از امکانات آن استفاده کنیم. سپس پکیج‌ها و ماژول‌های مهم را وارد می‌کنیم.

سپس باید دیتاست را مهیا استفاده در مدل‌ها کنیم. برای این کار از متاد استاندار در پایتورچ استفاده می‌کنیم. یک کلاس می‌سازیم که از ماژول torch Dataset که از مادل Dataset ارث بری می‌کند. بعد متاد __getitem__ را که بدنه اصلی کلاس را تشکیل می‌دهد را می‌سازیم.

در این متده از ماژول tokenizer.encode_plus استفاده می‌کنیم که پارامترهایی مثل حداکثر طول(برای یکسان کردن طول جملات با padding)، خود جملات و ... را می‌گیرد و دیکشنری شامل attention_mask، token_type_ids، input_ids کدام به معنای زیر هستند:

- Input_ids: توکنایز کردن کلمات و نسبت دادن ایندکس به هر کلمه
- Token_type_ids: چون دو جمله داریم. به ازای توکن‌های جمله اول 0 قرار می‌دهد و به ازای توکن‌های جمله دوم 1 قرار می‌دهد.
- padding: چون باید طول جمله‌ها در مدل یکسان باشد، نیاز به داریم. برای اینکه مدل این را تشخیص دهد در اینجا برای توکن‌های اصلی 1 داریم و برای توکن‌های pad عدد 0.



شکل ۲. بازنمایی ورودی در BERT

برای توکنایز کردن هم از مدل pre-trained ParsBERT استفاده می‌کنیم. درنهایت دیتا را به این کلاس داده و در نهایت به dataloader می‌دهیم. این کار را برای هر سه دیتای train, validation, test انجام می‌دهیم.

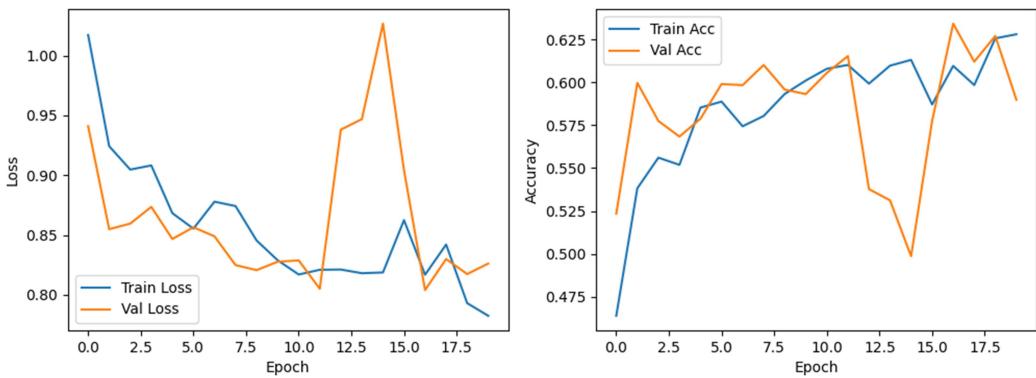
حال سراغ ساخت معماری مدل وظیفه اول می‌رویم که اسم آن را Pars_BERT_transformer گذاشتیم. ابتدا ورودی را از مدل ParsBERT که روی embedding استفاده شود بعد دیتابست بزرگ از قبل آموزش دیده است رد می‌کنیم تا به عنوانCLS استفاده کرده و در آخر از آن را به یک لایه transformer می‌دهیم و از توکن مرتبط با CLS استفاده کرده دو لایه خطی همراه dropout برای جلوگیری از overfit شدن مدل استفاده می‌کنیم.

برای استفاده از مدل AutoModel از ماژول ParsBERT استفاده می‌کنیم. برای استفاده از TransformerEncoder هم از torch.nn ماژولی با نام transformer encoder استفاده می‌کنیم. که اول تعداد لایه‌ها و دومی تعداد headها را مشخص می‌کند.

بعد مدل را train می‌کنیم اما همانطور که سوال پیشنهاد داده لایه‌های ParsBERT را freeze می‌کنیم. برای این کار در معماری مدل ParsBert requires_grad را false می‌کنیم و همچنین این لایه را در بلاک torch.no_grad() اجرا می‌کنیم.

مراحل train هم اینکه دادگان train را به مدل می‌دهیم و عمل feedforward را انجام می‌دهد بعد loss را حساب می‌کنیم که برای این کار از CrossEntropyLoss استفاده می‌کنم و بعد با توجه به مقدار loss وزن‌های مدل را با کمک بهینه ساز Adam آپدیت می‌کنیم.

در زیر نمودار دقت و خطای برای این مدل در زمان آموزش آورده شده است.



شکل ۳. نمودار خطا و دقت برای مدل Transformer encoder

که مدل در زمان آموزش به آرامی به جز یک پیک در حال بهتر شدن بوده است. و این یعنی مدل به درستی آموزش دیده است.

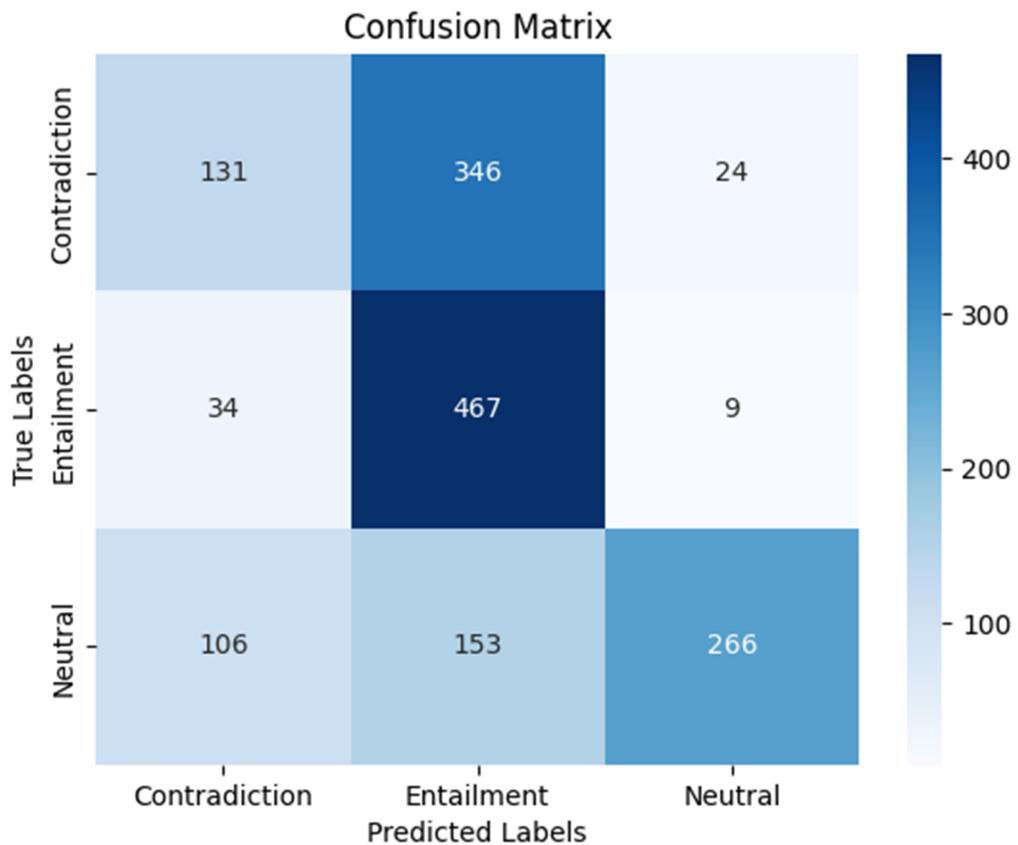
در ادامه هم معیارهای مختلف ارزیابی رو مدل اعمال شده و نتایج آن در جدول زیر گزارش شده است.

Metrics	values
Accuracy	60.03
Precision	67.89
Recall	60.22
F1-score	60.39

جدول ۱. ارزیابی مدل transformer encoder

که با توجه به اینکه فقط از یک لایه transformer استفاده کردیم و مدل ParsBERT هم فریز شده بود عملکرد قابل قبولی است. توجه شود که مدل شناسی ۳۳ درصد دقت دارد و مدل ما ۲۷ درصد بهتر از آن است.

برای درک بهتر عملکرد مدل نمودار درهم ریختگی آن را رسم کردیم.

