به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین امتیازی

e

نام و نام خانوادگی	پوریا تاج محرابی – سپهر آزردار
شماره دانشجویی	810199357 - 810199395
تاریخ ارسال گزارش	۱۴۰۲.۱۰.۲۱

فهرست

شکل‌ها

پاسخ ۲. Lora

۱.۲. LoRA چگونه عمل میکند؟

- **fine-tuning تمام پارامتر ها**

بک روش مرسوم برای adapt کردن مدل های pre-trained برای تسک جدید میباشد. در این روش مدل از قبل به روی دادگان بسیار بزرگی آموزش دیده است تا رفتار ها و representation ها گسترده ای را یاد بگیرد. در این روش مدل به روی تسک کوچک و دیتا خاص و منحصر به فرد تسک آموزش میبیند. با اینکار تمامی پارامتر های مدل در راستای تسک فعلی اصلاح می شوند. این روش به علت اینکه تمامی لایه ها را درگیر میکند به شدت resource-intensive هست که هم شامل منابع محاسباتی برای انجام محاسبات زیاد برای آپدیت کردن تمامی پارامتر ها می باشد و همچنین از آنجایی که تمامی پارامترها آپدیت میشوند باید نسخه آپدیت شده آن ها را نگه داریم پس memory-intensive هم میباشد. ولی از طرفی به خاطر اینکه پارامترهای مدل در این روش آپدیت میشوند به صورت خیلی خوبی به روی تسک جدید عمل میکند.

- **fine-tune کردن یک یا چند لایه از لایه های مدل**

در این روش تنها تعدادی از لایه ها را آپدیت میکنیم و به اصطلاح بقیه لایه ها را freeze می کنیم و آنها را آپدیت نمیکنم. اینکار باعث کاهش محاسبات و ممووری لازم برای نگه داشتن نسخه tune شده مدل میشود. این روش در خصوص مدل ها بزرگ کاربرد دارد زیرا که آنها از تعداد زیادی پارامتر برخوردار هستند. در این روش معمولا لایه ها نزدیک به خروجی را tune می کنیم زیرا این لایه ها تاثیر بیشتر و ارتباط بیشتری با تسک مورد نظر دارند. لایه ابتدایی freeze میشوند بدین معنا که وزن آنها تغییر نمیکند، علت این امر این است که این لایه مسئولیت یادگیری پترن های general را دارند و تغییر ندان آن ها منطقی است. مدل به روی دیتاست تسک مورد نظر ترین میشود ولی تنها لایه های unfrozen آپدیت میشوند. این روش به دلایل گفته شده کمتر از حالت قبلی resource-intensive میباشد. در بعضی موارد مخصوصا زمانی که داده تسک مورد نظر بسیار کوچک می باشد fine-tuning کردن تمامی \ارامتره ا باعث overfitting میشود. با ثابت نگه داشتن اکثر مدل از این پدیده جلوگیری میشود. با اینکار مدل generalization خود را حفظ میکند. و این روش سریعتر است به علت اینکه محاسبات کمتری انجام میشود و پارامترهای کمتری آپدیت میشوند. و همچنین deployment این روش سریع تر است زیرا که تنها بخشی از مدل باید برای تسک جدید تغییر کند و آن را به طور جداگانه ذخیره کنیم.

- **LoRA(Low-Rank Adaptation)**

این روش برای آموزش بهینه و resource-efficient مدل های زبانی بزرگ طراحی شده است.
به جای آپدیت کردن وزن های اصلی خود مدل، LoRA مارتیس های کوچک و قابل آموزش low-rank ای را معرفی میکند که تغییرات وزن های مدل را در در خود نگه میدارد.
پدر این روش وزن های مدل pre-trained فریز میشوند و آنها در فاز fine-tuning آپدیت نمیشوند.
این روش می تواند به لایه های خاصی اعمال شود. برای مثال لایه های self-attention و feedforward در transformer ها.

نحوه کار:

وزن های یک لایه در واقع یک ماتریس هستند. برای یک مثال برای یک ماتریس وزن، مانند Q,K,V در transformer ها، LoRA دو ماتریس low-rank به اسم های A,B اضافه میکند. سپس، **ماتریس وزن اصلی W به صورت W+BA آپدیت میشود.** که W همان ماتریس وزن لایه مورد نظر از مدل PRE-TRAINED هست و A,B دو ماتریس LOW-RANK ارائه شده توسط LoRA میباشد.

در حین آموزش، فقط ماتریس‌های A و B آموزش دیده می‌شوند. تغییراتی که آن‌ها تجربه می‌کنند به طور موثر نمایانگر سازگاری مدل با وظیفه جدید هستند، در حالی که بخش اعظم مدل (وزن‌های اصلی) بدون تغییر باقی می‌ماند.
این رویکرد تعداد پارامترهایی که نیاز به آموزش دارند را به شکل قابل توجهی کاهش می‌دهد، که منجر به کاهش نیازهای محاسباتی و حافظه می‌شود.

LoRA امکان سازگاری سریع مدل‌ها با وظایف و حوزه‌های جدید را با حداقل بار محاسباتی فراهم می‌آورد. همچنین امکان تغییر سریع بین وظایف را فراهم می‌کند، زیرا فقط ماتریس‌های کوچک LoRA نیاز به تعویض دارند.

از آنجا که مدل پایه بدون تغییر باقی می‌ماند، درک زبان عمومی و توانایی‌های آن حفظ می‌شود.

مدل سازگار شده (با ماتریس‌های LoRA) نسبت به مدل پیش‌آموزش دیده اصلی تاخیر اضافی در استنتاج ایجاد نمی‌کند.

LoRA به ویژه زمانی مفید است که با مدل‌های بسیار بزرگی سر و کار داریم که fine-tuning سنتی برای آن‌ها بیش از حد منابع‌بر خواهد بود یا زمانی که multiple task-specific adaptations از یک مدل پایه واحد مورد نیاز باشد. خلاصه اینکه، LoRA با معرفی ماتریس‌های کم‌رتبه کوچک و قابل آموزش که ماهیت تغییرات مورد نیاز برای تنظیم ریز را ضبط می‌کنند، کار می‌کند، در حالی که اکثریت وزن‌های مدل پیش‌آموزش دیده را منجمد نگه می‌دارد. این رویکرد تعادلی بین سازگار کردن یک مدل با وظایف جدید و حفظ کارآمدی از نظر منابع محاسباتی برقرار می‌کند.

۲.۲. دیتاست و تسک مورد نظر

- توضیح دیتاست Multi-genre NLI (MultiNLI) Corpus:

دیتاست MultiNLI (استنتاج زبان طبیعی چند ژانری) هدفش فراهم کردن منبعی برای توسعه و ارزیابی مدل‌های یادگیری ماشینی در زمینه فهم زبان طبیعی است، به ویژه از طریق استنتاج زبان طبیعی. **این تسک شامل تعیین رابطه بین جفت جملات است که به عنوان استنتاج، تناقض یا خنثی برچسب‌گذاری شده‌اند.**

MultiNLI به دلیل دامنه متنوعی از ژانرهای متن نوشتاری و گفتاری خود، شامل ده سبک متمایز، برجسته است. این گستردگی هدف دارد تا سیستم‌ها را در مقایسه با دیتاست‌های قبلی در چالش و ارزیابی پیچیدگی بیشتری از زبان قرار دهد و محیطی برای تطبیق دامنه‌ای فراژانر (cross-genre domain adaptation) فراهم کند. این دیتاست یکی از بزرگترین دیتاست‌های موجود برای تسک های **NLI** است، شامل ۴۳۳ هزار نمونه و طراحی شده تا چالش‌برانگیزتر باشد و پدیده‌های زبانی و طول جملات گسترده‌تری را پوشش دهد.

