به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین امتیازی

نام و نام خانوادگی	فاطمه جلیلی – سالار صفردوست
شماره دانشجویی	۸۱۰۱۹۹۳۹۸ -۸۱۰۱۹۹۴۵۰
تاریخ ارسال گزارش	۱۴۰۲/۱۰/۲۷

فهرست

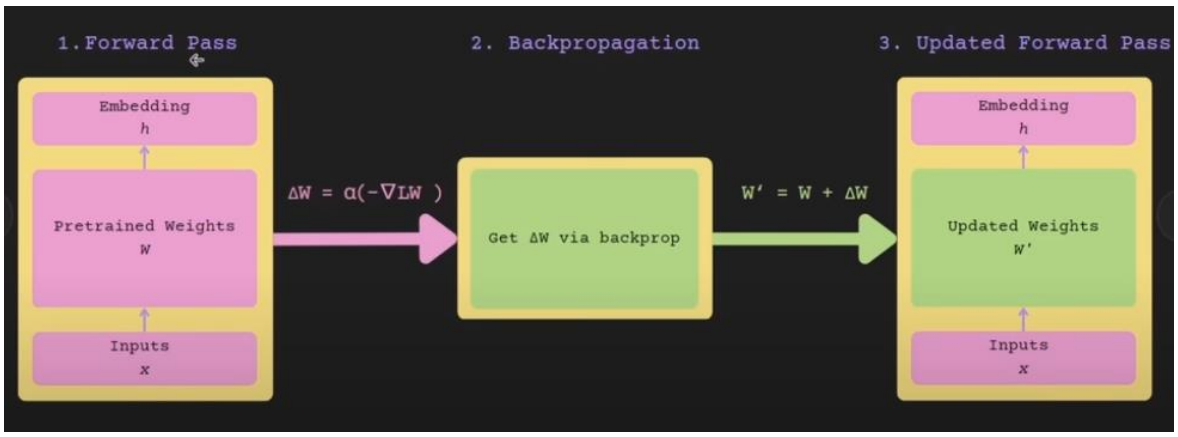
پاسخ ۲- LoRA

۲-۱. LoRA چگونه عمل می کند؟

Fine-tuning در آموزش مدل‌های Deep Learning به مرحله‌ای اشاره دارد که پارامترهای یک مدل پیش‌آموزش دیده را با استفاده از داده‌های جدید به‌روزرسانی می‌کنیم. این فرایند عموماً زمانی استفاده می شود که مدل پیش‌آموزش دیده، برای مسئله‌ای مشابه مورد استفاده قرار می‌گیرد.

Fine tune کردن کل پارامتر های مدل :

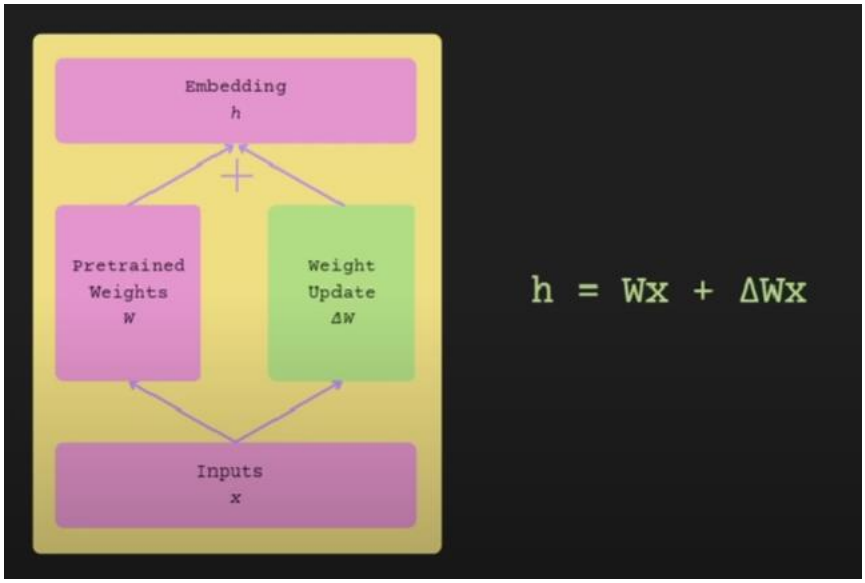
در این حالت، مدل پیش‌آموزش دیده را به عنوان یک مدل شبکه عصبی معمولی در نظر می‌گیریم و تمام پارامترها را با استفاده از داده جدید آموزش می‌دهیم، فرآیند update پارامتر ها در فاز backpropagation انجام می شود و forward pass ، update می شود.



این فرایند معمولاً زمان‌بر و از نظر محاسباتی گران است، زیرا تعداد زیادی پارامتر در مدل وجود دارد و همه آنها باید به‌روز شوند. اگر داده‌های آموزش **کافی** باشند، می‌توان انتظار داشت که کل پارامترها به داده‌های جدید سازگار شوند و عملکرد بهبود پیدا کند.

Fine tune کردن برخی از لایه های مدل :

به جای به‌روزرسانی تمام پارامترها، ما فقط پارامتر های لایه‌های مشخصی را با داده‌های جدید آموزش می‌دهیم. این روش معمولاً زمان کم تری می گیرد و محاسبات کم تری نیاز دارد، زیرا تنها بخشی از مدل به‌روزرسانی می‌شود و دیگر پارامتر های مربوط به لایه های فریز شده نیاز به به روزرسانی در فرآیند آموزش مدل در هر ایپاک ندارند.

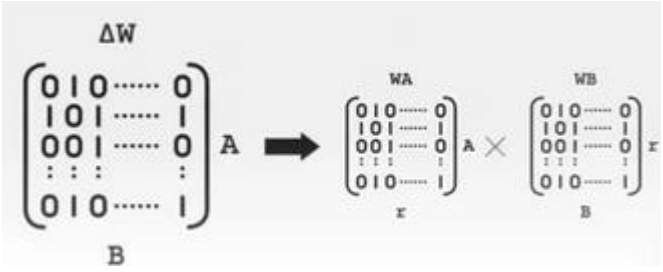


علاوه بر این، اگر داده های کافی برای آموزش یک مدل سنگین از ابتدا را نداشته باشیم با fine tune کردن کل پارامتر ها ممکن است با overfit مواجه شویم در این حالت به وزن های از پیش تعیین شده برای مدل که روی دیتاست بزرگ از پیش آموزش دیده اند برای لایه های ابتدایی دست نمی زنیم که وظیفه استخراج ویژگی های کلی از داده دارند و فقط لایه های انتهایی مدل که ویژگی های جزیی تر را فرا می گیرند آموزش می دهیم تا مدل به خوبی روی دیتاست جدید ما که ممکن است توزیع متفاوتی از دیتاهایی که مدل قبلا روی آن ترین شده است داشته باشد، ترین شود.

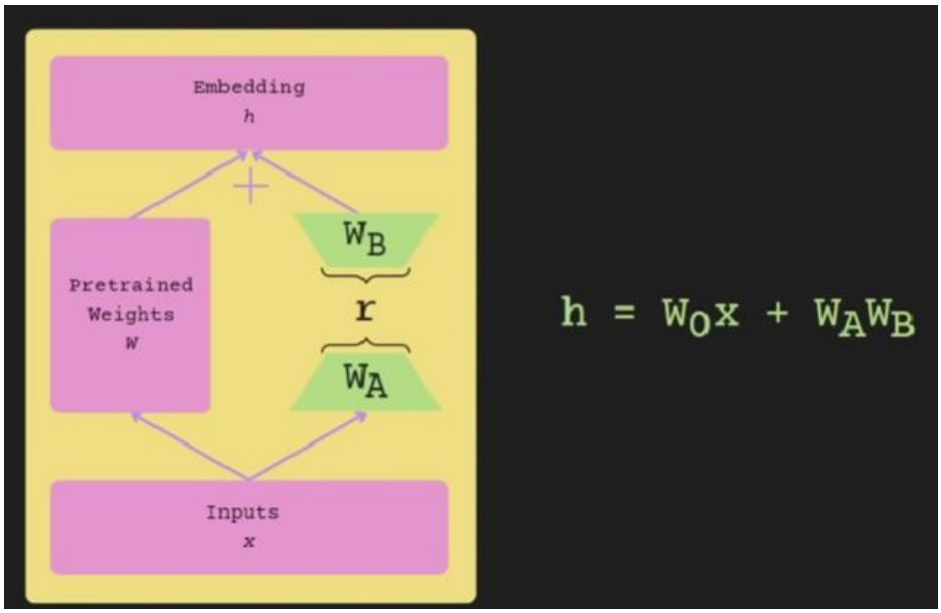
روش LoRA :

فرض کنیم ماتریس وزن های مدل یک ماتریس $A \times B$ باشد ، بنابراین ماتریس تغییرات وزن ها هم ابعادی مشابه خواهد داشت. ایده اصلی LoRA این است که تعداد ماتریس وزن های زیادی از مدل‌های از پیش آموزش‌دیده شده، ابعاد ذاتی(intrinsic dimension) نسبتاً کمی دارند، به این معنی که ابعاد واقعی ماتریس که شامل سطر ها و ستون هایی است که به یکدیگر وابستگی ندارند و مستقل هستند بسیار کم تر از A, B است.