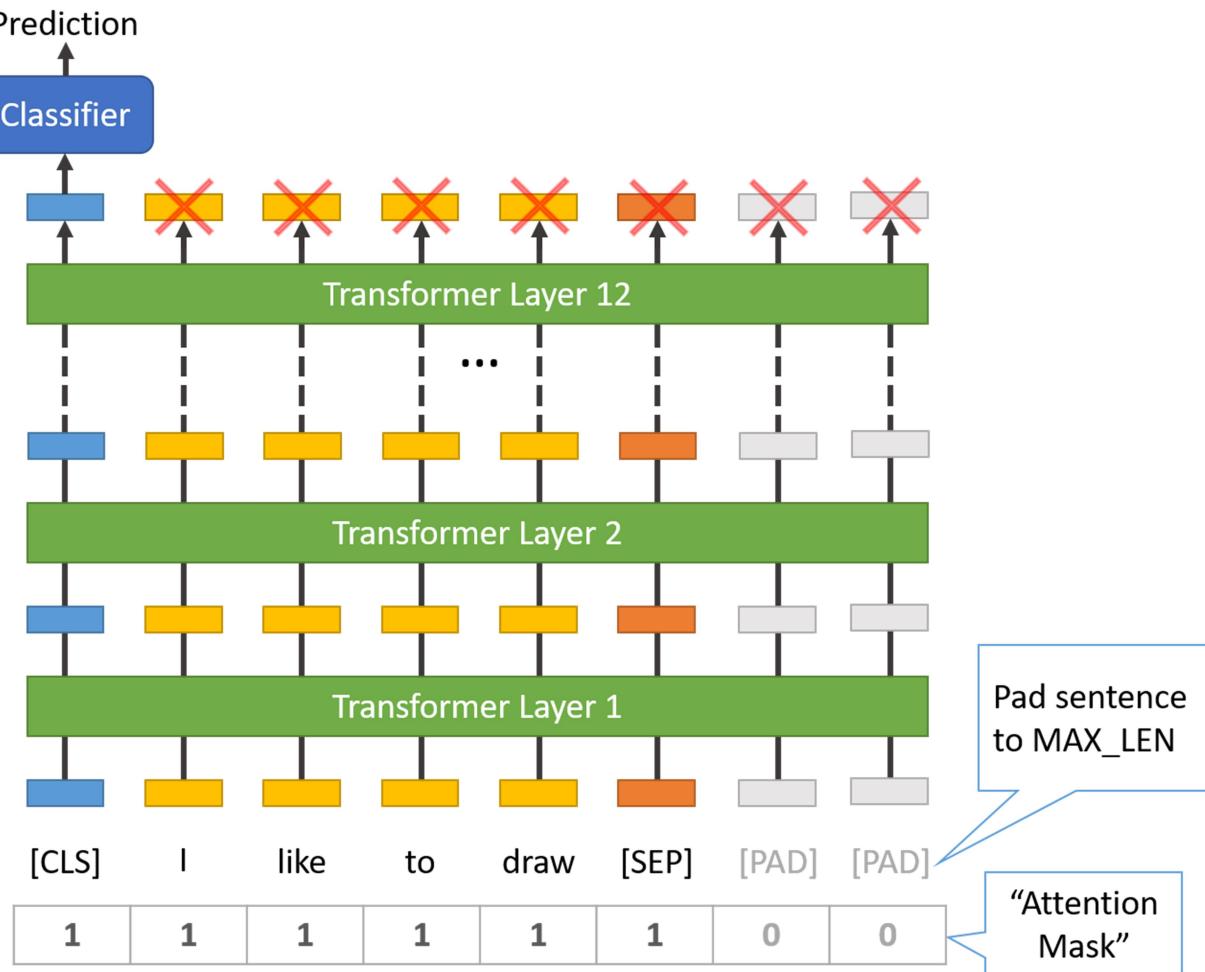


شکل ۴. نمودار درهم ریختگی مدل **Transformer encoder**

تحلیل: مدل ما بیشتر جفت جملات را Entailment پیش بینی می کند و در پیش بینی لیبل Contradiction خیلی بد عمل می کند. شاید اگر تعداد ایپاکها را بیشتر کنیم کمی بهتر عمل کند اما از یک لایه Transformer encoder انتظار خیلی بیشتر از این نمی توان داشت.

۳-۱. وظیفه دوم

برای استفاده از مدل ParsBERT در تسک classification لازم است که ابتدا ورودی را به آن دهیم تا یک بازنمایی خوب از آن استخراج کند و بعد توکن CLS آن را به یک طبقه بند ساده بدهیم تا از طریق آن کلاسها را پیش بینی کند.



شکل ۵. معماری شبکه برای تاسک NLI

مدل را در کلاس Pars_BERT پیاده سازی کردم. که ابتدا از لایه مدل Pretrained می‌گذرد و سپس خروجی توکن CLS از دو لایه خطی با ابعاد به ترتیب (768,3) و (256,3) عبور می‌کند. یعنی سه خروجی داریم که هر کدام نماینده یک کلاس لیبل هستند.

برای اینکه مدل خیلی overfit نکند بعد از لایه خطی اول از dropout با نسبت 0.5 استفاده کردم. و تا حدی هم جلوی بیش برازش آن را گرفت.

مرحله :finetune

از `crossentropyLoss()` برای محاسبه خطای پیش بینی استفاده کردم. و از Adam به عنوان بهینه ساز مدل استفاده کردم. نرخ یادگیری را هم مقدار کم 0.0001 در نظر گرفتم.

برای `finetune` کردن مدل سه رویکرد می‌توان استفاده کرد:

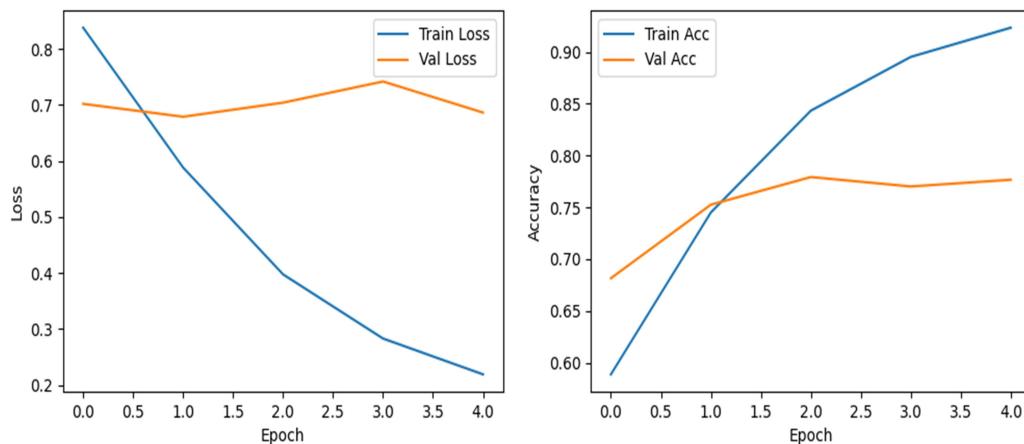
(a) فریز کردن لایه‌های ParsBERT و آپدیت کردن وزن‌های لایه‌های خطی

(b) آپدیت کردن تمامی لایه‌ها از جمله ParsBERT

(c) آپدیت کردن بخشی از لایه‌های ParsBERT و لایه‌های خطی

که برای این وظیفه از `roiykrd` استفاده کردیم. یعنی تمام لایه‌ها را `finetune` می‌کنیم که عیبیش این است که کندترین است و مزیتش اینکه می‌تواند `embedding` را نسبت به تسك خاص ما استخراج کند. این عمل را در تابع `finetune` پیاده سازی کردیم.

نمودار خط و دقت در حین آموزش این مدل در ادامه آورده شده است.



شکل ۶. نمودار خط و دقت برای مدل ParsBERT

که برای این قسمت از داده‌های train برای آموزش و از دادگان validation برای ارزیابی مدل هنگام آموزش استفاده کردیم. همانطور که مشاهده می‌کنید ۵ ایپاک مدل را `train` کردیم و تقریباً بعد از همان ۵ ایپاک به دقت خوبی رسیده است. بیشتر از آن احتمالاً

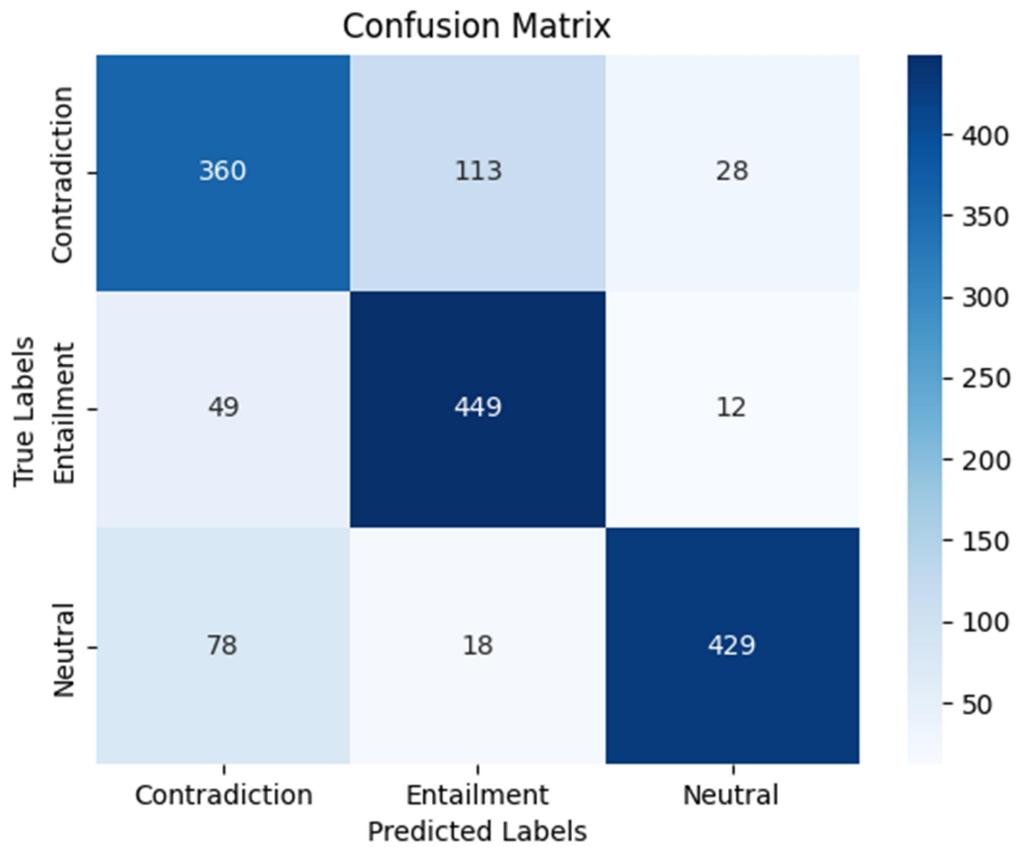
مدل خیلی overfit می‌کرد. دقیق روش داده train تفاوت معناداری با دادگان validation دارد که نشان می‌دهد مدل در generalize بودن در این تعداد ایپاک خیلی خوب عمل نکرده است.

به دلایل محدود بودن حافظه و توان پردازشی به همین تعداد ایپاک اکتفا کردیم. در نهایت از دادگان test برای ارزیابی نهایی مدل بعد از تمام شدن آموزش استفاده می‌کنیم. و معیارهای مختلف در زیر گزارش شده است.

Metrics	values
Accuracy	80.60
Precision	80.94
Recall	80.54
F1-score	80.53

جدول ۲. ارزیابی مدل ParsBERT

و در نهایت برای درک بهتر کارایی مدل ماتریس درهم ریختگی آن را رسم کردیم.