به طور خلاصه:
ازاین دیتاست برای ساخت و سنجش سیستم های فهم زبان طبیعی مانند مدل RoBERTa استفاده میشود. این دیتاست مربوط به تسک (Natural Language Inference) یا به اختصار NLI هست. که در آن یک جفت جمله به مدل داده میشود و از مدل خواسته میشود که رابطه بین این دو را بفهمد. پس جنس داده ها مجموعه ای جفت جملات هست که در ژاندر ها مختلف هستند و یک لیبل که نشان دهند رابطه بین این دو جمله هست. تسک classificaiton هست و همان زور که بالاتر گفته شد ۳ تا کلاس داریم. پس MultiNLI شامل تعداد زیادی جفت جمله است که یکی از این رابطه بین جفت ها برقرار هست. جملات از انواع مختلف ژانرهای متن گفتاری و نوشتاری گرفته شده‌اند که این امر آن را به یک دیتاست متنوع و چالش‌برانگیز تبدیل می‌کند. این دیتاست شامل

ژانرهایی مانند داستان، اسناد دولتی و مکالمات تلفنی است که توانایی مدل را در فهم زبان در زمینه‌های مختلف افزایش می‌دهد. همچنین از این دیتاست برای آموزش و ارزیابی مدل‌ها بر اساس توانایی آن‌ها در فهم و استنتاج روابط بین جملات استفاده می‌شود.

MultiNLI اغلب به عنوان یک دیتاست معیار در جامعه پردازش زبان طبیعی (NLP) استفاده می‌شود. مدل‌هایی مانند RoBERTa در برابر آن آزمایش می‌شوند تا عملکرد و توانایی‌های آن‌ها در فهم زبان طبیعی سنجیده شود.

به طور خلاصه، دیتاست MultiNLI یک منبع کلیدی برای آموزش و ارزیابی مدل‌های NLP مانند RoBERTa است، به ویژه در حوزه استنتاج زبان طبیعی(NLI). محتوای متنوع آن از چندین ژانر آن را به یک ابزار مهم برای ارزیابی قابلیت تعمیم و مقاومت چنین مدل‌هایی تبدیل می‌کند.

- نحوه آموزش مدل به روی دیتاست مورد نظر

برای آموزش مدل RoBERTa با هدف این دیتاست(**NLI**)، احتمالاً از رویکرد یادگیری چند وظیفه‌ای (**multi-task learning approach**) استفاده می‌شود، جایی که مدل در طول آموزش به نمونه‌هایی از ژانرهای مختلف معرفی می‌شود، و توانایی آن را در تعمیم بین سبک‌ها و دامنه‌های زبانی مختلف افزایش می‌دهد.

برای تنظیم دقیق (fine-tuning) مدل‌هایی مانند RoBERTa با استفاده از رویکرد LoRA، مکانیزم توجه (**attention mechanism**) باید آپدیت بشود. این کار با معرفی ماتریس‌های کم‌رتبه **(low-rank)** انجام می‌شود که وزن‌های self-attention را به شکلی تطبیق می‌دهد که روابط بین ژانرهای مختلف حاضر در دیتاست MultiNLI را ضبط می‌کند. این امر آپدیتهای کارآمد پارامترها که مخصوص adaptation task هستند، را بدون افزایش قابل توجه در تعداد پارامترها امکان‌پذیر می‌سازد.

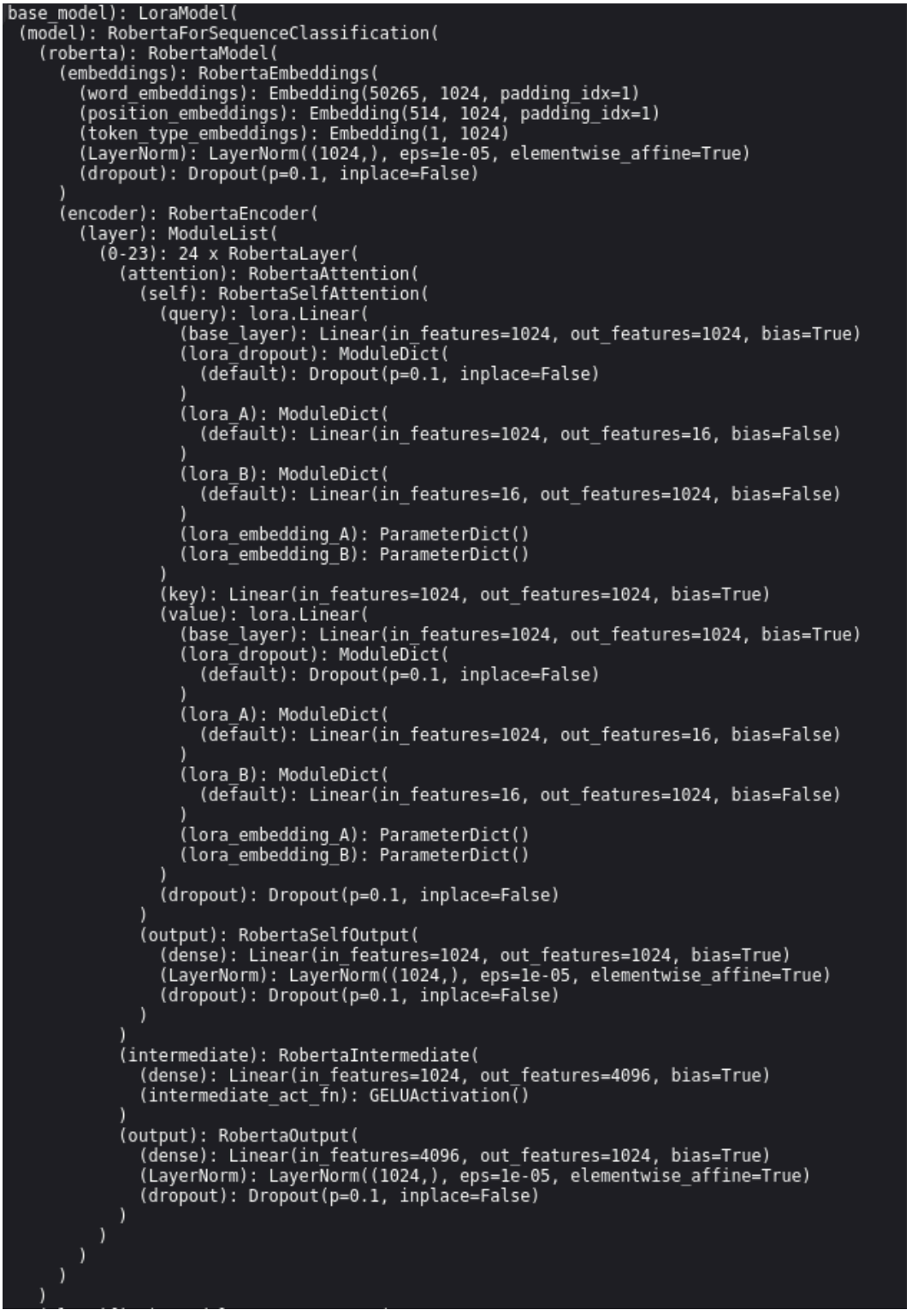
در تسک NLI که هدف دیتاست MultiNLI میباشد، **ورودی به مدل** معمولاً **یک جفت جمله است: یک پیش‌فرض و یک فرضیه. خروجی طبقه‌بندی رابطه** بین این جملات در یکی از سه دسته است: استنتاج، تناقض یا خنثی. تسک NLI دراینجا در واقع یک تسک classification هست. و باید ماتریس خروجی مدل یا high-dimensional representations مدل را از یک mlp عبور دهیم تا پیش بینی رابطه بین دو جمله را انجام دهیم.

