به نام خدا

دوره کارآموزی هوش مصنوعی

پردازش زبان طبیعی

بخش چهارم پروژه - طبقه‌بندی با مدل‌های آماده

محدثه رهنما - زینب تقوی

آبان الی دی ۹۹

از آن‌جا که هدف این فاز طبقه‌بندی متن با استفاده از مدل‌های از پیش آموزش دیده است، بنابراین از مدل‌های مبتی بر bert استفاده می‌شود. بدین منظور در سایت hugging face برخی مدل‌های آموزش دیده با متون فارسی مورد استفاده قرار گرفته‌است. همچنین مدل چند زبانه‌ای (multi lingual) که زبان فارسی نیز در فهرست زبان‌های آموزش دیده‌اش موجود است، مورد بررسی قرار گرفته است.

در ابتدا تلاش شد تا با استفاده از کتاب‌خانه‌ی tensorflow پیاده سازی صورت گیرد که به دلیل عدم موفقیت کتاب‌خانه‌ی pytorch جایگزین گردید. معماری به کار رفته بدین صورت است که در ابتدا ورودی به bert اعمال شده و در نهایت خروجی bert برای طبقه‌بندی به کار گرفته می‌شود. در واقع خروجی برت به دو لایه‌ی تماماً متصل هدایت می‌شوند. واضح است که در آخرین لایه تعداد نرون‌ها به تعداد کلاس‌های موردنظر است. تعداد لایه‌ها و تعداد نرون‌ها نیز از پارامترهایی است که با آزمون و خطا به دست می‌آید.

به دلیل نتایج بسیار پایین و مقادیر تک رقمی درصد معیارهای ارزیابی، معماری از جهات مختلف بررسی شد که در ادامه بحث می‌شود:

- تابع loss: به دلیل ویژگی ذاتی مسئله‌ی موردنظر، مبنی بر این که هر نمونه می‌تواند به بیش از یک کلاس تعلق داشته باشد، بنابراین ضروری است تا از تابع loss مناسب استفاده شود. در pytorch برای محاسبه‌ی زیان به روش cross entropy با استفاده از یکی از توابع زیر امکان پذیر است:
  - BCEWithLogitsLoss
  - BCELoss

نکته‌ی قابل توجه این است که تفاوت این دو روش در نوع مقادیر ورودی است. در BCEWithLogitsLoss باید مقادیر نرون‌های لایه‌ی آخر بدون اعمال تابع سیگموید به تابع داده شود. زیرا در پیاده سازی این تابع سیگموید اعمال شده و سپس binary cross entropy محاسبه می‌شود. در پایان هنگام تست مدل نیز باید تابع سیگموید به خروجی مدل آموزش دیده اعمال شود تا عملیات از نظر منطقی درست باشد. در مقابل در BCELOSS باید مقدار نرون‌های لایه‌ی آخر پس از اعمال سیگموید به عنوان ورودی به این تابع داده شود.

- بهینه ساز: واضح است که روش بهینه‌سازی تأثیر بسزایی در همگرایی وزن‌ها دارد. به طوری که ممکن است برخی روش‌ها نتنها موجب کاهش زیان (loss) نشده بلکه سبب افزایش آن شود و بنابراین

میزان loss در طول فرایند آموزش بدون تغییر باقی بماند. بدین منظور از روش stochastic gradient descent(sgd) استفاده شد.

- محاسبه‌ی وزن کلاس‌ها: به دلیل عدم تعادل در نمونه‌های هر کلاس، بهتر است تابع loss برای هر کلاس با در نظر گرفتن وزن محاسبات را انجام دهد. برای محاسبه‌ی وزن در مسائل چند کلاسه، توابع آماده‌ای موجود است. اما از آنجا که مسئله‌ی موردنظر multi label است، این محاسبات بدون استفاده از تابع آماده انجام شد و وزن هر کلاس به دست آمد. روش محاسبه‌ی وزن‌ها به این ترتیب است که برای تعداد نمونه‌های هر کلاس بر تعداد کل نمونه‌ها به دست آمده و وزن‌های به دست آمده به تنسور تبدیل می‌شود. ذکر این نکته ضروری است که در این حالت باید نوع تنسورهای وزن و تنسورهای برچسب‌ها (لایه‌ی آخر) از نوع float باشد تا خطای محاسباتی رخ ندهد.