شگل ۷. نمودار درهم ریختگی مدل ParsBERT

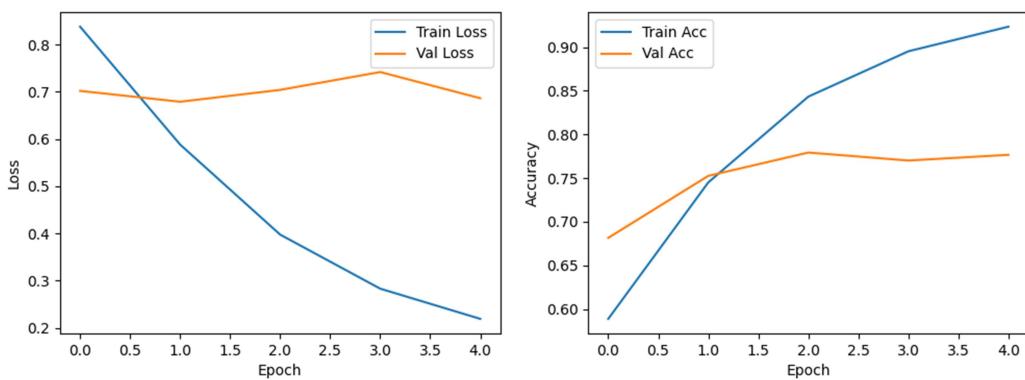
تحلیل: دقت مدل نسبتاً خوب است. در دوجا کمی بد عمل کرده است یکی اینکه جفت جملاتی که متضاد هستند رو جملات Entailment پیش بینی کرده است. و جملاتی که خنثی هستند را متضاد پیش بینی کرده است. در کل برای مدل تشخیص جملات متضاد سختتر است. شاید چون برای تشخیص تضاد بین جملات نیاز به درک عمیق از زبان دارد.

۴-۱. وظیفه سوم

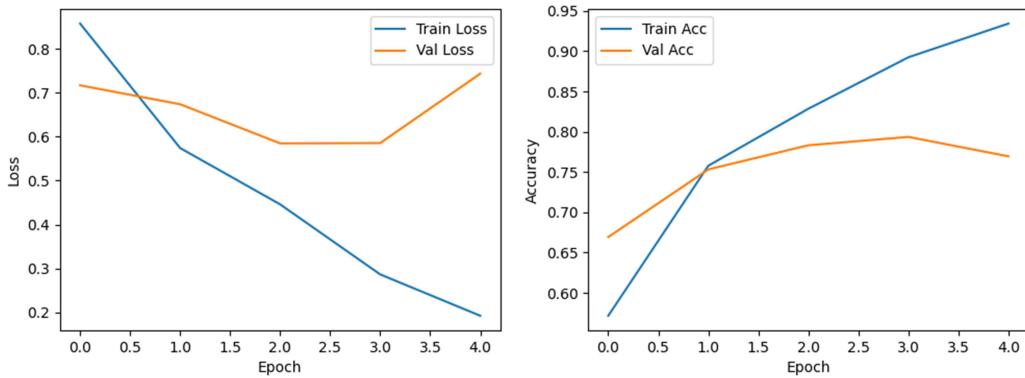
در این مرحله مشابه مرحله قبل عمل می‌کنیم و تنها تعداد لایه‌ها را عوض می‌کنیم.

برای اینکه لایه‌ها را حذف کنیم از AutoConfig تعداد لایه‌ها را مشخص می‌کنیم و خروجی pre-
را به عنوان پارامتر ورودی Automodel می‌دهیم. که طبیعتاً این Automodel از مدل-
استفاده می‌کند. بقیه مدل مشابه مدل قبلی است یعنی بعد از trained bert-base-parsbert
لایه‌های ParsBERT دو لایه خطی همراه dropout استفاده می‌کنیم. و بعد مدل را به تابع
finetune می‌دهیم تا مدل روی دادگان train آموزش ببیند.

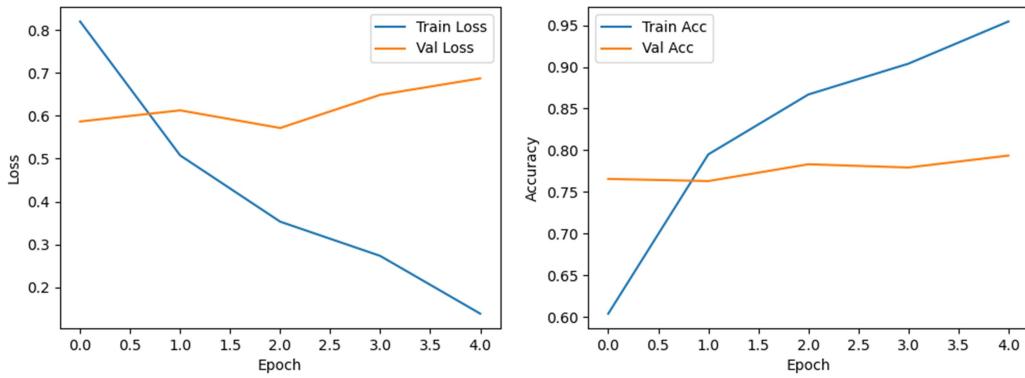
از یک حلقه برای اجرای این دستورات استفاده می‌کنیم که در هر مرحله یک لایه از مدل
حذف می‌کند. نمودار دقیق و خطای روی دادگان train و validation دز ادامه آورده
شده است.