در اینجا نحوه استفاده از خروجی مدل آمده است:

- لایه MLP: پس از پردازش جملات ورودی توسط مدل، می‌توانید یک پرسپترون چند لایه (MLP) را روی خروجی مدل اعمال کنید.
  این MLP به عنوان یک طبقه‌بند عمل می‌کند که
  نمایش‌های بعد بالا (high-dimensional representations) تولید شده توسط RoBERTa را می‌گیرد **و توزیع احتمال بر روی سه کلاس را خروجی می‌دهد.**
- تنظیم دقیق با LoRA: برای اعمال LoRA(همان طور که بالاتر در قسمت توضیح LoRA گفته شد)، باید **(attention mechanism)** مکانیزم توجه RoBERTa را با **اضافه کردن ماتریس‌های کم‌رتبه تنظیم** کنید. این به طور موثر پارامترهای مدل را با افزایش حداقلی در تعداد پارامترها تطبیق می‌دهد، که آن را در وظیفه خاص NLI بدون افزایش قابل توجه بار محاسباتی ماهرتر می‌کند.

استفاده از LoRA امکان تطبیق هدفمند مدل را ضمن حفظ دانش پیش‌آموزش دیده‌ای که RoBERTa کسب کرده است، فراهم می‌کند، که برای حفظ عملکرد در سراسر وظایف زبانی متنوع حیاتی است. پس از اعمال LoRA، باید با آموزش مدل روی دیتاست MultiNLI ادامه دهید، با استفاده از جفت‌های پیش‌فرض و فرضیه به عنوان ورودی‌ها و برچسب‌های توضیح داده شده به عنوان اهداف برای یادگیری از آن‌ها.

۳.۲. آموزش مدل



```
base_model): LoraModel(
 (model): RobertaForSequenceClassification(
   (roberta): RobertaModel(
     (embeddings): RobertaEmbeddings(
       (word_embeddings): Embedding(50265, 1024, padding_idx=1)
       (position_embeddings): Embedding(514, 1024, padding_idx=1)
       (token_type_embeddings): Embedding(1, 1024)
       (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
       (dropout): Dropout(p=0.1, inplace=False)
     )
     (encoder): RobertaEncoder(
       (layer): ModuleList(
         (0-23): 24 x RobertaLayer(
           (attention): RobertaAttention(
             (self): RobertaSelfAttention(
               (query): lora.Linear(
                 (base_layer): Linear(in_features=1024, out_features=1024, bias=True)
                 (lora_dropout): ModuleDict(
                   (default): Dropout(p=0.1, inplace=False)
                 )
                 (lora_A): ModuleDict(
                   (default): Linear(in_features=1024, out_features=16, bias=False)
                 )
                 (lora_B): ModuleDict(
                   (default): Linear(in_features=16, out_features=1024, bias=False)
                 )
                 (lora_embedding_A): ParameterDict()
                 (lora_embedding_B): ParameterDict()
               )
               (key): Linear(in_features=1024, out_features=1024, bias=True)
               (value): lora.Linear(
                 (base_layer): Linear(in_features=1024, out_features=1024, bias=True)
                 (lora_dropout): ModuleDict(
                   (default): Dropout(p=0.1, inplace=False)
                 )
                 (lora_A): ModuleDict(
                   (default): Linear(in_features=1024, out_features=16, bias=False)
                 )
                 (lora_B): ModuleDict(
                   (default): Linear(in_features=16, out_features=1024, bias=False)
                 )
                 (lora_embedding_A): ParameterDict()
                 (lora_embedding_B): ParameterDict()
               )
               (dropout): Dropout(p=0.1, inplace=False)
             )
             (output): RobertaSelfOutput(
               (dense): Linear(in_features=1024, out_features=1024, bias=True)
               (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
               (dropout): Dropout(p=0.1, inplace=False)
             )
           )
           (intermediate): RobertaIntermediate(
             (dense): Linear(in_features=1024, out_features=4096, bias=True)
             (intermediate_act_fn): GELUActivation()
           )
           (output): RobertaOutput(
             (dense): Linear(in_features=4096, out_features=1024, bias=True)
             (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
             (dropout): Dropout(p=0.1, inplace=False)
           )
         )
       )
     )
   )
```

ساختار مدل در صورت استفاده از روش LoRA

برای ماتریس ها Q,K,V در لایه های SELF-ATTENTION مدل، ماتریس ها مرتبه پایین روش LORA را افزوده ایم.

```
)
Number of trainable parameters: 2625539
```

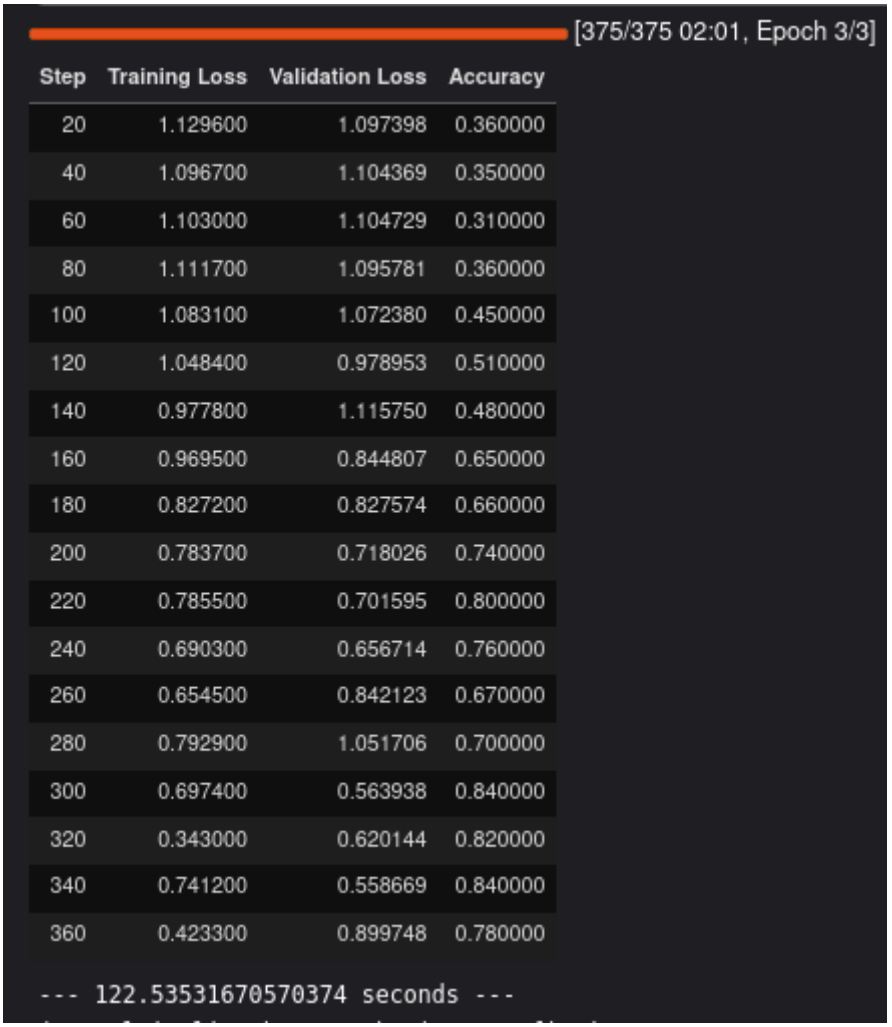
تعداد پارامتر های قابل تغییر در مقایسه با fine-tuning معمولی که برابر با کل پارامترهای مدل هست(355,362,819) بسیار کمتر میباشد. به **نسبت ۱۵۰ ام کاهش** داشته است.

```
RobertaForSequenceClassification(
  (roberta): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(50265, 1024, padding_idx=1)
      (position_embeddings): Embedding(514, 1024, padding_idx=1)
      (token_type_embeddings): Embedding(1, 1024)
      (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): RobertaEncoder(
      (layer): ModuleList(
        (0-23): 24 x RobertaLayer(
          (attention): RobertaAttention(
            (self): RobertaSelfAttention(
              (query): Linear(in_features=1024, out_features=1024, bias=True)
              (key): Linear(in_features=1024, out_features=1024, bias=True)
              (value): Linear(in_features=1024, out_features=1024, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): RobertaSelfOutput(
              (dense): Linear(in_features=1024, out_features=1024, bias=True)
              (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): RobertaIntermediate(
            (dense): Linear(in_features=1024, out_features=4096, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): RobertaOutput(
            (dense): Linear(in_features=4096, out_features=1024, bias=True)
            (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
  (classifier): RobertaClassificationHead(
    (dense): Linear(in_features=1024, out_features=1024, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=1024, out_features=3, bias=True)
  )
)
355362819
```