با استفاده از این ایده decomposition روی ماتریس تغییرات وزن ها ΔW انجام می شود و به فرم زیر نوشته می شود:



که r عدد بسیار کوچک تری از ابعاد ماتریس است ، حال فرآیند fine tune به شکل زیر در می آید:



W_A, W_B ویژگی های اصلی ΔW را حفظ می کنند با این تفاوت که پارامتر های بسیار کم تری نیاز به Fine tune و update شدن در هر دوره دارند.

برای مثال فرض کنید $A = B = 1000$ باشد و $r = 4$ ، در حالتی که بخواهیم همه پارامتر های مدل را Fine tune کنیم نیاز به update 10^6 پارامتر در هر دوره داریم ولی با استفاده از LoRA این مقدار به $2 \times 1000 \times 4 = 8000$ کاهش می یابد که به طور شگفت انگیزی کم تر است لذا فرآیند Fine tune با سرعت بیش تری انجام می گیرد و نیاز به فضای کم تری برای ذخیره کردن checkpoint ها دارد.

۲-۲. قرار است چه کاری را روی چه داده هایی انجام دهیم؟

دیتاست گروه ما دیتاست QQP است.

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

این دیتاست شامل دو ستون برای دو سوال است که ستون آخر مشخص کننده این است که دو سوال یکسان هستند یا نه.

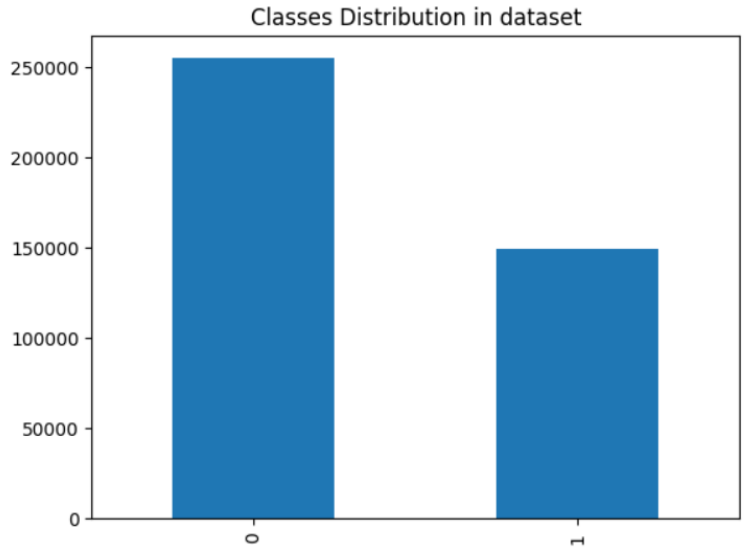
همچنین ۳ ستون اضافه برای index ردیف ها و سوالات اختصاص داده شده بود که چون در ادامه نیازی به آن ها نداشتم آن ها را حذف کردم.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 3 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   question1     404289 non-null  object
 1   question2     404288 non-null  object
 2   is_duplicate  404290 non-null  int64 
dtypes: int64(1), object(2)
memory usage: 9.3+ MB
```

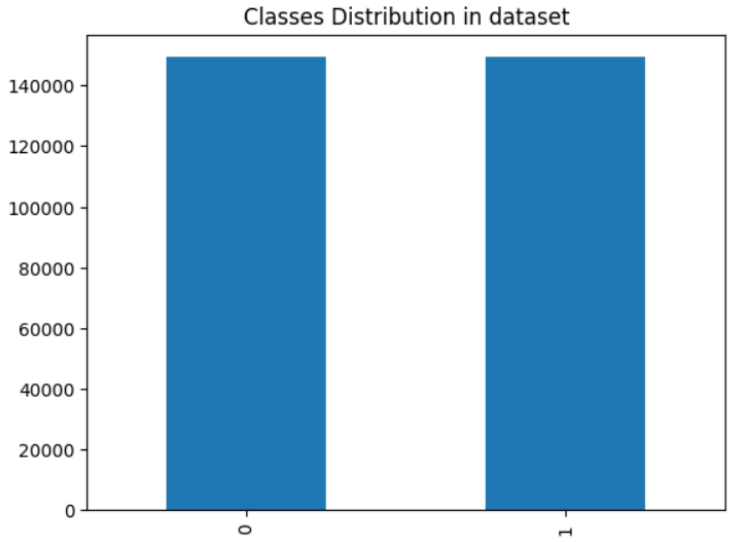
با مقایسه تعداد ردیف ها و تعداد non-null متوجه شدیم تعدادی ستون با داده null در دیتا ست وجود دارد لذا آن ها حذف کردم:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 404287 entries, 0 to 404289
Data columns (total 3 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   question1     404287 non-null  object
 1   question2     404287 non-null  object
 2   is_duplicate  404287 non-null  int64 
dtypes: int64(1), object(2)
memory usage: 12.3+ MB
```

در ادامه هم توزیع دیتاست برای جملات یکسان و متفاوت را رسم کردم:



همانطور که دیده می شود دیتاست دارای بایاس است ، همچنین در ادامه آموزش مدل بسیار زمان بر است و مجبوریم بخش بسیار کمی از دیتا را در نظر بگیریم لذا همین جا تعدادی از ۰ ها را به طور رندوم حذف کردم تا تعداد ۰ و ۱ ها برابر شوند:



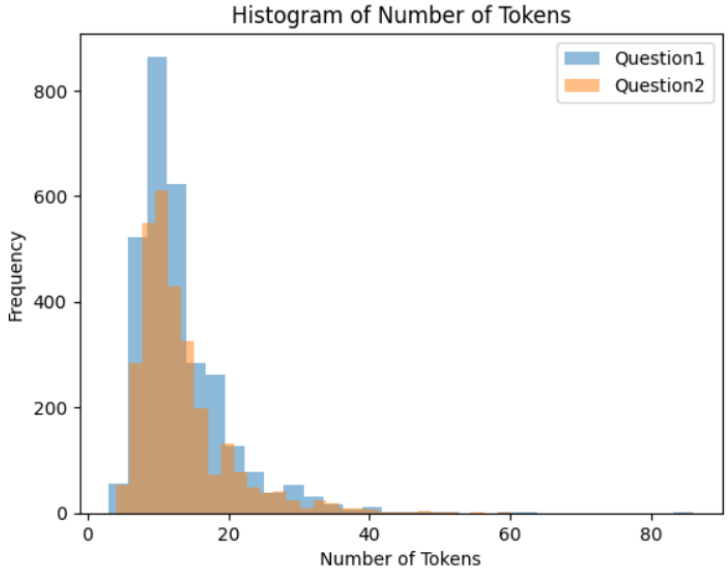
ورودی مدل های خانواده BERT به فرم زیر هستند:

- single sequence: [CLS] X [SEP]
- pair of sequences: [CLS] A [SEP] B [SEP]

که در این جا چون دو جمله داریم از حالت دوم استفاده می کنیم.