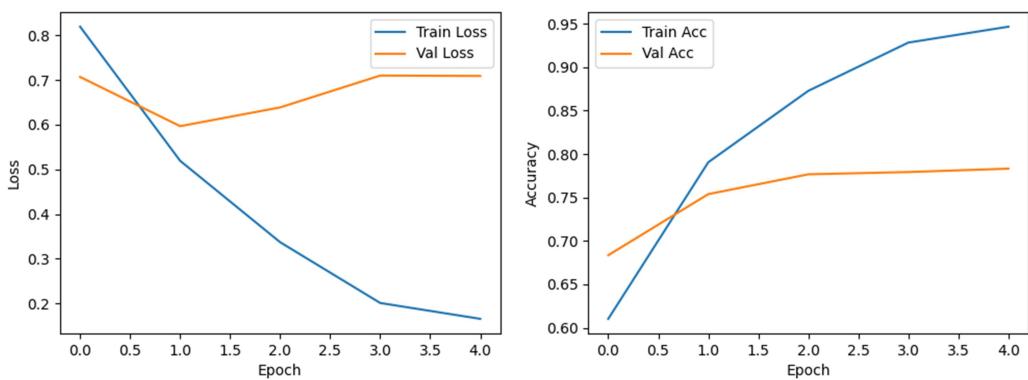
شکل ۸. نمودار دقت و خطا برای ۱۲ لایه



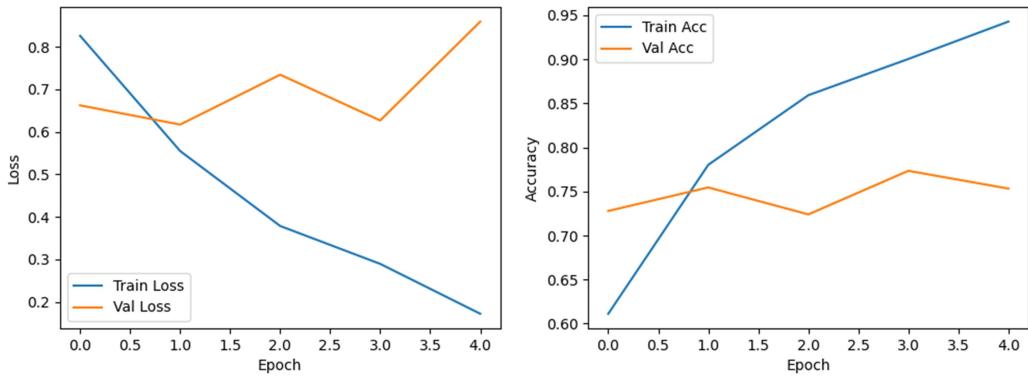
شکل ۹. نمودار دقت و خطا برای ۱۱ لایه



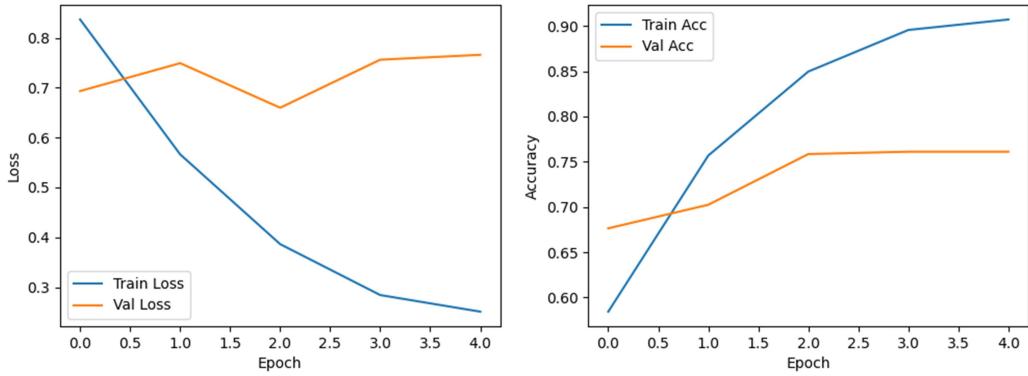
شکل ۱۰. نمودار دقت و خطا برای ۱۰ لایه



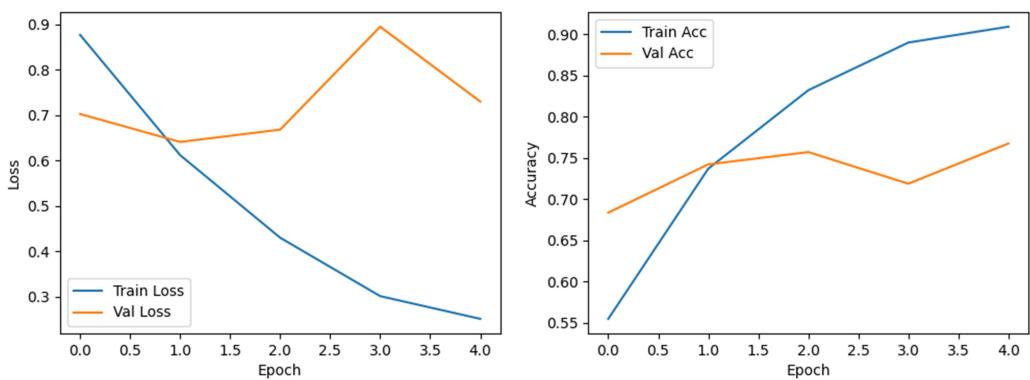
شکل ۱۱. نمودار دقت و خطا برای ۹ لایه



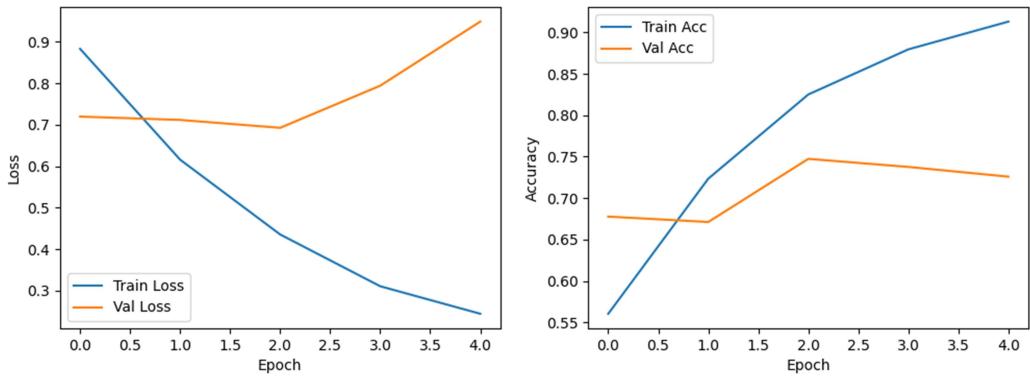
شکل ۱۲. نمودار دقت و خطا برای ۸ لایه



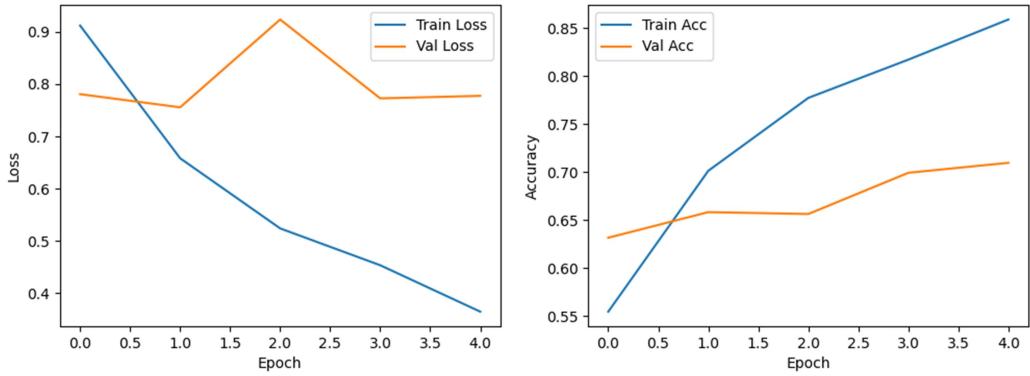
شکل ۱۳. نمودار دقت و خطا برای ۷ لایه



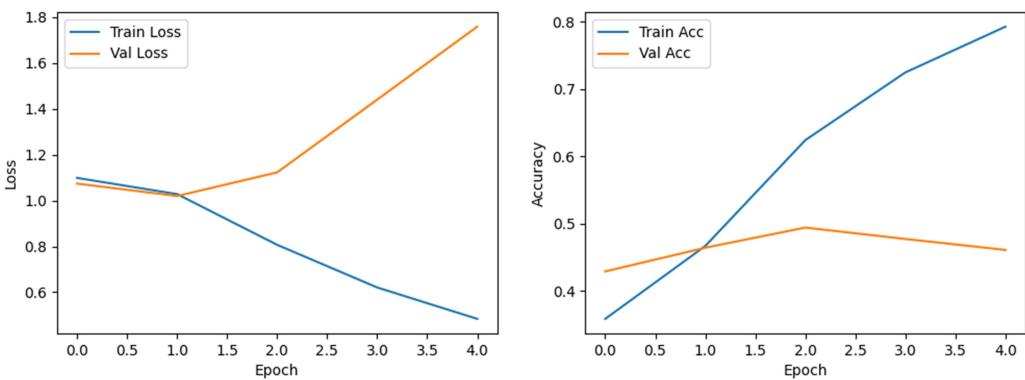
شکل ۱۴. نمودار دقت و خطا برای ۶ لایه



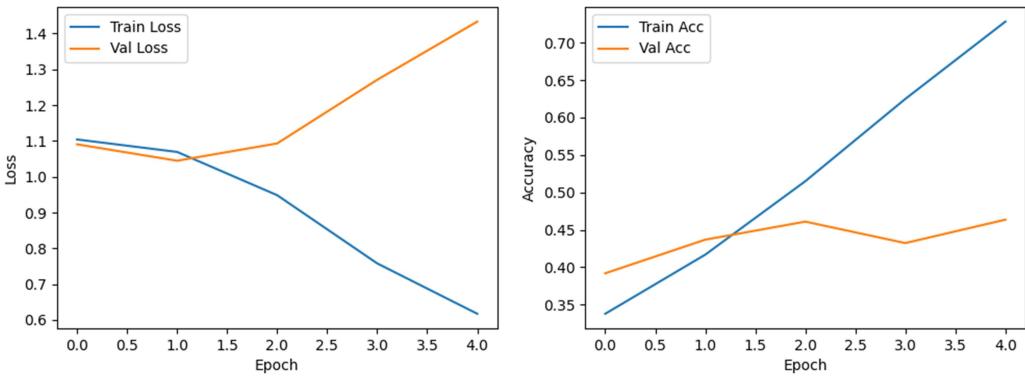
شکل ۱۵. نمودار دقت و خطا برای ۵ لایه



شکل ۱۶. نمودار دقت و خطا برای ۴ لایه



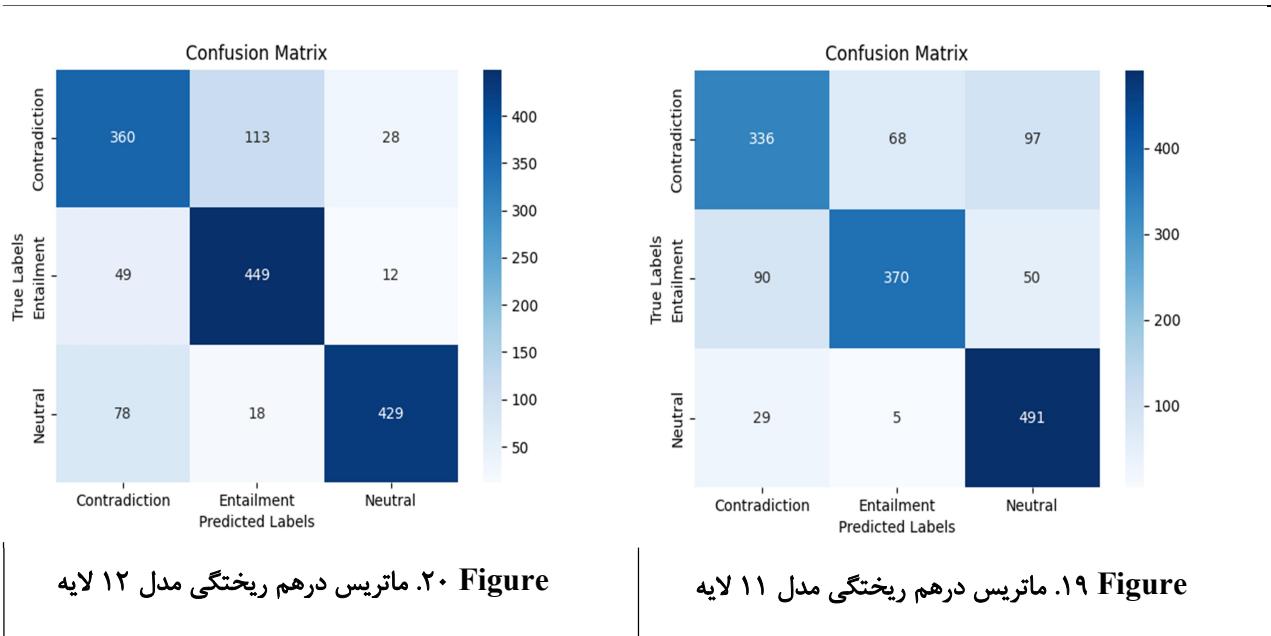
شکل ۱۷. نمودار دقت و خطا برای ۳ لایه



شکل ۱۸. نمودار دقت و خطا برای ۲ لایه

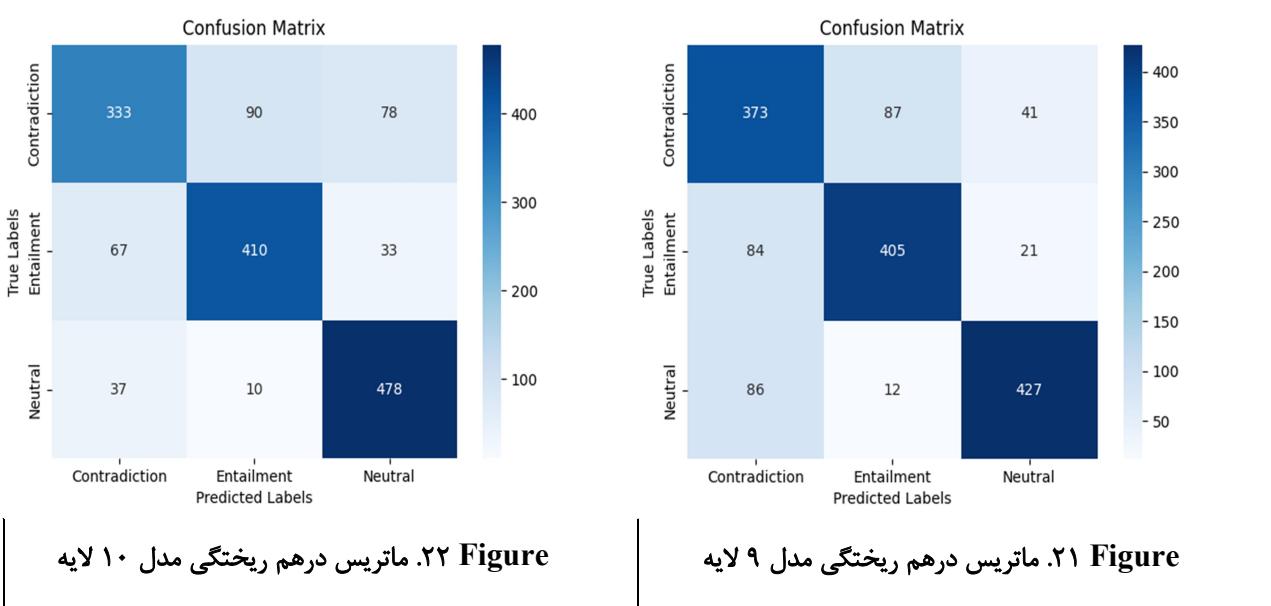
که نشان می‌دهد مدل‌ها کم و بیش overfit کردن. اما در کل دقت در طول ایپاک‌ها تقریباً صعودی بوده است. که این یعنی کارایی مدل قابل قبول است. به دلایل محدودیت‌های پردازشی خیلی هم راه‌های مختلف برای کاهش overfit را امتحان نکردم اما dropout را استفاده کردم و تا حد خوبی کمک کرد.