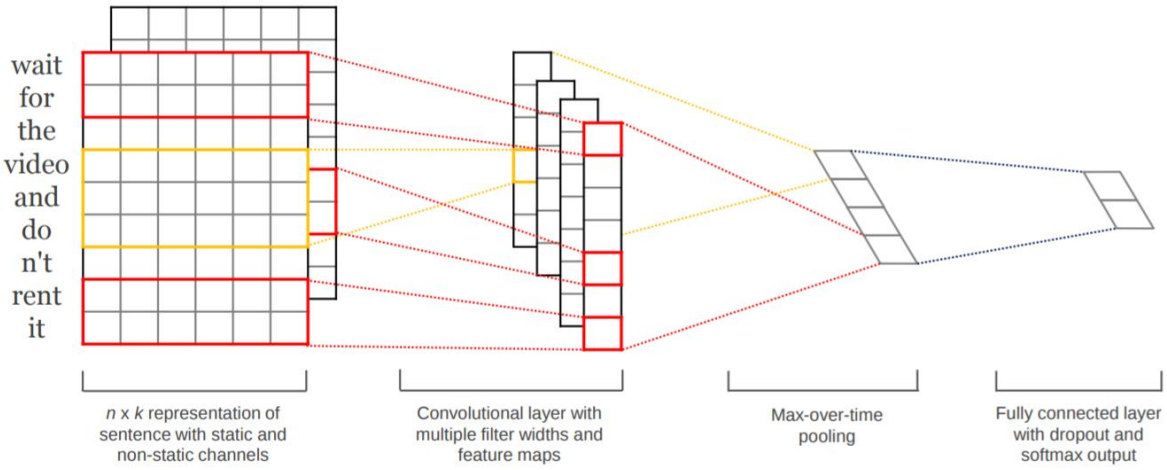
- پیش پردازش: به دلیل مشکلات پردازش زبان فارسی، انجام پیش پردازش امری ضروری است. در میانه‌ی کار مشخص شد که عدم انجام پیش پردازش طبق مدل آماده‌ی bert فارسی، یکی از مهم‌ترین دلایل نتایج پایین در این فاز است. بنابراین با بررسی بیشتر و مشاهده‌ی نمونه‌ای در گیت‌هابِ مدلِ parsbert، پیش پردازش متون مشابه پردازش‌های مدل آماده انجام شد. بدین منظور از کتابخانه‌های hazm و cleantext استفاده گردید. همین امر مقدار f1-score را از حدود ۵ درصد به حدود ۲۸ درصد افزایش داد.

- نرخ یادگیری: یکی از مهم‌ترین پارامترهایی که در همگرایی شبکه مؤثر است نرخ یادگیری است که معمولاً دو استراتژی برای تنظیم آن به کار گرفته می‌شود: ساده‌ترین راه این است که نرخ یادگیری در طول فرآیند یادگیری ثابت باشد و راه دیگر این است که در ابتدا نرخ یادگیری زیاد بوده (نزدیک یک) و حین آموزش به تدریج کاهش یابد. بدین ترتیب نرخ یادگیری با هر دو استراتژی تنظیم شد و در نهایت راه دوم انتخاب گردید.

- افزودن لایه‌ی batch normal: برای تسریع در همگرایی شبکه و یکسان کردن توزیع ورودی و خروجی هر لایه، این کار انجام شد.

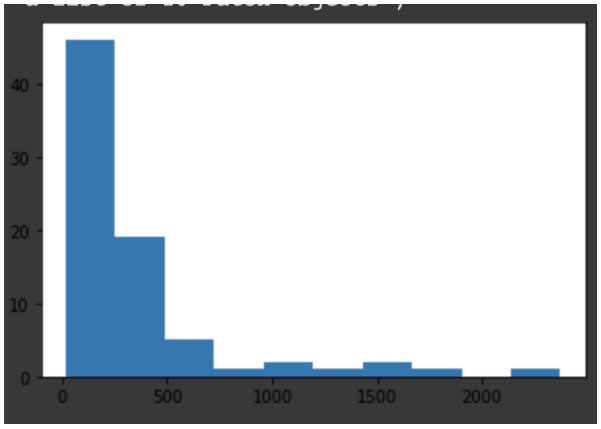
- با تغییر تعداد لایه‌ها و افزایش آن‌ها به سه و چهار لایه، این نتیجه حاصل شد که افزایش تعداد پارامترها بدین شیوه در بهبود نتایج تأثیر ندارد. بنابراین به نظر می‌رسد دو لایه برای این مسئله مناسب‌تر است و باید معماری از جنبه‌های دیگری بررسی گردد.