A و B جملات توکنایز شده هستند، برای توکنایز کردن دیتا از توکنایزر خود مدل استفاده می کنیم منتها پیش از توکنایز کردن برای اینکه جملات همگی یک اندازه داشته باشند (برای padding) نیاز داریم تا ماکسیمم تعداد توکن جملات را بدانیم، برای این منظور هیستوگرام تعداد توکن های سوال ۱و ۲

```
Maximum number of tokens in question1s: 86
Maximum number of tokens in question2s: 60
```



و ماکسیمم تعداد توکن هر کدام را بدست می آوریم و هنگام اعمال padding ، max_length را برابر مجموع ماکسیمم ها و ۱۰ تا بیش تر قرار می دهیم.

```
def batchTokenize(sample):
    return tokenizer(sample["question1"], sample["question2"],  padding='max_length', truncation=True, max_length = max_tokens_q1 + max_tokens_q2 + 10)

train_dataset = Dataset.from_pandas(train_df)
test_dataset = Dataset.from_pandas(test_df)

train_dataset = train_dataset.map(batchTokenize, batched=True)
test_dataset = test_dataset.map(batchTokenize, batched=True)
```

رای تبدیل جملات توکنایز شده به فرمت ورودی مدل که بالاتر توضیح دادم مطابق کد فوق از کتابخانه ی dataset استفاده می کنیم.

خروجی مدل آمده در صورت پروژه به صوری درصدی است که جملات ورودی به چه احتمالی به کدام کلاس تعلق دارند برای مثال ۹۰٪ کلاس ۰ و ۱۰٪ کلاس ۱ از آنجایی که ما فقط شماره کلاس مربوطه را می خواهیم نیاز به یک classification head در بالای مدل داریم، چنین مدلی را با می توانیم با `AutoModelForSequenceClassification` وارد کنیم و به صورت مجزا در انتها دستی درصد های خروجی را به کلاس تبدیل نکنیم.

می توانیم LoRA را روی هر لایه دلخواهی اعمال کنیم منتها مطابق مقاله طبق توضیحات سوال ۱ روی attention layer ها تمرکز می کنیم و بقیه ماژول های MLP را فریز می کنیم. توضیحات بیش تر راجب پارامتر های لورا که مطابق مقاله تعیین کردیم در ادامه آمده است.

۲-۳. و بالاخره کد نویسی : آموزش مدل

Fine tune کردن کل پارامتر های مدل :

متاسفانه به دلیل بزرگ بودن بودن و دیتاست با ۲۰ ایپاک انجام شده در مقاله و بقیه پارامتر های مشابه مقاله آموزش مدل حدود ۶۵۸ ساعت زمان لازم داشت:

برای رسیدن به زمانی معقول تعداد ایپاک ها را برابر ۱۰ قرار دادم و روی ۰.۰۱ دیتا مدل را آموزش دادم ، با توجه به اینکه دیتاست به اندازه کافی بزرگ بود و با توجه به درصد های بدست آمده مشکلی از لحاظ overfit نداشتم.

Fine tune روی کل پارامتر های مدل:

معماری مدل:

```
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=2, bias=True)
    )
  )
```

تعداد پارامتر های مدل که همگی قرار است Fine tune شوند:

```
trainable_params1 = sum(p.numel() for p in model.parameters() if p.requires_grad)
print("Number of trainable parameters in initial model:", trainable_params1)
```

```
Number of trainable parameters in initial model: 355361794
```

```
trainer_args = TrainingArguments(
    output_dir                  = "./result1",
    overwrite_output_dir        = True,
    per_device_train_batch_size = 16,
    per_device_eval_batch_size  = 16,
    gradient_accumulation_steps = 1,
    learning_rate               = 1e-06,
    adam_epsilon                = 1e-08,
    logging_steps               = 100,
    num_train_epochs            = 10.0,
    save_strategy               = "epoch",
    evaluation_strategy         = "epoch",
    load_best_model_at_end      = True,
    metric_for_best_model       = "accuracy",
    report_to = 'wandb',
    )
```

مطابق مقاله batch_size را برابر ۱۶ قرار می دهیم ، learning rate پیشنهاد شده مقاله به درستی برای من عمل نمی کرد و دقت های بدست آمده در ایپاک ها فقط کمی بالا پایین می شد و تغییری نداشت لذا لرنینگ ریت تا مجدد تنظیم کردم و متوجه شدم 1e-6 به درستی عمل می کند.

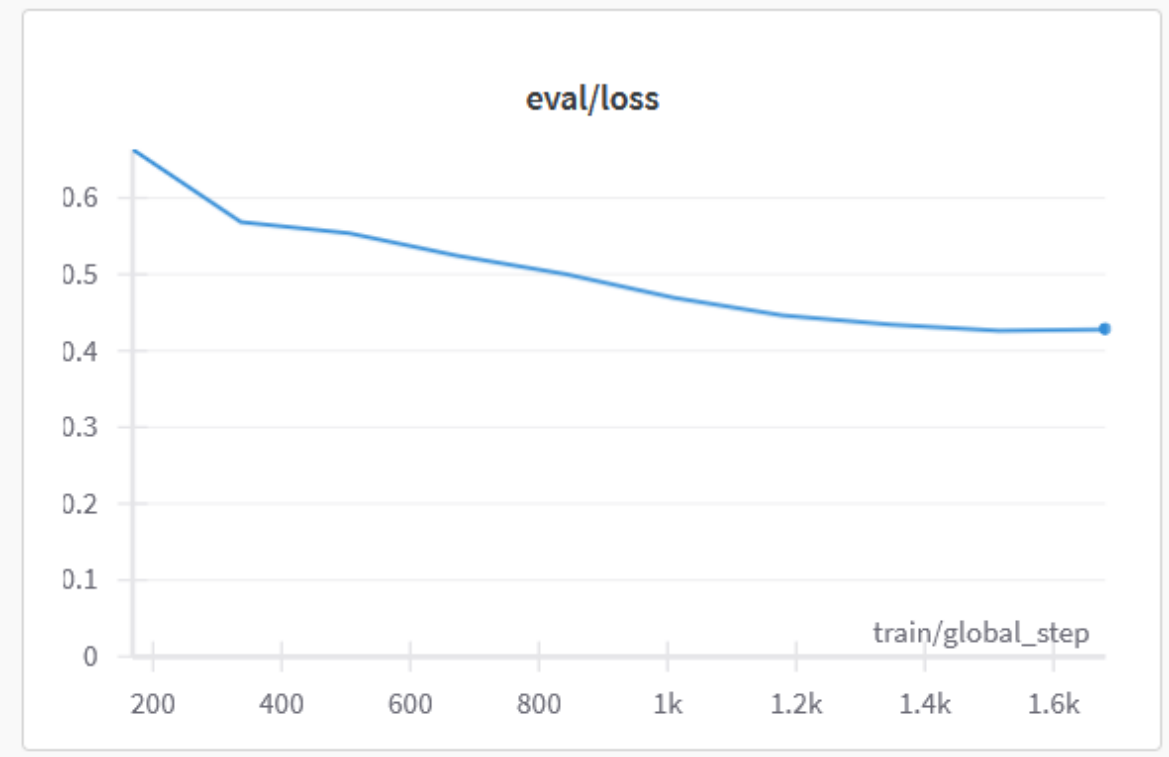
- برای رسم نمودار ها از ابزار wandb استفاده کردم که تمام نمودار های مربوطه را همزمان با ترین شدن رسم می کند ، عکس همه نتایج در گزارش هم آمده است ، برای دیدن پلات ها در کد آپلود شده نیاز به دسترسی دارین ، من لینک دعوت را به ایمیل شما پیش تر ارسال کرده ام با وارد شدن از طریق آن لینک می توانید پلات ها را در نوت بوک در زیر قسمت ترین و در سایت wandb که لینک مربوط به هر پلات زیر آن در نوت بوک آپلود شده تحت عنوان View run at قرار دارد مشاهده کنید. نام تیم که لینک دعوت آن برای شما ارسال شده است fatemeh_nndl است.