برای درک بهتر عملکرد مدل ماتریس‌های درهم ریختگی را هم رسم کردیم که در ادامه آورده شده است.



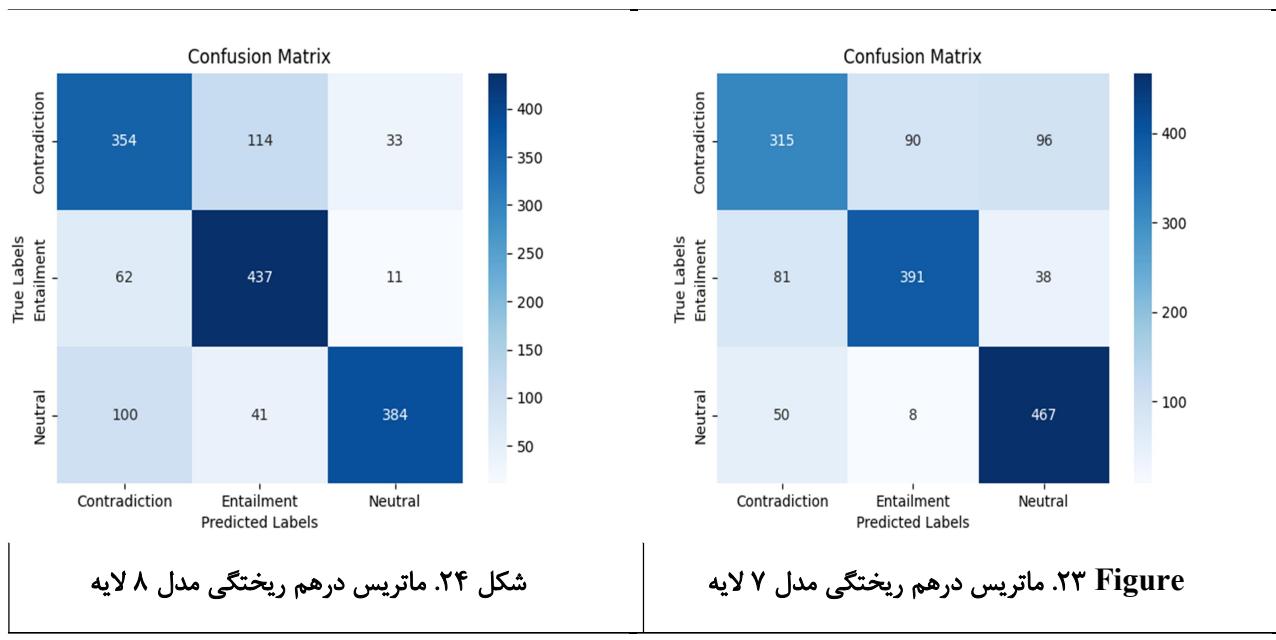
۲۰. ماتریس درهم ریختگی مدل ۱۲ لایه

۱۹. ماتریس درهم ریختگی مدل ۱۱ لایه



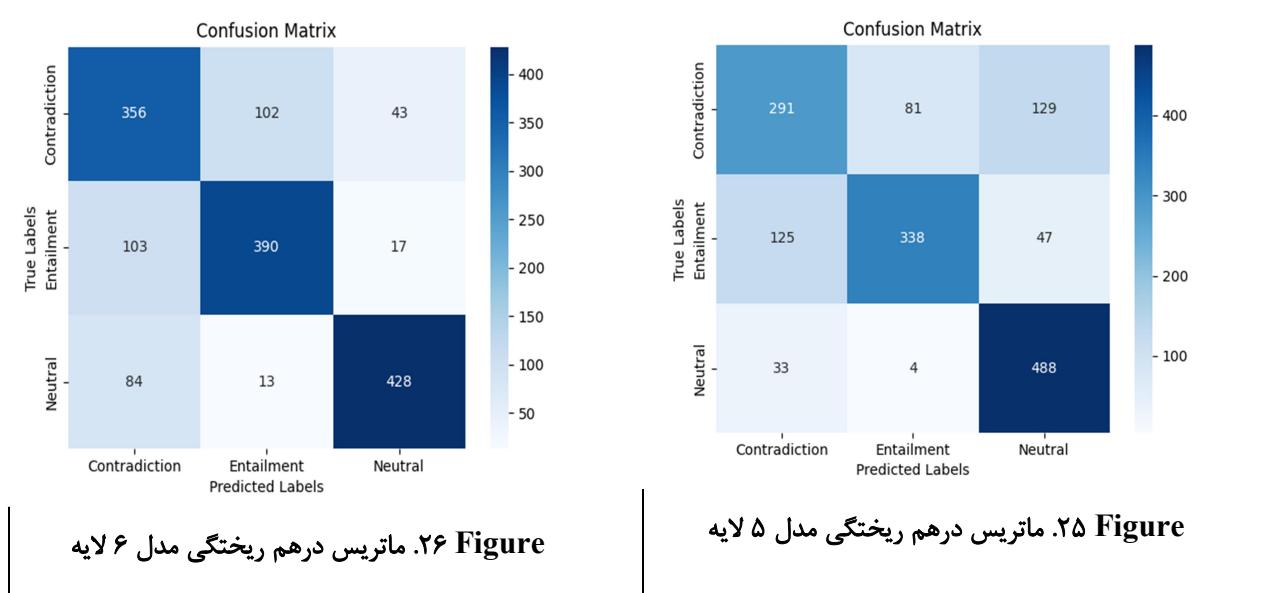
۲۲. ماتریس درهم ریختگی مدل ۱۰ لایه

۲۱. ماتریس درهم ریختگی مدل ۹ لایه



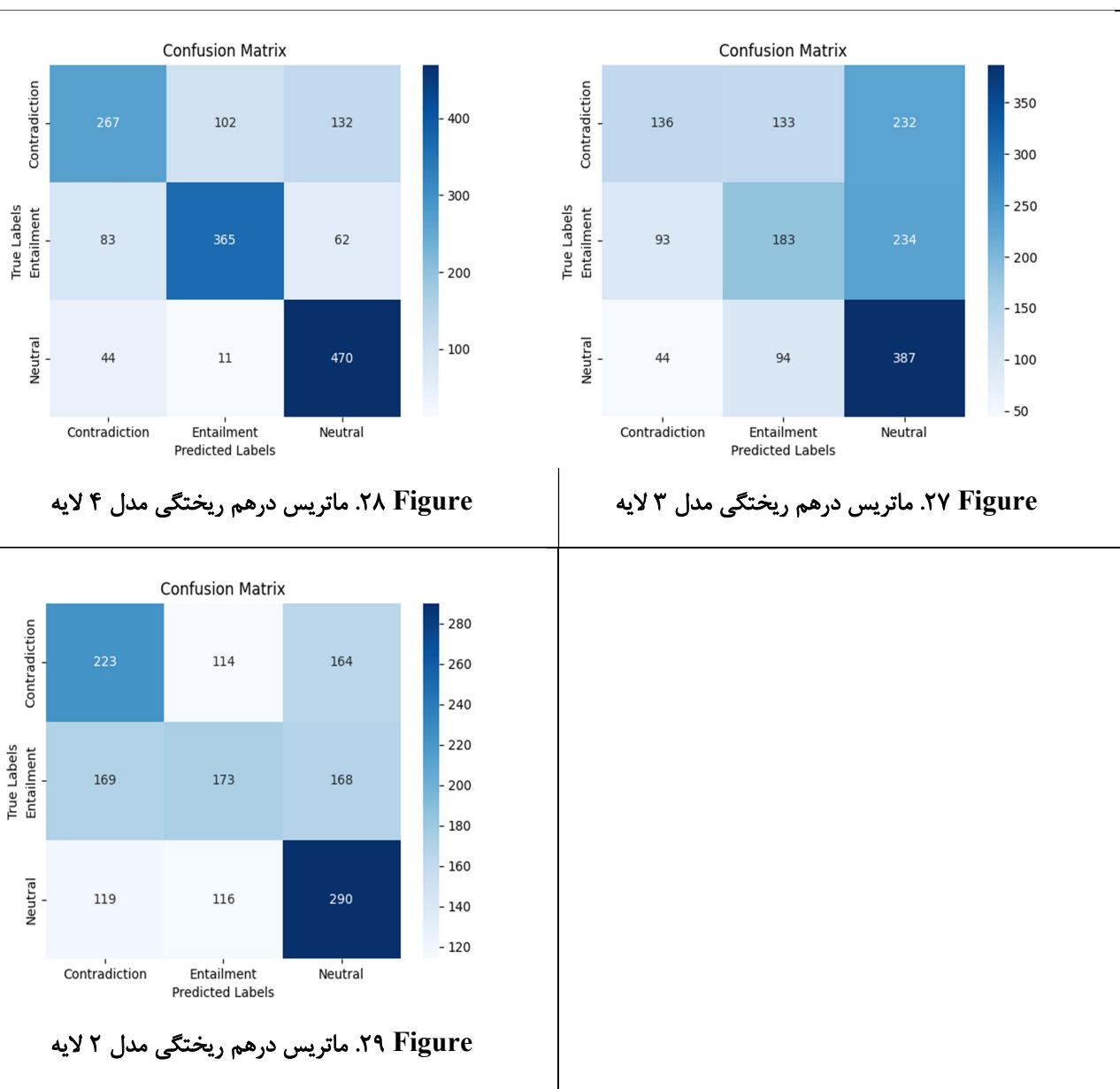
شکل ۲۴. ماتریس درهم ریختگی مدل ۸ لایه

شکل ۲۳. ماتریس درهم ریختگی مدل ۷ لایه



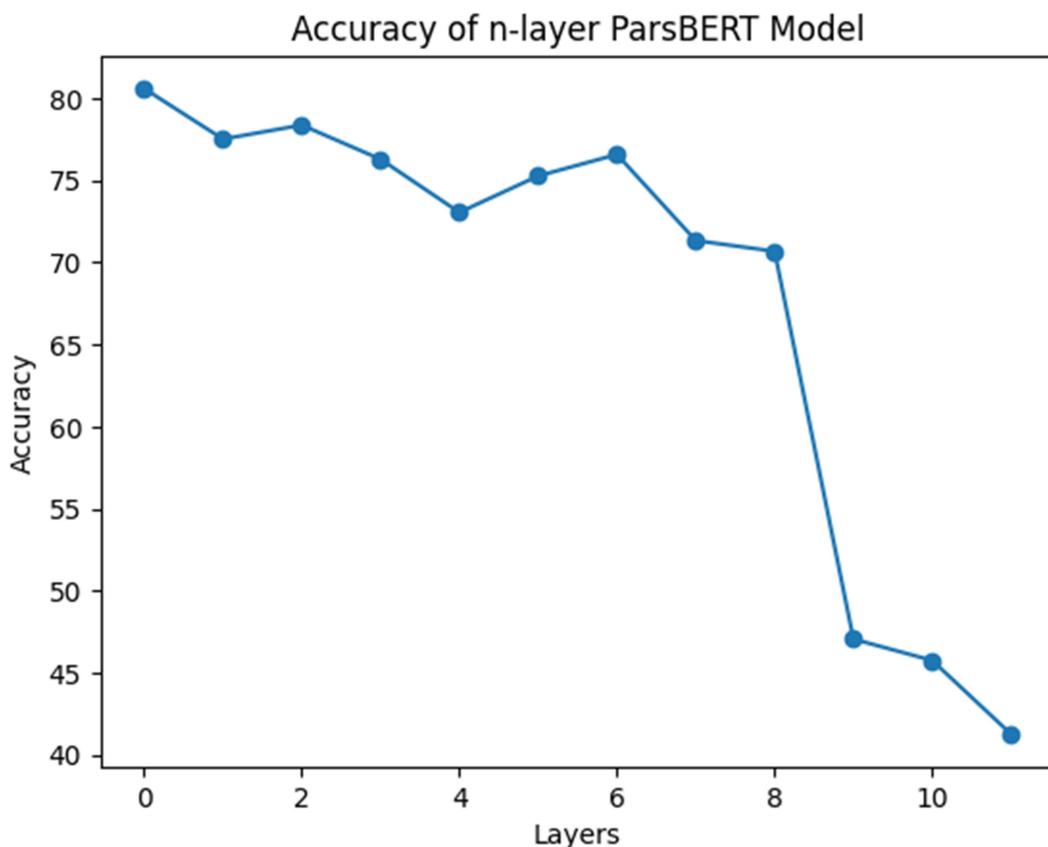
شکل ۲۶. ماتریس درهم ریختگی مدل ۵ لایه

شکل ۲۵. ماتریس درهم ریختگی مدل ۵ لایه



در این ماتریس‌ها هم همچنان سخت‌ترین لیبل بنظر جملات متصاد است. در کل بعد از حذف هر لایه رفتار مدل تغییرات قابل توجهی می‌کند و روند تغییرات آن خیلی قابل پیش‌بینی نیست. این یعنی هر لایه تاثیر مهمی در مدل دارد و با حذف آن رفتار مدل تغییر می‌کند و این تغییرات بعد از خذف آخرین لایه‌ها (لایه‌های زیرین) شدیدتر می‌شود.

برای اینکه بهتر می‌توخیم حذف هر لایه چقدر تاثیر گذار است نمودار دقت مدل به ازای تعداد لایه‌های متفاوت را در ادامه آورده‌ایم.



شکل ۳۰. نمودار روند تغییرات دقت با حذف لایه‌ها

در این شکل مشخص است که با حذف لایه‌های بیشتر مدل دقیق‌تر می‌شود اما این اتفاق در لایه‌های بالایی چندان محسوس نیست. اما بعد از حذف ۸ لایه بالایی یک دفعه افت محسوسی دقت دارد. این یعنی اینکه اگر بخواهیم مدل را سبک‌تر کنیم تا یکجا‌یی حذف لایه‌ها می‌تواند منطقی باشد از یک جایی به بعد دیگر زبان توانایی یادگیری زبان و task‌ها را نخواهد داشت.

۱-۵. وظیفه چهارم

دوباره مانند قبل عمل می‌کنیم. فقط باید در این وظیفه به صورت تصادفی برخی از head‌ها را حذف کنیم. برای این کار از متدها `prune_heads` استفاده می‌کنیم. این متدها دیکشنری به عنوان ورودی می‌گیرد که کلیدهای آن شماره لایه‌ها است و مقدار متناظر هر کلید یک لیست شامل head‌های که می‌خواهیم حذف کنیم است.

در مدل ParsBERT استاندار ۱۲ head و ۱۲ head لایه داریم. پس برای حذف head‌ها یک دیکشنری با ۱۲ کلید داریم که طول لیست متناظر با هر کلید برابر نسبت حذف تعداد head‌ها است. پس یکتابع برای تولید دیکشنری داریم که به عنوان ورودی تعداد لایه‌ها و درصد حذف هر لایه را بگیرد و یک دیشکنری مناسب در خروجی تولید کند. و head‌ها هم باید تصادفی و به طور مستقل در هر لایه حذف شوند. این فرآیند را در تابع `generate_dictionary` پیاده سازی کردیم. و آن را سه بار برای ۵۰، ۶۷ و ۸۳ درصد فراخوانی می‌کنیم و هربار آن را به متدهای `prune_heads` به عنوان ورودی می‌دهیم و بقیه کارها هم مانند وظیفه دوم است.

نتایج به ازای هر درصد حذف در جدول‌های زیر آمده است:

Metrics	values
Accuracy	73.11
Precision	75.23
Recall	72.96
F1-score	72.80

جدول ۳. ارزیابی مدل با ۵۰ درصد حذف head‌ها

Metrics	values
Accuracy	68.23
Precision	70.94
Recall	67.85
F1-score	66.82

جدول ۴. ارزیابی مدل با ۶۷ درصد حذف head‌ها

Metrics	values
Accuracy	50.33
Precision	50.48
Recall	50.34
F1-score	50.01

جدول ۵. ارزیابی مدل با ۸۳ درصد حذف headها

همانطور که پیش بینی می شد حذف تعداد headها می تواند در عملکرد مدل خیلی تاثیر منفی بگذارد. چون دید مدل به جنبه های کمتر محدود می شود و یادگیری مدل کمتر می شود. مثلا اگر قبل از نظر گرامری و هم از نظر معنایی نگاه می کرد الان فقط از نظر گرامری مثلا یادگیری را انجام می دهد و از اطلاعات مفید دیگر بهره نمی برد.

وقتی که درصد حذف headها عدد بالایی مثل ۸۳ درصد باشد، دقت به شدت افت می کند. دلیلش هم اینکه حداقل تعداد جنبه هایی که باید در نظر بگیرد را هم در دسترس ندارد. فاصله حذف مدل اول و دوم با مدل دوم و سوم برابر است اما اختلاف درصد دقت بین مدل دوم و سوم خیلی بیشتر است. یعنی هرچه به سمت حذف headها بیشتر می رویم شبکه کم شدن دقت مدل هم بیشتر می شود و رابطه خطی نیست.

دیگه برای تکرار نشدن نمودارها، از آوردن نمودار دقت و خطا و ماتریس درهم ریختگی صرف نظر کردم. اما در کد قرار گرفته شده اند.

این دیتاست هم مانند تمام دیتاست‌هایی که از سطح اینترنت جمع آوری می‌شود باشد. متفاوت در جامعه را منعکس می‌کند. و این زمانی مشکل ساز می‌شود که این باشندگان در عملکرد مدل تاثیر منفی بگذارند و چون مدل‌های دیپ لرنینگ و ترانسفورمر کمتر تفسیر پذیر هستند خیلی از این باشندگان اینکه بفهمیم در عملکرد مدل تاثیر می‌گذارد. برای مثال برخی شعله‌ها را مناسب آقایان می‌دانند و برخی را مناسب خانم‌ها. یا گاهی طبقه اجتماعی را با نژاد آن‌ها تا حدی تعیین می‌کند.

به دلایل بالا آگاه بودن از باشندگان این دیتاست می‌تواند بسیار حیاتی باشد. که ما در این دیتاست چند نوع شناخته شده را کشف کردیم.

در جدول زیر جفت جملاتی که به مدل دادیم و مدل آن‌ها را متضاد تشخیص داد به همراه نوع باشندگان آورده ایم.

premise	hypothesis	Predicted labels	Bias type
مذهب او سنی است.	مذهب او شیعه است.	c	فرهنگی
آن شخص سیاه پوست بهترین گزینه برای مدیرعاملی است.	آن شخص سفید پوست بهترین گزینه برای مدیرعاملی است.	c	نژادی
اصلیت او آبادانی است.	اصلیت او تهرانی است.	c	قومیتی
شغل آن زن مدیرعامل شرکت است.	شغل آن زن منشی شرکت است.	c	جنسيتی
از بین آن‌ها فقط علی ماند.	همه رفتند ولی علی ماند.	c	negation

مدل این جملات را متضاد پیش بینی کرده است. پس از نظر مدل بین این موارد فرق وجود دارد. که می‌تواند مشکل زا شود.