معماری مدل در حالت استفاده از fine-tuning (بدون تغییر باقی میماند.)

[22/73632 00:09 < 9:42:32, 2.11 it/s, Epoch 0.00/3]

Step	Training Loss
10	1.087900
20	1.115500


[375/375 02:01, Epoch 3/3]

Step	Training Loss	Validation Loss	Accuracy
20	1.129600	1.097398	0.360000
40	1.096700	1.104369	0.350000
60	1.103000	1.104729	0.310000
80	1.111700	1.095781	0.360000
100	1.083100	1.072380	0.450000
120	1.048400	0.978953	0.510000
140	0.977800	1.115750	0.480000
160	0.969500	0.844807	0.650000
180	0.827200	0.827574	0.660000
200	0.783700	0.718026	0.740000
220	0.785500	0.701595	0.800000
240	0.690300	0.656714	0.760000
260	0.654500	0.842123	0.670000
280	0.792900	1.051706	0.700000
300	0.697400	0.563938	0.840000
320	0.343000	0.620144	0.820000
340	0.741200	0.558669	0.840000
360	0.423300	0.899748	0.780000

--- 122.53531670570374 seconds ---

دقت و مدت زمان آموزش مدل در حالت fine-tuning به روی تمامی پارامترهای مدل.

۴.۲. چرا LoRA؟

رویکرد LoRA (تطبیق کم‌رتبه) و تنظیم دقیق سنتی دو روش برای تطبیق و بهینه‌سازی مدل‌های پیش‌آموزش دیده برای وظایف خاص هستند. این روش‌ها به ویژه در زمینه مدل‌های مقیاس بزرگ، مانند آن‌هایی که در پردازش زبان طبیعی استفاده می‌شوند، مرتبط هستند. در اینجا مقایسه‌ای بین این دو آمده است:

رویکرد LoRA

- مفهوم: LoRA با افزودن ماتریس‌های کم‌رتبه به وزن‌های لایه‌های منتخب در حین آموزش، یک مدل پیش‌آموزش دیده را تغییر می‌دهد.
- تنظیم پارامتر: به جای **تنظیم تمام پارامترهای مدل**، LoRA بر روی یک **مجموعه کوچکتر از پارامترهای اضافی تمرکز می‌کند**، که بار محاسباتی و اثر حافظه را کاهش می‌دهد.
- کارآمدی: نسبت به تنظیم ریز کامل مدل، به ویژه برای مدل‌های بزرگ، کارآمدتر است زیرا شامل به‌روزرسانی کمتری از پارامترها می‌شود.
- سرعت آموزش و نیازمندی‌های منابع: **آموزش سریع‌تر و** نیازمندی‌های کمتر منابع به دلیل به‌روزرسانی کمتر پارامترها.
- انطباق‌پذیری و تعمیم‌پذیری: LoRA می‌تواند بسیاری از دانش عمومی از مدل پیش‌آموزش دیده را حفظ کند در حالی که به وظایف خاصی تطبیق می‌یابد. با این حال، اثربخشی آن ممکن است بسته به پیچیدگی وظیفه و معماری مدل پایه متفاوت باشد.
- موارد استفاده: ایده‌آل برای سناریوهایی که منابع محاسباتی محدود هستند یا هنگام کار با مدل‌های بسیار بزرگ که تنظیم ریز کامل امکان‌پذیر نیست.

تنظیم ریز سنتی

- مفهوم: تنظیم ریز شامل تنظیم **تمام یا بخش قابل توجهی از پارامترهای مدل** پیش‌آموزش دیده در فرآیند آموزش بر روی دیتاست ویژه وظیفه است.
- تنظیم پارامتر: این رویکرد می‌تواند شامل به‌روزرسانی تمام پارامترهای مدل (تنظیم ریز کامل) یا فقط یک زیرمجموعه (تنظیم ریز جزئی) باشد.
- کارآمدی: نسبت به LoRA برای مدل‌های بزرگ کمتر کارآمد است، زیرا نیازمند به‌روزرسانی تعداد بیشتری از پارامترها است.
- سرعت آموزش و نیازمندی‌های منابع: به طور کلی کندتر و نیازمند منابع بیشتر، به ویژه برای مدل‌های بزرگتر، به دلیل به‌روزرسانی بالاتر تعداد پارامترها.
- انطباق‌پذیری و تعمیم‌پذیری: تنظیم ریز امکان تطبیق گسترده‌تر با وظایف خاص را فراهم می‌کند که ممکن است به عملکرد بهتر در آن وظایف منجر شود. با این حال، ممکن است همچنین منجر به بیش‌برازش شود، به ویژه با دیتاست‌های کوچک.
- موارد استفاده: ترجیح داده می‌شود وقتی که منابع محاسباتی محدودیت عمده‌ای نیستند و وقتی وظیفه نیازمند تطبیق قابل توجه مدل پیش‌آموزش دیده است. همچنین استفاده می‌شود وقتی اندازه مدل قابل مدیریت است.

نتیجه‌گیری

- LoRA بیشتر برای سناریوهایی با منابع محاسباتی محدود یا برای مدل‌های بسیار بزرگ مناسب است و **بر روی به‌روزرسانی یک مجموعه کوچکتر از پارامترها برای تطبیق ویژه وظیفه تمرکز می‌کند.**

- تنظیم ریز سنتی جامع‌تر است، به‌روزرسانی بخش بزرگتر یا تمام پارامترهای مدل را شامل می‌شود که آن را برای وظایفی که نیازمند تطبیق قابل توجه مدل هستند مناسب می‌کند اما با هزینه بالاتر نیازمندی‌های محاسباتی.

Multi-Task Learning/Inference

در حالت fine-tune نمیتوانیم در چندتا تسک از یک مدل استفاده کنیم، مگر اینکه لاس را به گونه ای تعریف کنیم که مدل در هر دو تسک به خوبی عمل کند. در غیر این صورت برای هر تسک باید مدل را به طور کامل آموزش دهیم و تمام وزن های جدید برای تسک مورد نظر ذخیره کنیم در واقع در روش سنتی، مدل برای هر تسک، وزن های جداگانه ای دارد که همان طور که در سوال ۱و۲ گفته شد اینکار باعث مصرفی زیاد از ممووری و منابع محاسباتی میشود و deploy کردن چنین تسک های سخت میشود.