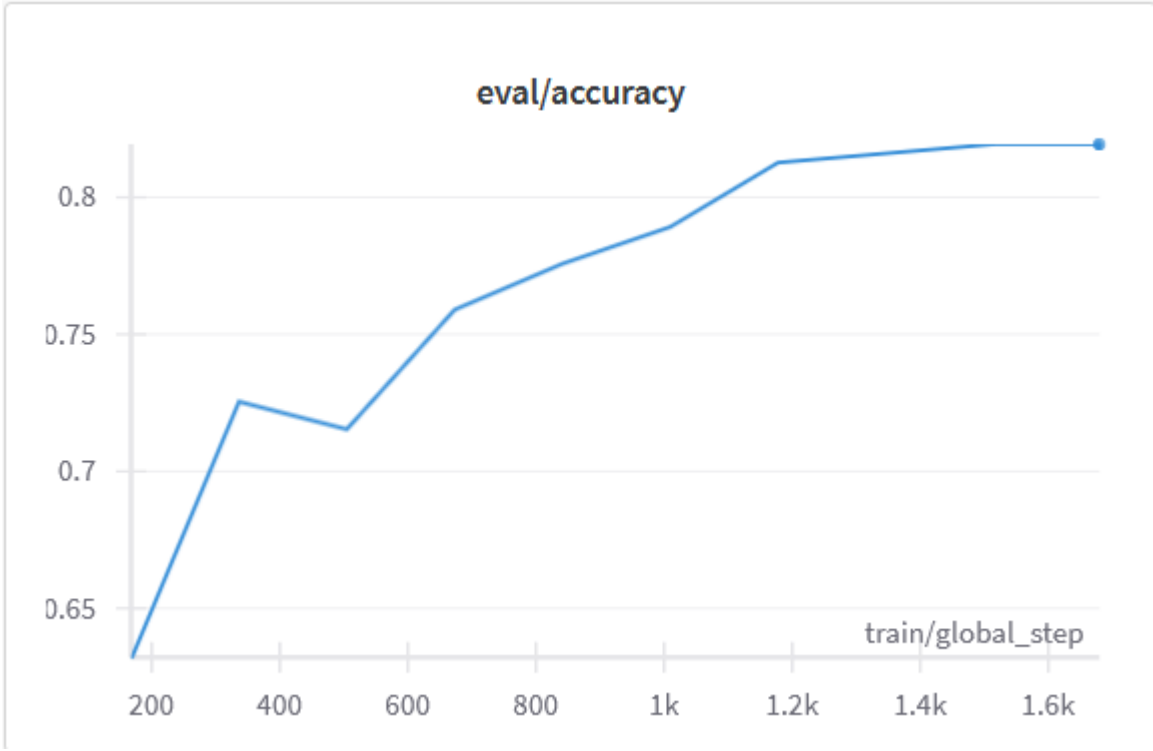
زمان صرف شده جهت آموزش مدل برای ۱۰ ایپاک و دقت های بدست آمده :

[1680/1680 40:30, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.697100	0.663496	0.632107
2	0.622700	0.568707	0.725753
3	0.581300	0.554128	0.715719
4	0.559900	0.524771	0.759197
5	0.534700	0.500775	0.775920
6	0.505900	0.469979	0.789298
7	0.492600	0.447043	0.812709
8	0.485100	0.435029	0.816054
9	0.467600	0.427066	0.819398
10	0.455700	0.428638	0.819398

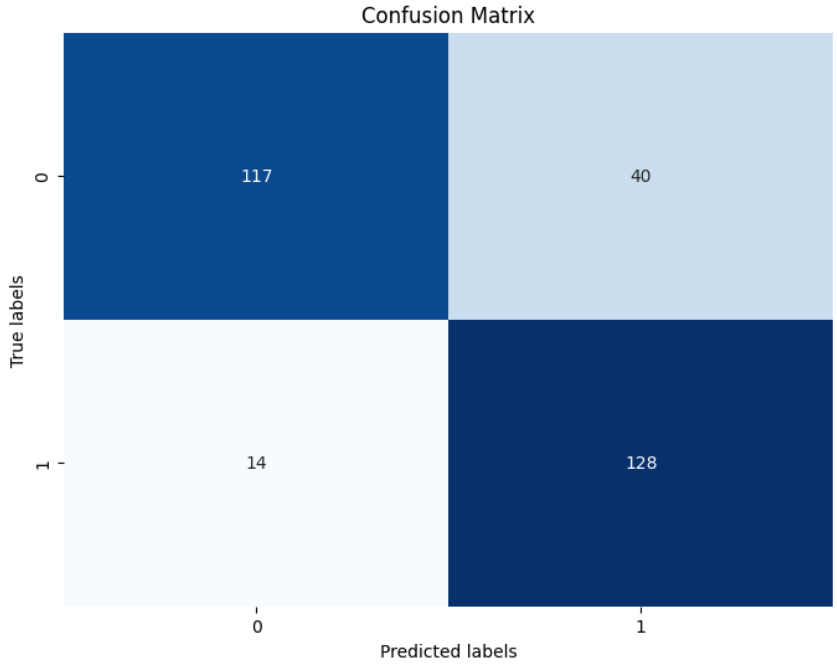
نمودار های خطا روی داده تست و ترین و نمودار دقت روی داده ترین :





دیگر نتایج بدست آمده روی داده تست:

	precision	recall	f1-score	support
0	0.89	0.75	0.81	157
1	0.76	0.90	0.83	142
accuracy			0.82	299
macro avg	0.83	0.82	0.82	299
weighted avg	0.83	0.82	0.82	299



استفاده از LoRA:

برای اعمال LoRA از کتابخانه peft و lora_config استفاده می کنیم:

```
lora_config = LoraConfig(
    r=8,
    lora_alpha=16,
    lora_dropout=0.05,
    target_modules=["query", "key","value"],
    bias='none',
    task_type=TaskType.SEQ_CLS
)
peft_model = get_peft_model(model_copy, lora_config)
```

طبق مقاله که نتایج روی مدل های مختلف را گزارش کرده بود پارامتر هایی که برای مدل Roberta-large انتخاب کرده بودند را قرار می دهیم یعنی r=0.8 و lora_alpha = 16 ، در مورد lora_dropout چیزی گفته نشده بود لذا این مقدار را خیلی کم قرار می دیم.

مطابق مقاله LoRA را روی attention layer ها تنها اعمال می کنیم لذا target_modules را هر سه مورد quary, key, value در لایه های ترسفورمری قرار می دهیم ، task_type هم مطابق تسکی که داریم یعنی sequence classification قرار می دهیم.

با ورودی دادن مدل اولیه و lora_config به get_peft_model مدلی که LoRA روی آن اعمال شده و تنها پارامتر های مربوط به آن trainable هستند و بقیه پارامتر ها فریز شده اند را خروجی می گیریم.

معماری مدل:

```
PeftModelForSequenceClassification(
  (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=0.05, inplace=False)
                    )
                    (lora_A): ModuleDict(
                      (default): Linear(in_features=1024, out_features=8, bias=False)
                    )
                    (lora_B): ModuleDict(
                      (default): Linear(in_features=8, out_features=1024, bias=False)
                    )
                    (lora_embedding_A): ParameterDict()
                    (lora_embedding_B): ParameterDict()
                  )
                  (key): lora.Linear(
                    (base_layer): Linear(in_features=1024, out_features=1024, bias=True)
                    (lora_dropout): ModuleDict(
                      (default): Dropout(p=0.05, inplace=False)
                    )
                    (lora_A): ModuleDict(
                      (default): Linear(in_features=1024, out_features=8, bias=False)
                    )
                    (lora_B): ModuleDict(
                      (default): Linear(in_features=8, out_features=1024, bias=False)
                    )
                    (lora_embedding_A): ParameterDict()
                    (lora_embedding_B): ParameterDict()
                  )
                  (value): lora.Linear(
                    (base_layer): Linear(in_features=1024, out_features=1024, bias=True)
                    (lora_dropout): ModuleDict(
                      (default): Dropout(p=0.05, inplace=False)
                    )
                    (lora_A): ModuleDict(
                      (default): Linear(in_features=1024, out_features=8, bias=False)
                    )
                    (lora_B): ModuleDict(
                      (default): Linear(in_features=8, out_features=1024, bias=False)
                    )
                    (lora_embedding_A): ParameterDict()
                    (lora_embedding_B): ParameterDict()
                  )
```

```
          (value): lora.Linear(
            (base_layer): Linear(in_features=1024, out_features=1024, bias=True)
            (lora_dropout): ModuleDict(
              (default): Dropout(p=0.05, inplace=False)
            )
            (lora_A): ModuleDict(
              (default): Linear(in_features=1024, out_features=8, bias=False)
            )
            (lora_B): ModuleDict(
              (default): Linear(in_features=8, 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)
      )
    )
  )
)
(classifier): ModulesToSaveWrapper(
  (original_module): 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=2, bias=True)
  )
  (modules_to_save): ModuleDict(
    (default): 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=2, bias=True)
    )
  )
)
)
)
)
```

همانطور که مشخص است در module هایی که مشخص کردیم یعنی query, key, value پس از لایه base_layer که پارامتر های آن اکنون فریز هستند و قبل از لایه dropout مربوط به مدل اصلی لایه های مربوط به لورا یعنی lora_a, lora_b, lora_dropout و لایه های embedding مربوط به A, B اضافه شده اند ولی بقیه معماری تغییری نکرده و ثابت است.