- در بررسی‌های انجام شده مشخص شد که مقدار بیشتر نرون‌ها در هنگام آموزش برابر با صفر می‌شوند. علت اصلی این است که اکثر مقادیر خروجی برت منفی هستند و هنگامی که مقادیر منفی از تابع فعال کننده‌ی relu عبور می‌کنند سبب صفر شدن نرون‌ها شده و در نتیجه صفر شدن گرادیان خواهد شد. برای جلوگیری از این مشکل از تابع فعال کننده‌ی elu در لایه‌های پنهان استفاده شد.
- اضافه کردن لایه‌ی cnn: در فاز قبل شبکه‌ی پیشنهادی در مقاله (Kim, 2014) پیاده سازی شد:



به دلیل این که معماری فوق در فاز قبل نتایج خوبی داشت، تلاش شد تا معماری مشابه آن پیاده سازی شود. به طوری که تفاوت آن با فاز قبل در نوع ورودی‌های شبکه باشد. یعنی در فاز قبل مقادیر embedding جملات به لایه‌ی کانولوشن وارد می‌شد و حال در این مرحله خروجی bert وارد لایه‌ی کانولوشن شود و سپس با اعمال pooling و چند لایه‌ی تماماً متصل خروجی‌ها به دست آیند. گرچه نتایج به دست آمده از این معماری نیز مناسب نبوده‌است.

- به دلیل نامتعادل بودن داده‌های هر کلاس و ذات مسئله -که هر نمونه می‌تواند متعلق به بیش از یک کلاس باشد- بنابراین به نظر می‌رسد یکی از مهم‌ترین دلایلی که شبکه نمی‌تواند به نتایج قابل قبولی دست یابد داده‌ها هستند. همچنین طول متن ورودی به bert نیز از پارامترهایی است که در عملکرد معماری تأثیرگذار است. با توجه به این که bert در تعداد ورودی‌ها محدودیت داشته و حداکثر طول جملات برابر با ۵۱۲ است، این مقدار به عنوان طول جملات در نظر گرفته شد. لازم به ذکر است که طول جملات داده‌ها طبق هیستوگرام زیر بیشتر از این مقدار است:

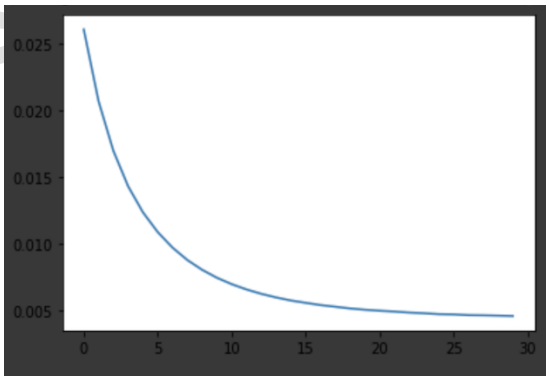


در ادامه داده‌ها محدودتر شدند و این کار با این هدف انجام شد تا از صحت معماری و برنامه‌ی نوشته شده اطمینان حاصل شود. بدین منظور تنها کلاس‌هایی انتخاب شد که نمونه‌های بیشتری از آن در اختیار است و کلاس‌های با نمونه‌های کمتر کنار گذاشته شد.

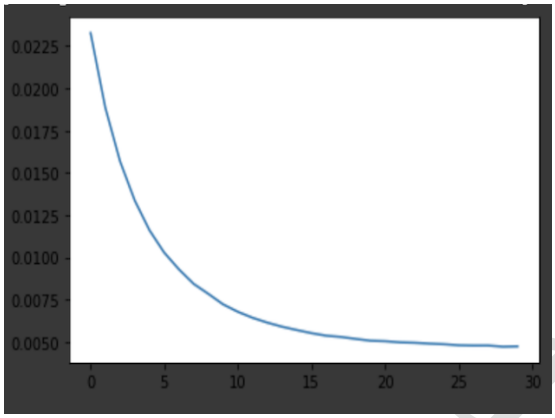
با بررسی تعداد نمونه‌های هر کلاس، با توجه به این که میانگین تعداد نمونه‌ها تقریباً برابر با ۷۵۰ نمونه است، چند بازه برای محدود سازی کلاس‌ها در نظر گرفته شد. بدیهی است که با این کار تعدادی از کلاس‌ها حذف می‌شوند؛ اما پیشتر بیان شد که این کار تنها با هدف اطمینان از صحت شبکه انجام می‌شود. بدین ترتیب سه دسته محدودیت برای کلاس‌ها اعمال شد:

۱. کلاس‌هایی که تعداد نمونه‌های آن‌ها بین ۳۲۵ تا ۷۵۰ نمونه باشد: با اعمال این محدودیت ۳۶ کلاس به دست آمد. در این حالت نمودارهای تابع زیان در حین آموزش به صورت زیر است. (epoch=30)

نمودار train_loss:



نمودار validation_loss :