LoRA در یادگیری/استنباط چند وظیفه‌ای

- کارآمدی پارامتر: نقطه قوت LoRA در تنظیمات چند وظیفه‌ای، **کارآمدی پارامتری آن است.** از آنجایی که تنها تعداد کمی از پارامترها قابل آموزش را معرفی می‌کند، تطبیق یک مدل واحد برای چندین وظیفه بدون افزایش قابل توجه اندازه مدل، قابل مدیریت‌تر است.
- بار حافظه و محاسباتی: نیازمندی‌های کمتر حافظه و محاسبات LoRA را برای سناریوهایی با منابع محاسباتی محدود یا هنگام کار با مدل‌های بسیار بزرگ در چندین وظیفه جذاب می‌کند.
- انطباق‌پذیری: LoRA می‌تواند به طور موثر **به وظایف مختلف با تغییر فقط یک زیرمجموعه از پارامترهای مدل تطبیق یابد**. این می‌تواند به ویژه موثر باشد اگر وظایف از مشترکاتی بهره‌مند شوند که از دانش حفظ شده در پارامترهای تغییر نیافته مدل پیش‌آموزش دیده بهره‌مند شوند.
- تعمیم‌پذیری: این رویکرد ممکن است تعمیم‌پذیری بهتری را در سراسر وظایف حفظ کند، **زیرا خطر overfitting به** هر وظیفه تکی را با توجه **به تعداد محدود پارامترهای** به‌روزرسانی شده **کاهش** می‌دهد.
- دینامیک‌های آموزش: دینامیک‌های آموزش می‌توانند پایدارتر باشند، زیرا ساختار اساسی مدل عمدتاً بدون تغییر باقی می‌ماند.

تنظیم ریز سنتی در یادگیری/استنباط چند وظیفه‌ای

- کارآمدی پارامتر: در تنظیم ریز سنتی، پارامترهای بیشتری به‌روزرسانی می‌شوند که می‌تواند به افزایش اندازه مدل هنگام تطبیق برای چندین وظیفه منجر شود. این ممکن است نیاز به **مدل‌های جداگانه** یا مجموعه‌های پارامتر برای هر وظیفه داشته باشد که پیچیدگی و نیازمندی‌های منابع را افزایش می‌دهد.
- بار حافظه و محاسباتی: نیازمندی‌های منابع بالاتر به دلیل نیاز به به‌روزرسانی و ذخیره **تعداد بیشتری از پارامترها**. این می‌تواند محدودیت قابل توجهی در تنظیمات چند وظیفه‌ای باشد، به ویژه با مدل‌های بزرگ.
- انطباق‌پذیری: تنظیم **ریز امکان تطبیق دقیق‌تری را برای هر وظیفه خ**اص فراهم می‌کند که ممکن است به عملکرد بهتر در وظایف فردی منجر شود، به ویژه اگر آن‌ها متنوع باشند و نیازمند تغییرات قابل توجه مدل باشند.
- تعمیم‌پذیری: خطر overfitting به task مورد نظر وجود دارد، به ویژه با دیتاست‌های کوچکتر. این می‌تواند نگرانی در تنظیمات چند وظیفه‌ای باشد جایی که مدل باید عملکرد را در سراسر وظایف مختلف متعادل کند.
- دینامیک‌های آموزش: مدیریت تعادل بین وظایف چالش‌برانگیزتر می‌شود، زیرا تغییرات گسترده در مدل برای یک وظیفه ممکن است بر عملکرد در وظیفه دیگر تأثیر منفی داشته باشد.

نتیجه‌گیری

- LoRA برای یادگیری و استنباط چند وظیفه‌ای هنگام کار با مدل‌های بزرگ و منابع محاسباتی محدود مناسب است. کارآمدی پارامتری و خطر کمتر بیش‌برازش آن را برای متعادل کردن عملکرد در سراسر چندین وظیفه مزیت‌بخش می‌کند.
- تنظیم ریز سنتی تطبیق‌پذیری عمیق‌تری را برای هر وظیفه ارائه می‌دهد که ممکن است به عملکرد بالاتر در وظایف فردی منجر شود. با این حال، نیازمند منابع بیشتر و مدیریت دقیق برای جلوگیری از overfitting و متعادل کردن نیازمندی‌های مختلف هر وظیفه است.

در عمل، انتخاب بین LoRA و تنظیم ریز سنتی در تنظیمات چند وظیفه‌ای بستگی به نیازمندی‌های خاص وظایف، اندازه دیتاست‌ها و منابع محاسباتی موجود دارد.

پاسخ ۳ . تشخیص تقلب (کلاه برداری)

پاسخ ۳.۱ .آشنایی با دیتاست

در زیر دیتاست را لود کرده ایم . فیچر ها (ستون های) این دیتاست به دلایل حفاظت از اطلاعات اصلی از یک PCA رد شده اند.

کلاس در صورت ۱ بودن به معنای Fraudilous بودن و ۰ به معنای معتبر بودن است .

```
1 import pandas
```
0.2s

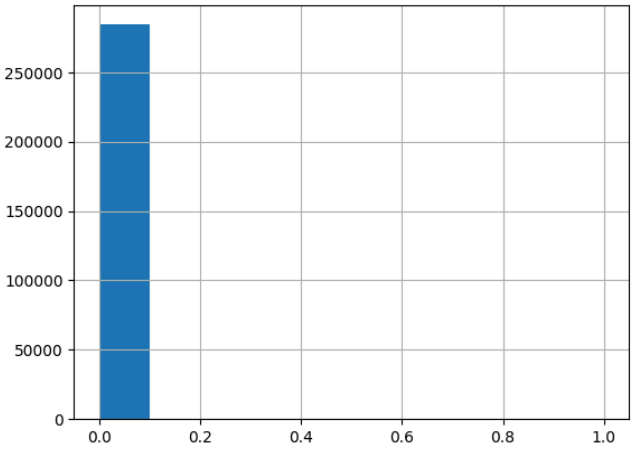
```
1 dataset = pandas.read_csv("creditcard.csv")
```
1.1s

```
1 dataset
```
0.0s

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0.77	0
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	24.79	0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	67.88	0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533	10.00	0
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	217.00	0

284807 rows × 31 columns

دیتاست لود شده Fraud Detection



نمودار هیستگرام کلاس ها

```
1 dataset['Class'].value_counts()
```
0.0s

```
0    284315
1       492
Name: Class, dtype: int64
```

تعداد کلاس ها

۳-۱-۱ به نظر می رسد که تعداد داده ها از کلاس های متفاوت در این دادگان به شدت imbalanced باشد که می تواند چالش هایی برای هر مدل

یادگیری ماشینی که روی این دادگان آموزش می بیند ایجاد کند .