تعداد پارامتر های مدل که قرار است Fine tune شوند:

```
trainable_params2 = sum(p.numel() for p in peft_model.parameters() if p.requires_grad)
print("Number of trainable parameters in LoRA applied model:", trainable_params2)
print("percentage of trainable model parameters in comparison to initial model:", trainable_params2 * 100 / trainable_params1)
```

```
Number of trainable parameters in LoRA applied model: 2231298
percentage of trainable model parameters in comparison to initial model: 0.6278947364836862
```

مشاهده می شود که پارامتر های trainable حدود ۶۳٪ حالت اول هستند.

```
trainer_args = TrainingArguments(
    output_dir                  = "./result2",
    overwrite_output_dir        = True,
    per_device_train_batch_size = 16,
    per_device_eval_batch_size  = 16,
    gradient_accumulation_steps = 1,
    learning_rate               = 1e-04,
    adam_epsilon                = 1e-08,
    logging_steps               = 100,
    num_train_epochs            = 10.0,
    save_strategy               = "epoch",
    evaluation_strategy         = "epoch",
    load_best_model_at_end      = True,
    metric_for_best_model       = "accuracy",
    report_to = 'wandb',
    )
```

مطابق مقاله batch_size را برابر ۱۶ قرار می دهیم ، learning rate پیشنهاد شده مقاله به درستی برای من عمل نمی کرد و دقت های بدست آمده در ایپاک ها به خوبی پیشرفت نمی کرد و پس از مقدار پیشرفت فقط بالا پایین می شد لذا لرنینگ ریت تا مجدد تنظیم کردم و متوجه شدم 1e-4 به درستی عمل می کند.

زمان صرف شده جهت آموزش مدل برای ۱۰ ایپاک و دقت های بدست آمده :

View run at https://wandb.ai/fatemeh_rmdi/huggingface/runs/nowmxoju

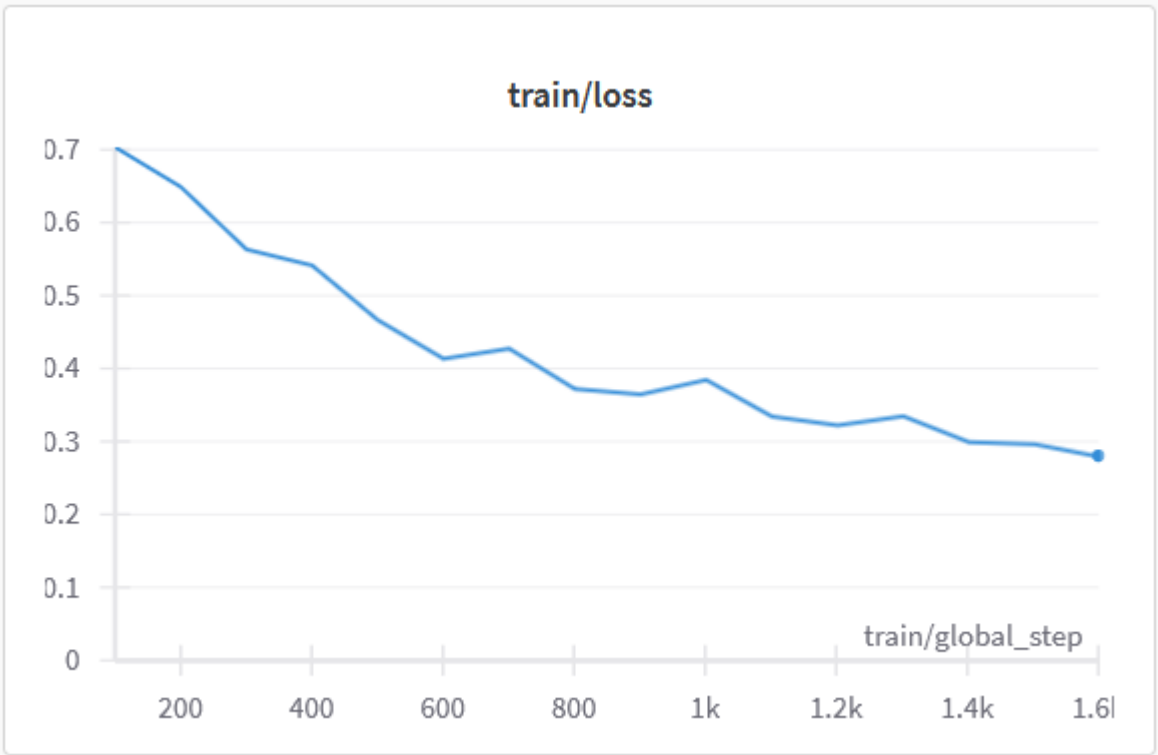
[1680/1680 27:36, Epoch 10/10]

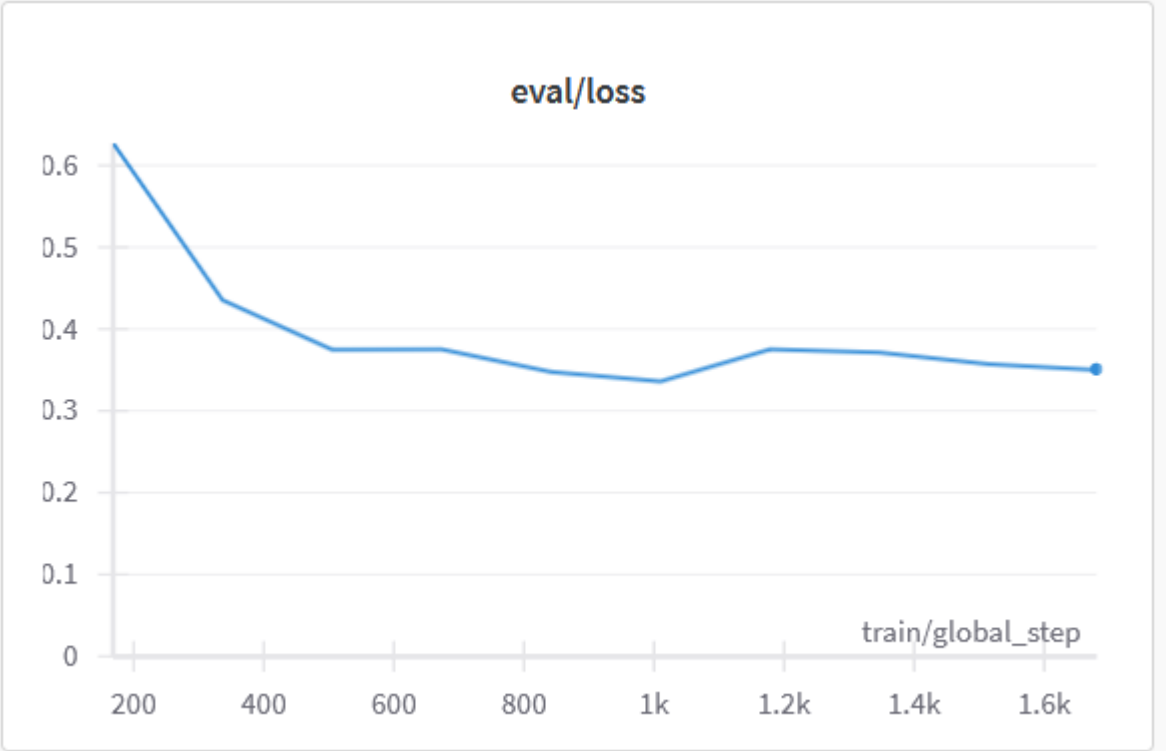
Epoch	Training Loss	Validation Loss	Accuracy
1	0.703400	0.627229	0.759197
2	0.563700	0.436011	0.799331
3	0.467200	0.375744	0.819398
4	0.414400	0.376062	0.842809
5	0.373100	0.348590	0.842809
6	0.385500	0.337056	0.862876
7	0.335500	0.376064	0.872910
8	0.335900	0.372193	0.846154
9	0.297600	0.358021	0.856187
10	0.280300	0.350857	0.879599