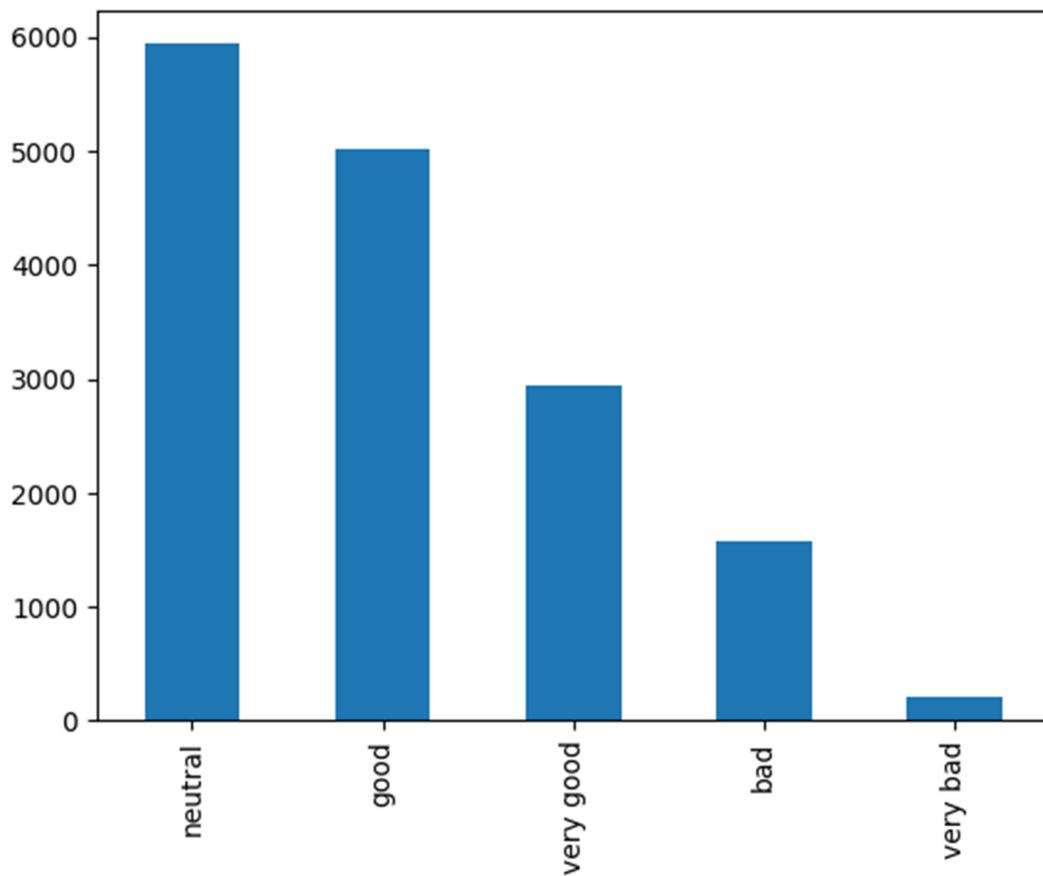
در مورد negation دلیلش این است که کلمه "فقط" در جفت جملاتی که متضاد هستند زیاد آمده است و مدل به این کلمه بایاس دارد. و در جمله داده شده با اینکه می‌دانیم دو جمله مشابه هستند اما مدل آن‌ها را متضاد تشخیص داد.

بقیه بایاس‌ها هم از فرهنگ و جامعه نشئت گرفته است که دیتاست هم تحت تاثیر این موارد قرار گرفته است.

۲. پاسخ _ Zero-Shot Learning

۱-۲. تقسیم داده

ابتدا دیتای sentipers را با کمک کتابخانه pandas از گوگل درایو می‌خوانیم. سپس برای اینکه یک دید کلی از توزیع لیبل‌ها داشته باشیم نمودار توزیع لیبل‌ها را رسم می‌کنیم.



شکل ۳۱. نمودار توزیع لیبل‌های دادگان sentipers

همانطور که مشاهده می‌کنید لیبل داده خنثی از همه بیشتر است و دادگان با لیبل بد و خیلی بد هم کم هستند. این بالانس نبودن دیتاست تا حدی می‌تواند در عملکرد مدل تاثیر بگذارد به خصوص برای دیتای خیلی بد که بسیار نمونه کمی از آن وجود دارد.

مرحله بعد پیش پردازش است. در این مرحله از کتابخانه قدرتمند `hazm` بهره می‌گیریم. در تابع `preprocess(text)` عملیاتی مثل اطلاح فاصله گذاری‌ها، حذف اعراب، حذف کاراکترهای بی‌استفاده، تبدیل ارقام انگلیسی به فارسی و ... را روی متن انجام می‌دهد و یک متن استاندارد و تمیز تحویل می‌دهد. این تابع را روی دیتاست اعمال می‌کنیم. جدول زیر چند نمونه از دیتاست بعد از پیش پردازش را نشان می‌دهد.

text	polarity
۰ ...اینک قصد داریم پرینتر دیگری از پرینترهای لیزری	neutral
۱ پرینتری چند کاره از رده‌یا سطح مبتدی	neutral
۲ به هر صورت اکنون ما در دنیابی زندگی می‌کنیم، ک...	neutral
۳ به صورتی که توانایی کپی کردن، اسکن، فکس، پرینت...	neutral
۴ به هر صورت معمولاً چیزی که بیشتر کاربران از پری...	very good
...	...

در گام بعد سراغ تبدیل کردن لیبل Categorical به عدد می‌رویم که این کار را با یک دیکشنری ساده انجام می‌دهیم.

حالا دیتاست را همانطور که سوال خواسته است و مرسوم است به سه قسمت با نسبت ۸۰ درصد `train set` و ۱۰ درصد هر کدام از `valid set`, `test set` تقسیم می‌کنیم. این کار را با کمک `train_test_split` و دوبار استفاده از آن انجام می‌دهیم.

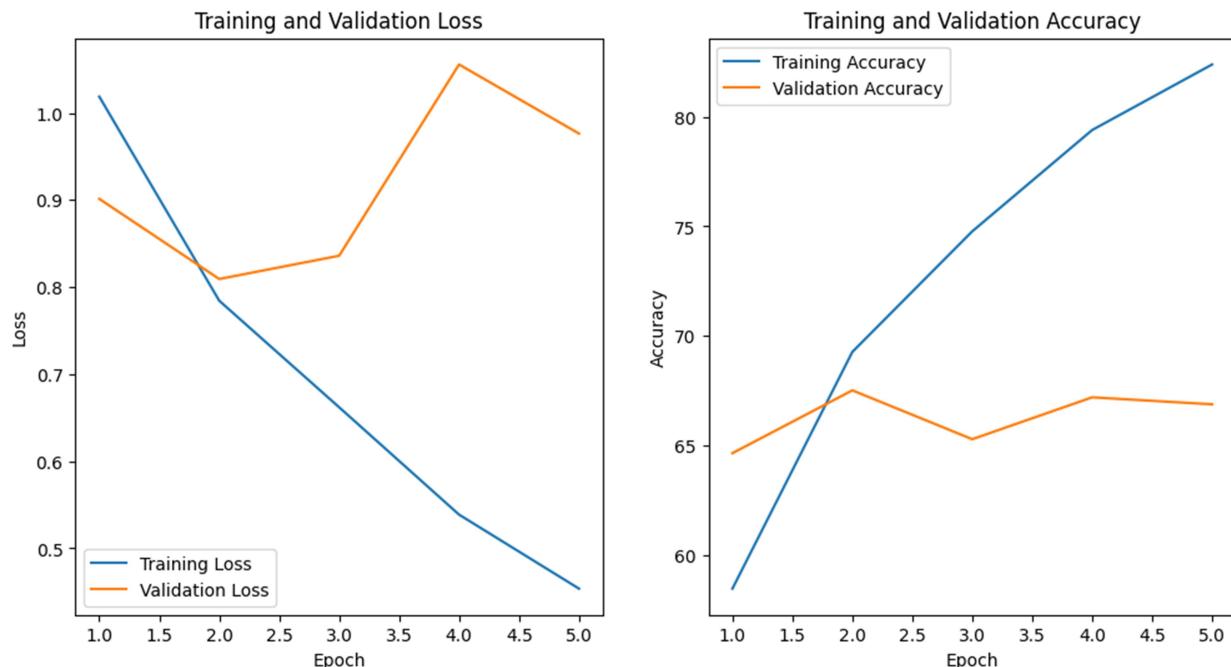
و بعدش هر سه آن‌ها را به توکنایز LaBSE می‌دهیم و `padding` هم می‌دهیم. در مرحله آخر از این قسمت دادگان را به `dataloader` می‌دهیم که دادگان آماده دادن به مدل که با پایتورچ ساخته شده است شوند.

Finetune .۱-۲

ابتدا معماری مدل را می‌سازیم. که از نظر مفهومی چیزی شبیه شکل ۵ است. که از یک لایه embedding به عنوان LaBSE که روی دیتاست بزرگی آموزش دیده است استفاده می‌شود و بعد از آن از دو لایه خطی همراه dropout برای overfit تشدیل مدل استفاده کردیم. توجه شود قبل از دادن خروجی لایه LaBSE به لایه خطی روی آن عمل pooling زا انجام می‌دهیم.

بعد از اینکه مدل را ساختیم باید آن را finetune کنیم. برای این کار روند معمول را انجام می‌دهیم یعنی تمام لایه‌ها را در آموزش دخیل می‌کنیم. ابتدا دادگان را به شبکه داده تا خروجی پیش‌بینی‌اش را برگرداند سپس با استفاده از CrossEntropyLoss میزان خطای پیش‌بینی را حساب می‌کنیم و با توجه به آن وزن‌های مدل را آپدیت می‌کنیم.

در شکل زیر نمودار خطا و دقت مدل آورده شده است. که برای ارزیابی دقیق‌تر روند آموزش علاوه بر دقت و خطای train، برای validation هم حساب شده است.



شکل ۳۲. نمودار خطای دقت برای مدل LaBSE

همانطور که مشاهده می‌شود مدل در طول ایپاک‌ها عملکردش بهتر شده است که یعنی مدل روند خوبی را طی کرده است. اگرچه مدل کمی overfit کرده است. و بیشتر مدل روی داده train پیشرفت داشته است تا داده validation.

۳-۲. ارزیابی روی دادگان sentipers

برای ارزیابی مدل روی این دادگان مرحله forward را انجام می‌دهیم بدون اینکه وزن‌ها آپدیت شود. نتایج ارزیابی دادگان test روی مدل در جدول زیر آمده است.