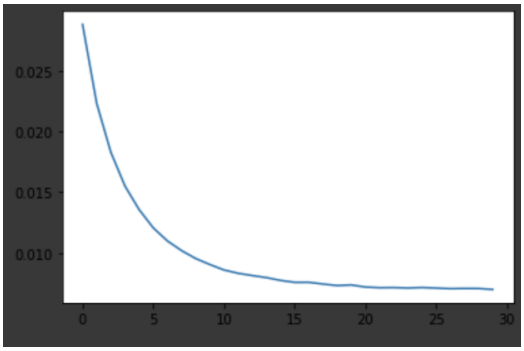


نتایج به دست آمده بر اساس تابع classification_report:

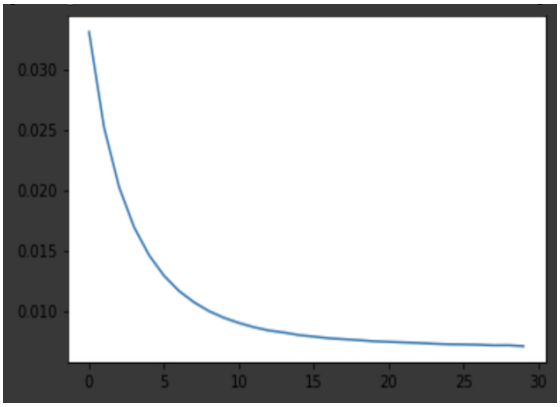
	precision	recall	F1-score
Micro avg	٪۲۵	٪۵۴	٪۳۵
Macro avg	٪۲۶	٪۵۲	٪۳۴
Weighted avg	٪۲۹	٪۵۴	٪۳۵

۲. کلاس‌هایی که تعداد نمونه‌های آن‌ها بین ۷۵۰ تا ۱۵۰۰ نمونه باشد (تعداد کلاس‌ها = ۲۶):

نمودار train_loss:



نمودار validaton_loss:

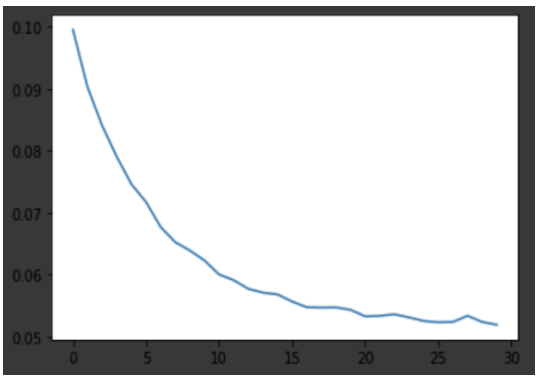


نتایج به دست آمده بر اساس تابع classification_report:

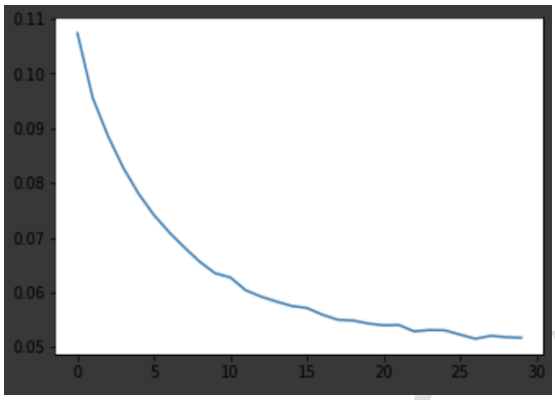
	precision	recall	F1-score
Micro avg	٪۳۵	٪۵۷	٪۴۳
Macro avg	٪۳۶	٪۵۶	٪۴۳
Weighted avg	٪۳۷	٪۵۷	٪۴۴

۳. کلاس‌هایی که تعداد نمونه‌های آن‌ها بین ۱۱۲۵ تا ۱۵۰۰ نمونه باشد (تعداد کلاس‌ها=۶): با توجه به این که نتایج به دست آمده در حالتی که کلاس‌ها نمونه‌هایی بین ۷۵۰ تا ۱۵۰۰ تا داشته باشند بهتر شده، تلاش شد تا این محدوده کمتر در نظر گرفته شود تا نتیجه‌ی حاصل از مجموعه‌ی داده‌ی محدودتر مشاهده شود:

نمودار train_loss:



نمودار validation_loss :



نتایج به دست آمده بر اساس تابع classification_report:

	precision	recall	F1-score
Micro avg	٪۶۵	٪۵۰	٪۵۶
Macro avg	٪۶۴	٪۵۰	٪۵۶
Weighted avg	٪۶۳	٪۵۰	٪۵۵

متاسفانه نتایج به دست آمده در این فاز بهتر از فاز قبل نبوده و به نظر می‌رسد در این مسئله و با این داده‌ها روش from scratch نتایج بهتری داشته است. تنها با کوچک کردن مجموعه داده یافته‌های به نسبت بهتری به دست آمد.