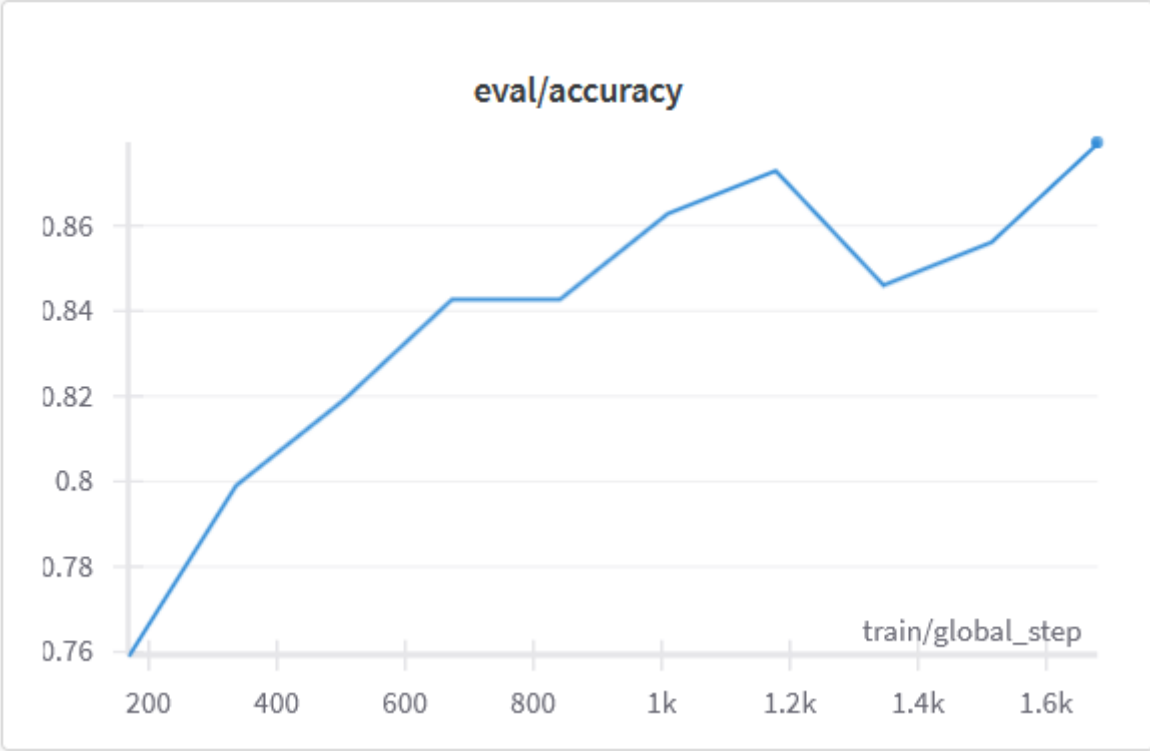
زمان صرف شده برای آموزش به ۲۸ دقیقه کاهش یافته که در مقایسه با حالت قبل که ۴۱ دقیقه بود ٪۳۲ بهتر شده است.

دقت هم به ٪۸۸ رسیده است که در مقایسه با ٪۸۲ حالت قبل پیشرفت قابل توجهی داشته است.

نمودار های خطا روی داده تست و ترین و نمودار دقت روی داده ترین :

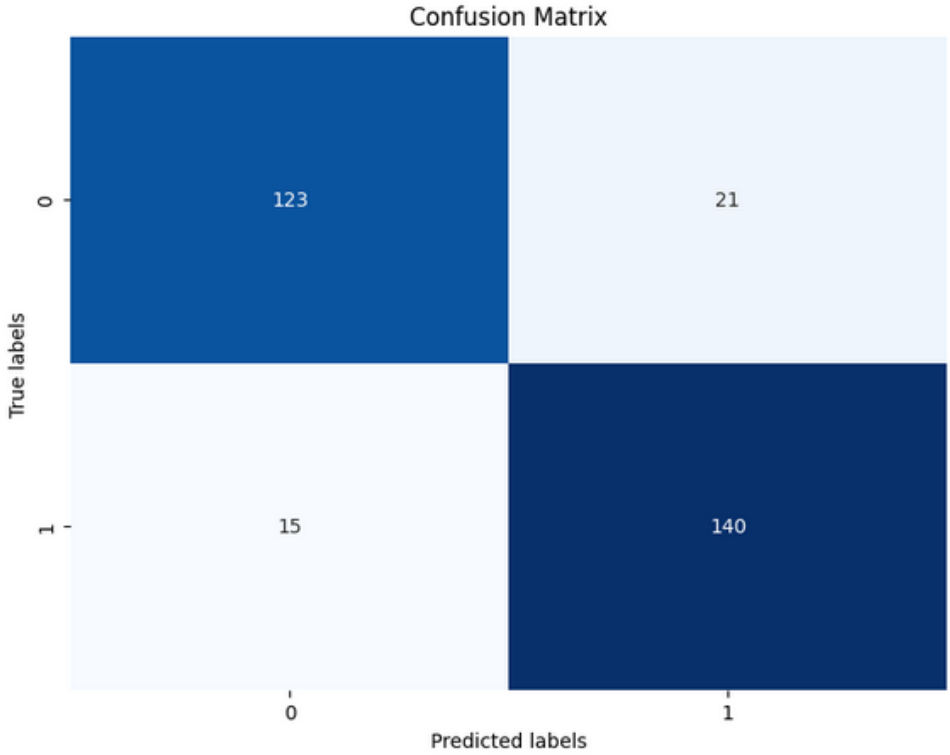



eval/loss
0.6
0.5
0.4
0.3
0.2
0.1
0
train/global_step
200
400
600
800
1k
1.2k
1.4k
1.6k


eval/accuracy
0.86
0.84
0.82
0.8
0.78
0.76
train/global_step
200
400
600
800
1k
1.2k
1.4k
1.6k

دیگر نتایج بدست آمده روی داده تست:

	precision	recall	f1-score	support
0	0.89	0.85	0.87	144
1	0.87	0.90	0.89	155
accuracy			0.88	299
macro avg	0.88	0.88	0.88	299
weighted avg	0.88	0.88	0.88	299



	Val accuracy	Val loss	F1 score	Number of trainable parameters	Time taken to train
Fine tune all parameters	0.82	0.43	0.81/0.83	355361794	41 min
LoRA	0.88	0.35	0.87/0.89	2231298	28 min

همانطور که مشاهده می کنید LoRA زمان کم تری برای آموزش نیاز دارد چرا که طبق توضیحات قبلی با matrix decomposition روی ماتریس تغییرات وزن ها تنها feature ها مستقل را در نظر می گیرد و

row های وابسته را حذف می کند لذا پارامتر های کم تری نیاز به update دارند و فرآیند سریع تر می شود.

در رابطه با دقت ، در اینجا ما دیتا ست کافی جهت آموزش کل پارامتر های مدل را داشتیم لذا آموزش کل پارامتر های چندان بد عمل نکرد ولی در صورتی که دیتا ناکافی بود با دقت بسیار کم تری روی داده تست مواجه می شدیم. باید در نظر گرفت ۶ درصد اختلاف دقت روی LoRA به خاطر سریع تر بودن مدل است، اگر زمان اجازه می داد و کل پارامتر های مدل را برای ایپاک های بیش تر و روی کل دیتاست ترین می کردیم قاعدتا به دلیل کافی بودن دیتاست باید به دقت بهتری در مقایسه با LoRA می رسیدیم چرا که بالاخره LoRA محدودیت هایی هم دارد و برخی feature های کم اهمیت را حذف کرده است منتها عموما چنین زمان و قدرت پردازشی در دسترس نیست و لذا LoRA کاربرد بسیار گسترده ای دارد.