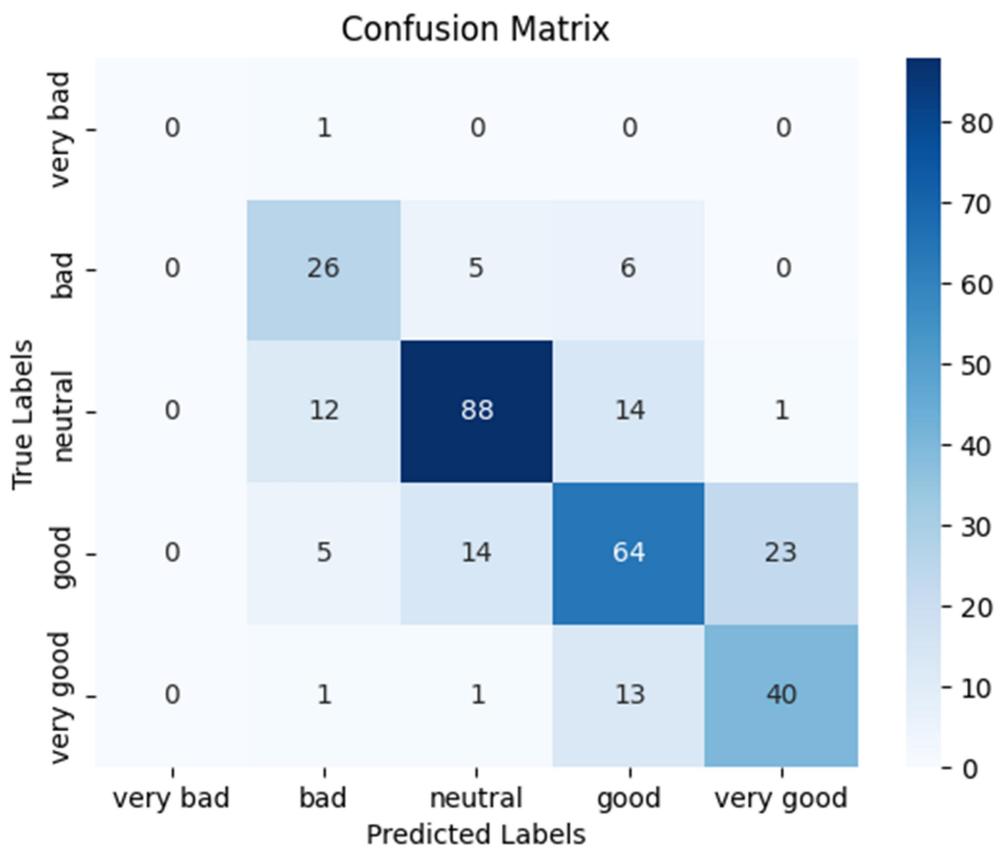
Metrics	values
Accuracy	69.43
Precision	69.87
Recall	69.43
F1-score	69.44

شكل ۳-۳. ارزیابی مدل LaBSE روی دادگان sentipers

دقت مدل با توجه به اینکه تنها تعداد کمی ایپاک آموزش دیده شده است قابل قبول است.

۴-۲. رسم ماتریس درهم ریختگی برای داده Sentipers

جدول درهم ریختگی به درک بهتر عملکرد مدل کمک می‌کند تا بتوانیم حتی بهتر خروجی‌های مدل را تفسیر کنیم. در ادامه نمودار این جدول آورده شده است.



۳۴. ماتریس درهم ریختگی برای مدل LaBSE روی دادگان sentiper Figure

۵-۲. اجرای مدل روی دادگان اسنپ فود

رویکرد zero shot learning برای استفاده در پیش بینی دیتاستی که داده کم دارد یا وقتی نمی خواهیم دوباره مدل را finetune کنیم کاربرد دارد. در این روش از نتایج مدل روی task مرتبط با دیتاست بزرگ تر استفاده کرده و برای پیش بینی دیتاست جدید استفاده می کنیم و دیگر عمل train یا finetune نداریم.

برای این کار یک مدل جدید می‌سازیم که از مدل قبلی استفاده می‌کند و فقط یک لایه softmax در انتهای برای هندل کردن تفاوت تعداد لیبل‌ها استفاده می‌کند. برای اجرای آن هم دیگر مدل را train نمی‌کنیم بلکه از همان وزن‌های مدل قبل استفاده می‌کنیم. و یک تابع برای پیش‌بینی سنتیمنت متون داده شده از دیتابست اسنپ فود پیاده سازی می‌کنیم که از مدل استفاده کرده و در خروجی سنتیمنت را برگرداند. این کار را در تابع predict انجام دادیم.

۶-۲. ارزیابی مدل روی دادگان اسنپ فود

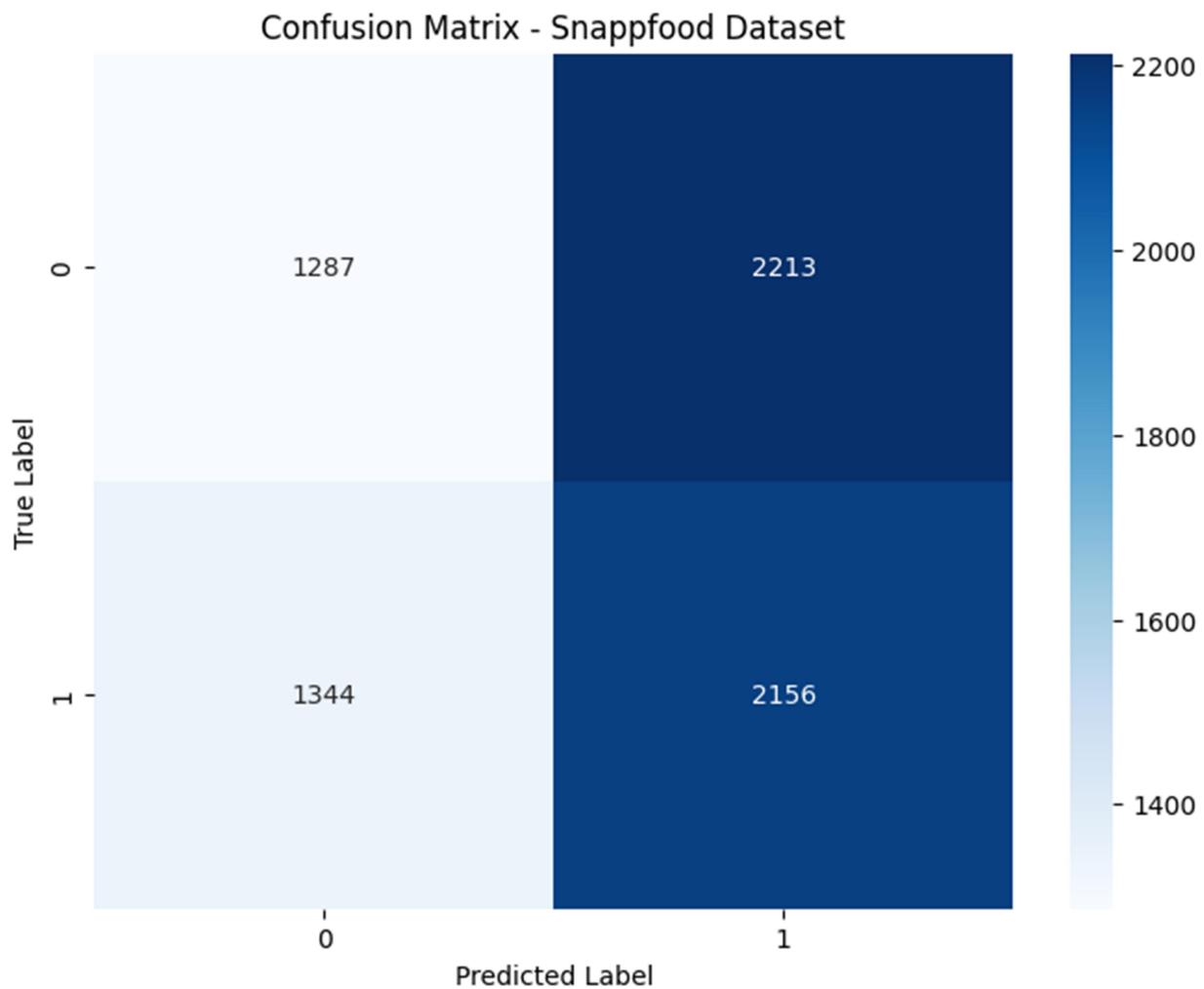
برای ارزیابی مدل از معیارهای مهم استفاده کردیم تا عملکرد مدل را روی دیتابست اسنپ‌فود مشاهده کنیم. که نتایج مهم‌ترین معیارها در جدول زیر آمده است.

Metrics	values
Accuracy	52.71
Precision	52.05
Recall	63.57
F1-score	57.34

شکل ۳۵. ارزیابی مدل ZLS در رویکرد LaBSE

۷-۲. رسم ماتریس درهم ریختگی روی دادگان اسنپ فود

مثل قبل برای درک بهتر عملکرد مدل از ماتریس درهم ریختگی استفاده می‌کنیم.



شکل ۳۶. ماتریس درهم ریختگی مدل LaBSE روی دادگان اسنپ فود

که خوب مشخص است تقریباً تصادفی انتخاب کرده است و فقط کمی بهتر از روش تصادفی پیش بینی کرده است.

۸-۲. نتیجه گیری

این روش شاید به اندازه fine-tuning خوب نباشد اما این کار را بدون آموزش انجام داده است و به دقت قابل قبولی رسیده است. پس در مواقعی که نخواهیم هزینه‌های لیبل زدن یا finetune را بدھیم انتخاب مناسبی می‌تواند باشد.

دقت شود که اگر در مرحله finetune کردن از دیتاست بزرگ‌تر و تعداد ایپاک بیشتر استفاده می‌کردیم احتمالاً در دقت خیلی تاثیر مثبت می‌گذاشت و بخشی از ضعف نتایج از همین کمبود ناشی شده است.

همچنین اگر چه دیتاست finetune با دیتاست تست یک task مشابه را انجام می‌دهد اما خود تفاوت تعداد لیبل‌ها می‌تواند در عملکرد مدل تاثیر منفی بگذارد.