۳-۱-۲ واضحا اگر مدل با این داده ها آموزش ببیند دیتای تعداد بسیار زیادی مشتری سالم را دریافت خواهد کرد و می تواند با مشکلات زیر روبرو شود

- متریک های عملکرد مدل گمراه کننده خواهد بود , برای مثال accuracy نزدیک ۹۹ درصد خواهد بود وقتی که هدف ما تشخیص fraudulent بودن واقعی است .
- مدل به شدت روی مشتری های سالم bias خواهد شد چون بسیاری از دیتای دریافتی آن flag نخواهد شد
- ممکن است مدل نتواند characteristic هایی که این دو کلاس را از هم جدا می کند خوب تشخیص بدهد

پاسخ ۳.۲ .پیاده سازی معماری مقاله

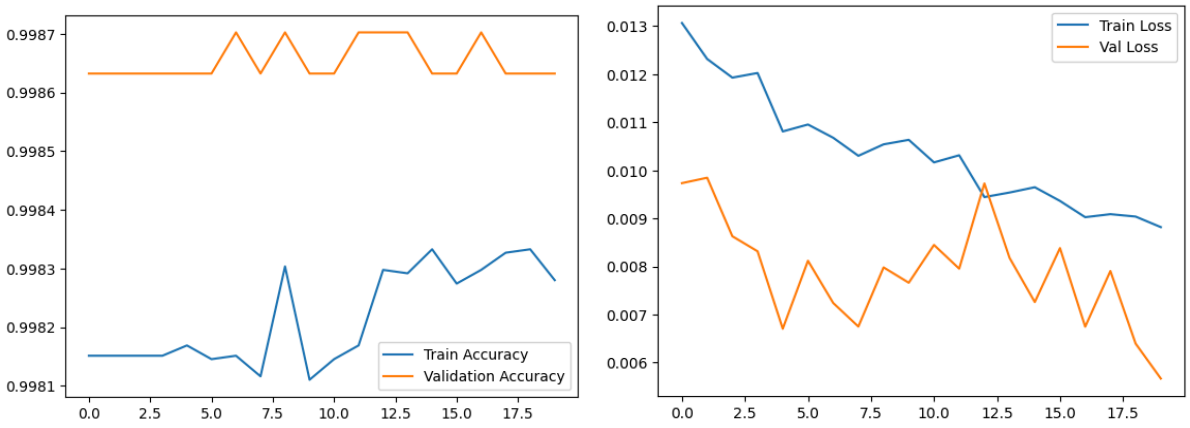
در این مرحله برای آموزش دادن بدون sample کردن از Optimizer = Adam با پارامتر های Default استفاده شده و همچنین برای Loss از Binary Cross Entropy With Logits استفاده شده است. در واقع Learning Rate = 0.001 ست شده برای ارزیابی از split های ۰.۲ روی کل

دیتاست برای تولید دادگان تست و ترین و همچنین ترین دوباره به ۲ قسمت با نسبت ۰.۲۵ برای ارزیابی و آموزش اصلی تبدیل شده است.

۱. با توجه به نمودار های زیر می توانیم ببینیم که مدل به شدت بد عمل میکند. دلیل این موضوع این است که تعداد sample هایی که کلاس non fraudulent را دارند در داده های داده شده به مدل بسیار زیاد است . در مدل هایی با خروجی ای شبیه نمودار های پایین لفظ overfitting غلط است چون وقتی به کار می رود که مدل شروع به یادگیری دادگان آموزش از جمله noise ها و پترن های خاص غیر general آن می کند .

در این حالات می گوییم Class Inbalance باعث Under fitting شده است و باعث شده است که مدل نتواند به اندازه کافی داده درباره کلاس های اقلیت داشته باشد تا آن ها را یادبگیرد و به شدت به سمت یک کلاس بایاس شده است .

.۲



3.1.Fraud with imbalanced labels

.۳

خیر . معیار F1 دقیقا به همین منظور تعبیه شده است . همچنین بهتر است از Recall و Precision نیز جداگانه استفاده کنیم . ولی Accuracy به دلیل اینکه نسبت به تعداد و توزیع دادگان ورودی در زمینه کلاس های پیش بینی agnostic نیست نمی تواند معیار خوبی باشد .

پاسخ ۳.۳ .نمونه برداری

۱) این روش بسیار شبیه SMOTE است در SMOTE با استفاده از K-Means برای تولید داده جدید شروع به تولید داده ها از Minority class میکنیم با استفاده از این روش که به اندازه K همسایه پیدا کرده k نقطه جدید بین همسایه و نقطه اصلی تولید میکنیم.

این روش از بعضی معایب برخوردار است مانند اینکه ممکن است داده Noisy اضافه کند چون اساسا اضافه کردن داده ها می تواند هماهنگ با توزیع واقعی کلاس ها نباشد همچنین می تواند در Boundry ها باعث به وجود آمدن داده هایی شود که قابل تمیز دادن خطی از یکدیگر نیستند چون توزیع لبه ها را نمیداند. در AdaSync بسیار شبیه SMOTE عمل می کنیم با این تفاوت که یک Difficulty metric وجود دارد که با استفاده از ان به بعضی داده هایی که سختی یادگیری بیشتری دارد وزن بیشتری برای تولید داده می دهیم و تعداد نمونه بیشتری ازین کلاس ها تولید میکنیم.

روش تشخیص Difficulty یک Sample به شکل زیر است :

اساسا به داده هایی که اطراف آن ها Majority class های زیادی وجود دارند وزن تولید بیشتری می دهد چون داده هایی

هستند که مدل سخت تر می تواند خود را Stretch کند تا این نمونه ها را یادبگیرد . همچنین با توجه به یک شعاع خاص در اطراف Minority Sample ها در Feature Space این همسایه ها را می شمارد (تعداد و فاصله) سپس ازین Difficulty استفاده کرده تا برای داده هایی که سخت تر قابل تمییز هستند چون با Majority class در هم تنیده هستند نمونه های بیشتری تولید کنند(اساسا مسیله Decision Boundry Noise Addition را با این تکنیک حل می کند)

مزایای

به طور adaptive برای نمونه های خاص با سختی خاص داده تولید می کند و با همه داده ها یکسان رفتار نمی کند (بر عکس SMOTE) همچنین این موضوع باعث می شود که مشکل Decision Boundry Overlapping کلاس ها که در SMOTE داشت کمتر اتفاق بیفتد چون هوشمندانه تر و در فضاهایی که Boundry پیچیده تری دارند بیشتر دیتا تولید می کند. همچنین این روش برای Highly Imbalanced datasets معمولا بهتر عمل می کند (مانند مثال سوال ما که به شدت Imbalanced است ولی دقت خوبی گرفتیم)

معایب

می تواند مدل را پیچیده کرده و باعث Over fitting شود

اگر داده هایی که سخت تر قابل یادگیری باشند از جنس Noise باشند با تولید کردن داده های سخت تر مانند همین داده ها هم باعث Over fitting می شود وهم باعث این می شود که مدل گیج شود چون داده های سخت تر بیشتری را خواهد دید و بدتر یاد خواهد گرفت. (Convergence) پایین تر در بعضی موارد . هر دو روش نمونه برداری Computationally Intensive هستند و می توانند بیشتر از روش های سنتی دیگر منابع محاسباتی مصرف کنند.

۲)

۱. در fit ابتدا مدل شروع به تنظیم کردن و ست کردن پارامتر های خود می کند که شمال تعداد کلاس ها و unique کلاس ها و همچنین تبدیل برچسب ها به یکدیگر است .

۲. در Generate Samples با استفاده از داده های تولید شده توسط K means به ازای یک سری نمونه ورودی تصمیم میگیریم که نمونه های جدید تولید کنیم سپس نگاه میکنیم که ایا در همسایه های این نمونه ها Minority وجود دارد یا نه اگر وجود نداشته باشد متوقف می شویم.

۳.در تابع Over sample اصل کار adaptive کردن Sampling انجام می شود به این شکل که به ازای هر نمونه به همسایه های آن در K MEANS نگاه کرده و با توجه به معیار هایی که بالا گفتیم (تعداد همسایه های Majority) یک Difficulty تعریف کرده و با استفاده از GenerateSamples به ازای این نمونه نمونه های دیگر تولید می کند در ادامه این داده ها را به Feature های اصلی اضافه کرده و این کار را مرتب تکرار می کند .