۲-۴. چرا LoRA؟

داریم :

$$W = W_0 + BA$$

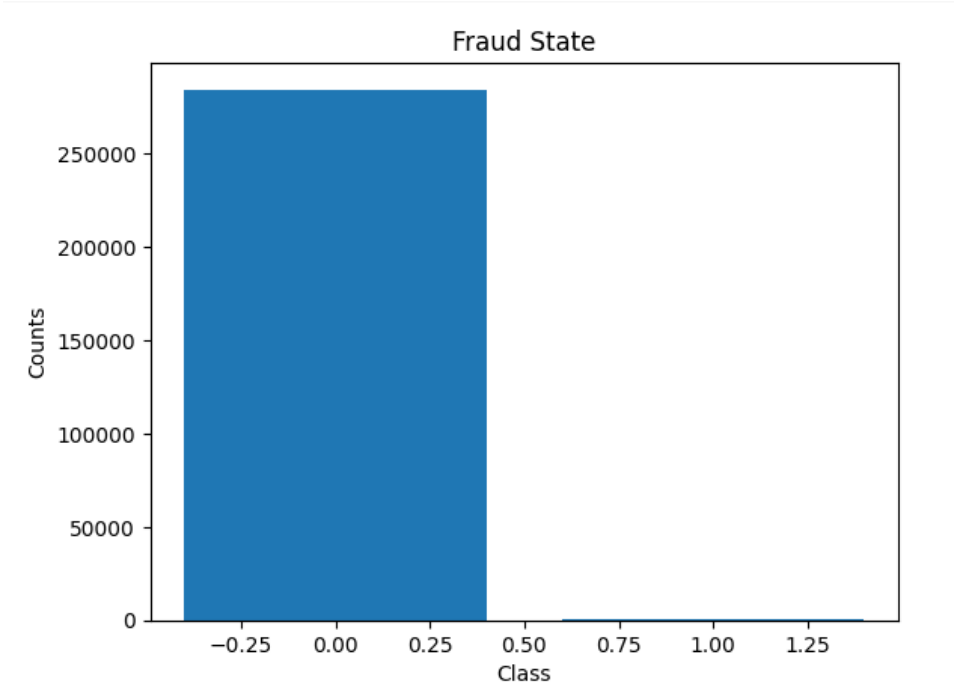
زمانی که بخواهیم به یک task دیگر سوییچ کنیم می توانیم W_0 را مطابق معادله فوق بدست آوریم و با جمع کردن با $B'A'$ جدید W جدید را با اشغال حافظه کم بدست آوریم.

اما زمانی که بخواهیم کل پارمتر های مدل را fine tune کنیم سوییچ کردن بین تسک ها کارآسانی نیست هر بار باید زمان بسیار زیادی برای آموزش مدل صرف کنیم و فضایی که برای ذخیره مدل جدید آموزش دیده نیاز داریم بسیار زیاد است زیرا هر بار تمام پارامتر های آن تغییر می کنند.

در رابطه با fine tune کردن برخی لایه ، محدودیت های دیگری داریم من جمله زمانی که دیتاست کافی نداشته باشیم ، با LoRA می توانیم همه لایه ها را از اول منتها با تعداد پارمتر های کم تر ترین کنیم و مجبور نیستیم یک لایه را به کل فریز کنیم و این از overfit جلوگیری می کند منتها زمانی که فقط چند لایه را ترین می کنیم این امکان را نداریم.

پاسخ ۳ – تشخیص تقلب

۳-۱. آشنایی با دیتاست



تعداد داده‌های موجود از هر کلاس

همانطور که قابل مشاهده است، دیتاست داده شده بسیار نامتوازن است، چرا که تعداد در تراکنش‌های ناسالم بانکی بسیار کمتر از تعداد تراکنش‌های سالم می‌باشد. (۲۸۴۳۱۵ مورد در مقابل ۴۹۲ مورد.)

این عدم توزان در تعداد داده‌های هر کلاس موجب ایجاد مشکلاتی در آموزش شبکه بشود. این مشکلات عبارتند از:

۱- مدل بر روی داده‌های کلاس ۰ بسیار بیشتر از کلاس ۱ آموزش می‌بینند، این امر موجب می‌شود که دقت شبکه در تشخیص کلاس ۱ پایین بیاید. هر چند که ممکن است accuracy شبکه به طور کل در انتها بالا باشد، اما معیارهای مهمی مانند precision و recall داده‌های کلاس ۱ ممکن است کمتر از حد انتظار به دست بیایند.

۲- آموزش شبکه با تعداد پایین داده در کلاس ۱ می‌تواند باعث overfit شدن آموزش شبکه روی داده‌های این کلاس بشود به طوری که توانایی شبکه در تشخیص داده‌های جدید پایین بیاید.

۳-۲. پیاده سازی معماری مقاله

۱.

مدل معرفی شده قرار است تا داده‌های مربوط به معاملات را که از آن‌ها PCA گرفته شده است، به صورت برداری از ویژگی‌ها دریافت کرده و سپس در خروجی کلاس آن‌ها را تشخیص دهد.

معماری پیشنهاد شده توسط مقاله بر پایه‌ی لایه‌های کانولوشنالی یک بعدی، لایه‌های MaxPool یک بعدی و لایه‌های FC تشکیل شده است. به این صورت که featureها (به غیر از داده‌های زمانی که اهمیتی ندارد) به صورت برداری از ساختار زیر عبور می‌کنند.

TABLE II. CONVNETS MODEL LAYER ORIENTATION

Layer	Description
1	Conv1D Layer (filters = 32, kernel_size = 2, activation = relu)
2	BatchNormalization
3	MaxPool1D (pool_size = 2)
4	Dropout (rate = 0.2)
5	Conv1D Layer (filters = 64, kernel_size = 2, activation = relu)
6	BatchNormalization
7	MaxPool1D (pool_size = 2)
8	Dropout (rate = 0.5)
9	Flatten
10	Dense (units = 64, activation = relu)
11	Dropout (rate = 0.5)
12	Dense (units = 64, activation = relu)

معماری شبکه‌ی ConvNet

در لایه‌های اولیه کانولوشن اول از ۳۲ فیلتر با سایز ۲ به همراه MaxPool با سایز ۲ در نظر گرفته شده است. خروجی به لایه‌های کانولوشن دوم با ۶۴ فیلتر و سایز ۲ و MaxPool با سایز ۲ وارد می‌شود.

سپس مقادیر خروجی که به صورت ماتریس می‌باشند از لایه‌ی flatten گذشته و به صورت بردار وارد دو لایه‌ی FC با تعداد ۶۴ نورون وارد می‌شوند. در انتها نیز یک لایه‌ی Dense با تابع فعال‌ساز sigmoid استفاده می‌شود تا خروجی را به صورت یک عدد نمایانگر احتمال تعلق به کلاس ۱ به دست آوریم.

همچنین لایه‌های BatchNormalization و Dropout به هدف یادگیری آسان‌تر و عدم overfit مابین ساختار قرار داده شده‌اند.

برای پیش‌پردازش در dataset عدم وجود داده‌ی غیرقابل قبول چک شد، سپس داده‌ها به سه قسمت train، evaluation و test تقسیم شد و در نهایت scaler روی داده‌های train فیت شده و تمامی دیتاست با استفاده از scaler به دست آمده استاندارد شد.

```
[ ]  x_train_raw, x_test_raw, y_train, y_test = train_test_split(dataset[:, 0:-1], dataset[:, -1], test_size=0.2, random_state=50)
     x_train_raw, x_eval_raw, y_train, y_eval = train_test_split(x_train_raw, y_train, test_size=0.25, random_state=50)

[ ]  scaler = StandardScaler()
     scaler.fit(x_train_raw)
     x_train = scaler.transform(x_train_raw)
     x_eval = scaler.transform(x_eval_raw)
     x_test = scaler.transform(x_test_raw)

[ ]  print(x_train.shape)
     print(x_eval.shape)
     print(x_test.shape)

     (170883, 29)
     (56962, 29)
     (56962, 29)
```