توضیح Config ها

مثلا ۱ : ratio

بعد از oversampling باید به اندازه majority class minority class oversampled داشته باشیم

imb thresholdدر واقع آستانه ای است که تصمیم گیرنده این است که یک کلاس نا موزون است

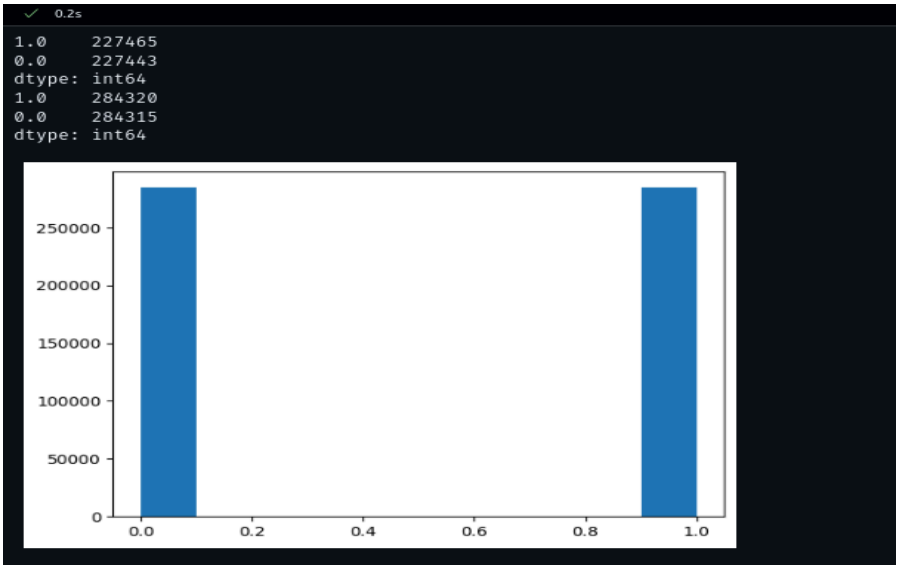
نمونه ها با نسبت حضور کمتر از imb threshold اقلیت حساب شده و به ازای آن ها over sampling انجام می شود

k در واقع هایپر پارامتر الگوریتم k means ما است

۳. در هیچ task یادگیری ماشین نباید اطلاعاتی به دادگان اضافه کنیم که این اطلاعات را از دیدن قسمت تست دادگان خود دریافت کرده ایم چون ممکن است مدل توزیع دادگان تست را بتواند با نگاه کردن به این اطلاعات یادبگیرد. اگر Adasyn روی کل دیتاست ران شود می تواند نمونه هایی تولید کند که حاصل از دیدن نمونه های اصلی دادگان تست است این به این معنی است که مدل ما از طریق یک واسطه یا proxy توانسته است test را ببیند و روی آن آموزش پیدا کند پس حتما باید oversampling فقط روی دادگان آموزش و بدون نگاه کردن به داده های test انجام شود و در غیر این صورت نوعی تقلب در task حساب می شود.

۴. نتیجه ران کردن Resampler ما با config پایین که نتیجه خوبی گرفته است

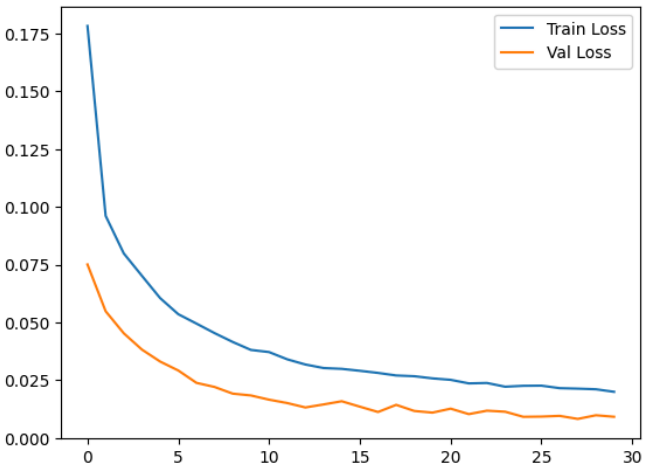
K	ratio	imb threshold
۵۰۰۰	1	0.6

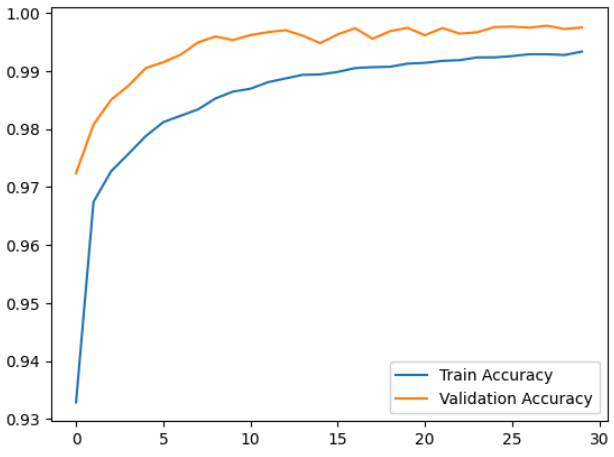


تعداد نمونه ها balance شده

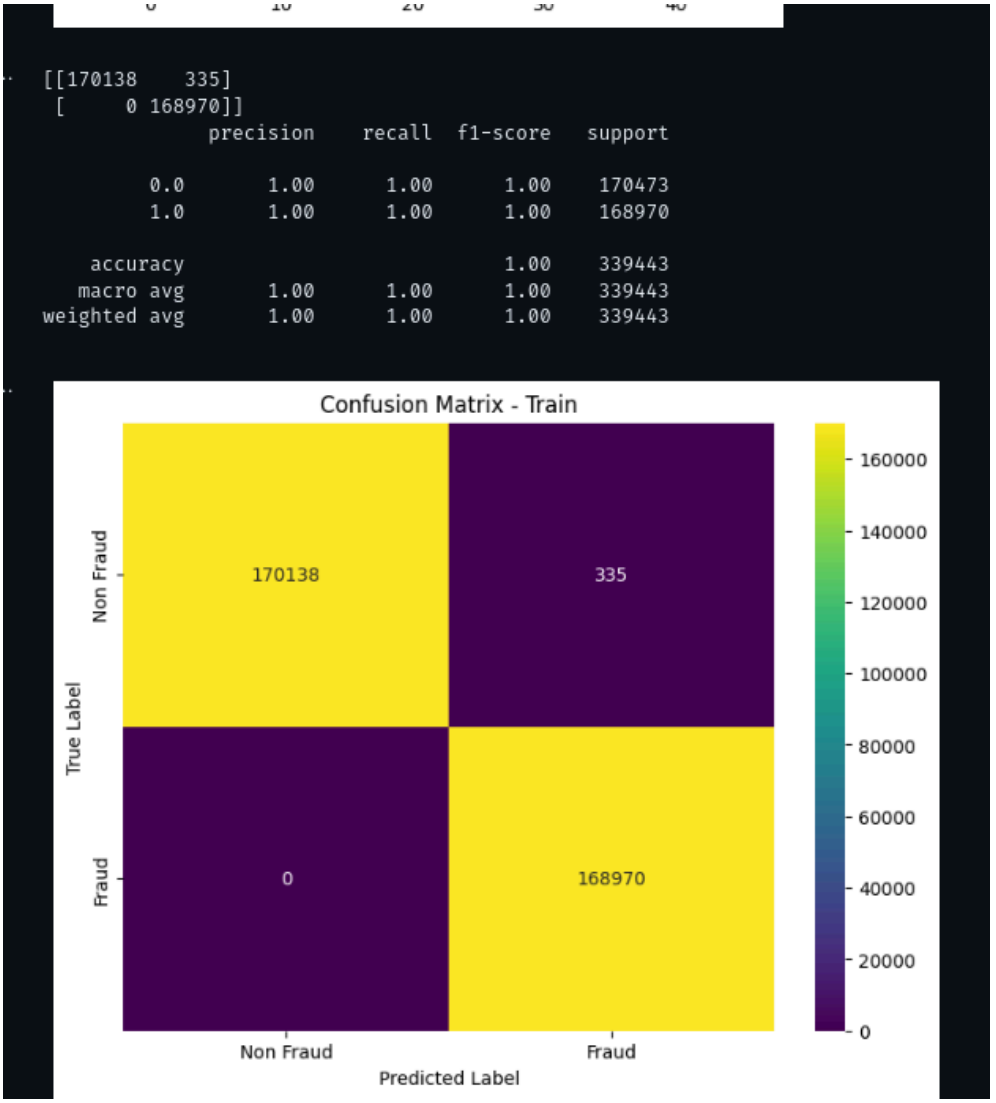
پاسخ ۴.۳ .آموزش بعد از نمونه برداری

نمونه برداری تاثیر چشم گیری در عملکرد مدل دارد به شکلی که loss ای که گیر کرده بود حالا هم در Validation و هم در Train کمتر می شود وهم روند است. لازم به ذکر است که در مقاله اصلی داده های تست هم از Oversampler رد می شوند و روی ان ها همه این متریک ها گزارش می شود که قابل نقد است چون احتمالا باعث تقلب در task می شود ولی ما برای نزدیک بودن به مقاله همین کار را انجام داده ایم



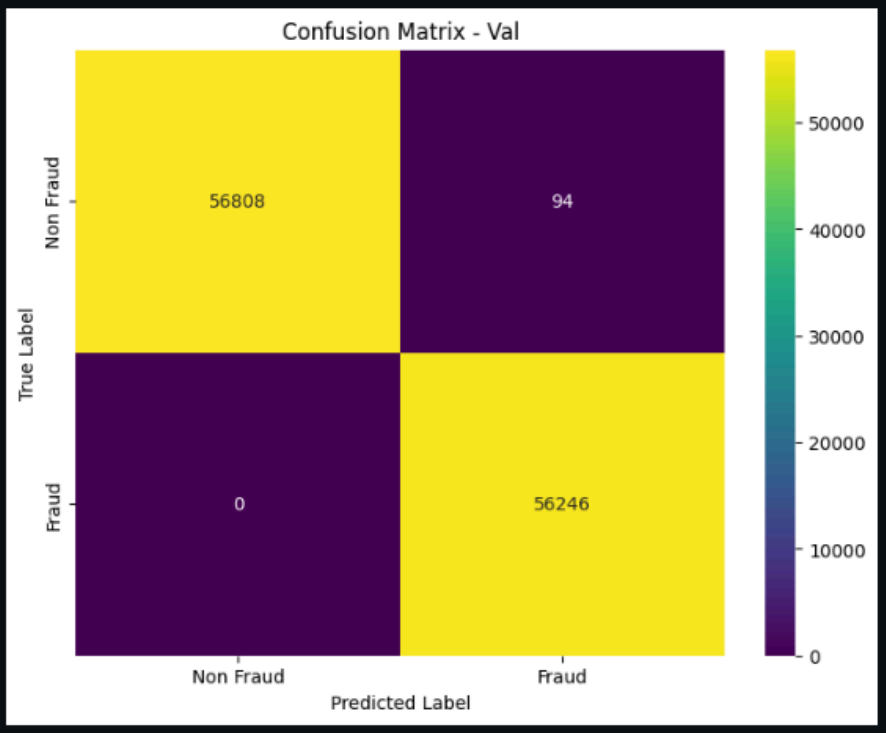


نمودار های دقت مدل روی آموزش و ارزیابی و Loss



Train Confusion Matrix

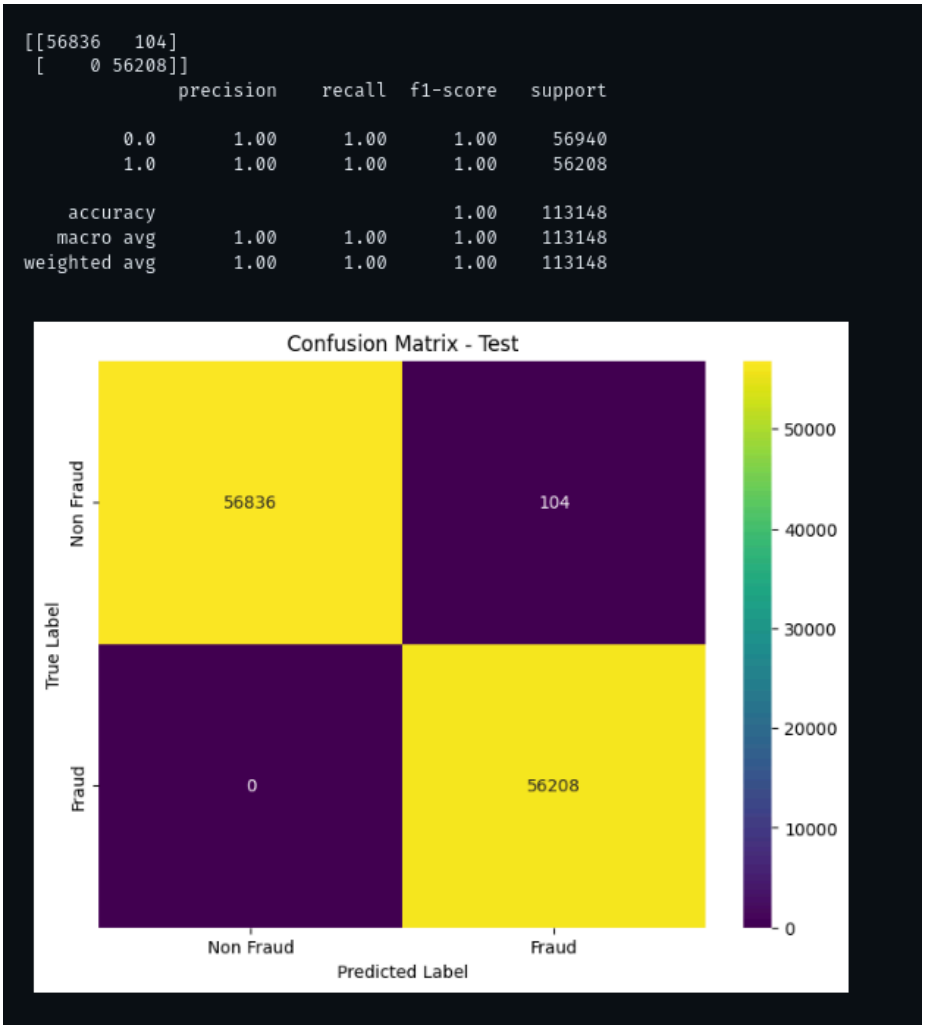
```
[[56808    94]
 [    0 56246]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     56902
         1.0       1.00      1.00      1.00     56246

    accuracy                           1.00    113148
   macro avg       1.00      1.00      1.00    113148
weighted avg       1.00      1.00      1.00    113148
```



Val Confusion Matrix

با توجه به نتایج به طور چشم گیری بهتر عملکرده که قابل انتظار بود همچنین حوزه اشتباه های مدل هم تغییر کرده قبلا بعضی ها را Fraud تشخیص می داد وقتی که نبودند و همه Fraud هارا اشتباه می گفت و حتی در فاز آموزش هم نمی توانست دقت خوبی به دست بیاورد چون Underfit می کرد با توجه به نتیجه های جدید می توانیم بگوییم که مدل توانسته Generalize کند و تا مقدار خوبی در Task به Performance مورد نیاز برسد .

بعضی از داده ها نیز Non Fradulent خوانده شده اند در حالی که Fraudulant بوده اند که به نظر طبیعی می آید.



Test Confusion Matrix