فرآیند پیش‌پردازش داده‌ها

هایپر پارامترهای در نظر گرفته شده تا حد امکان مشابه مقاله در نظر گرفته شد:

optimizer = Adam

learning_rate = 0.0001

loss = BinaryCrossentropy

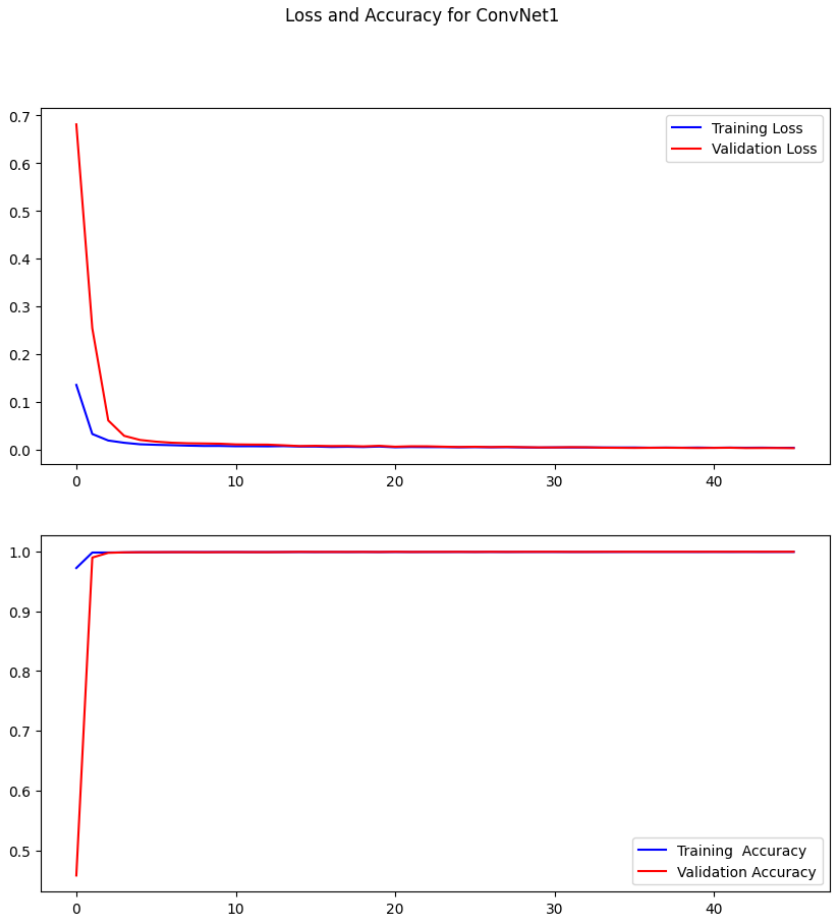
batch_size = 360

epochs = 46

```
[ ]  ConvNet = Sequential([
         Conv1D(filters=32, kernel_size=2, activation='relu', input_shape=(29, 1)),
         BatchNormalization(),
         MaxPool1D(pool_size=2),
         Dropout(rate=0.2),
         Conv1D(filters=64, kernel_size=2, activation='relu'),
         BatchNormalization(),
         MaxPool1D(pool_size=2),
         Dropout(rate=0.5),
         Flatten(),
         Dense(units=64, activation='relu'),
         Dropout(rate=0.2),
         Dense(units=64, activation='relu'),
         Dense(units=1, activation='sigmoid')
     ])
```

پیاده‌سازی معماری مدنظر با استفاده از Tensorflow

پس از این مراحل فرآیند آموزش شبکه آغاز شد.



نمودارهای تابع هزینه و دقت برای دو دیتاست train و evaluation به ازای هر epoch

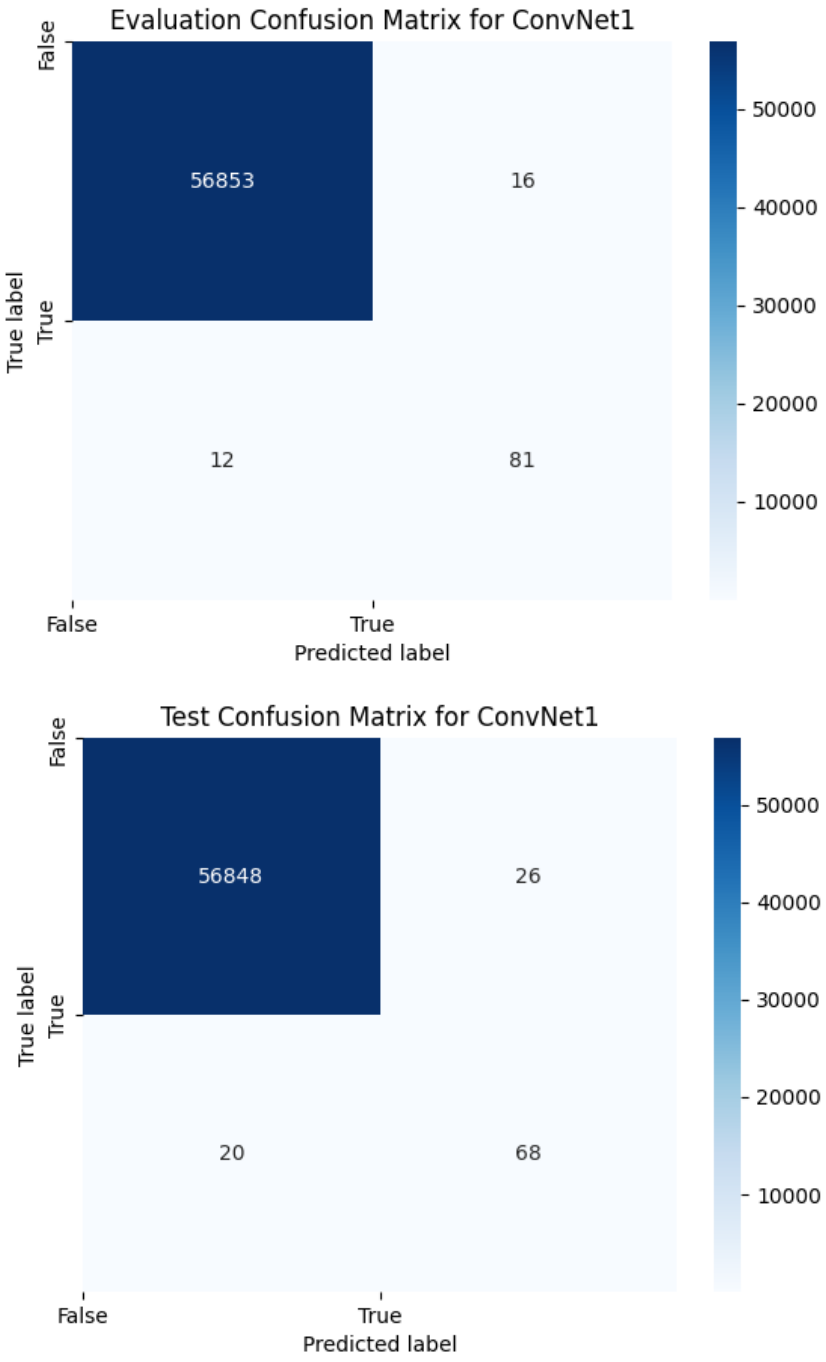
همانگونه که قابل مشاهده است شبکه به خوبی آموزش می‌بیند و چون نمودار loss دیتاست‌های train و test در نهایت به مقادیر کم همگرا می‌شوند، می‌توان فهمید روی داده‌های آموزش overfit رخ نداده است.

۲.

```
Evaluation Accuracy:   0.99951
Evaluation Precision:  0.83505
Evaluation Recall:     0.87097
Evaluation F1-score:   0.85263

   Test Accuracy:   0.99919
   Test Precision:  0.7234
   Test Recall:     0.77273
   Test F1-score:   0.74725
```

مقادیر متریک‌ها پس از آموزش شبکه روی دیتاست‌های test و evaluation



ماتریس‌های در هم‌ریختگی برای دیتاست‌های test و evaluation

۳.

همانگونه که در شکل‌های قبل مشخص است، دیده می‌شود که علی رغم مقدار خوب accuracy، سایر پارامترهای مدنظر یعنی precision، recall و f1score مقادیر بسیار کمتری را دارند. این به این معناست که عملکرد کاملا خوبی را از شبکه شاهد نبوده‌ایم. به طور مثال اگر شبکه‌ای وجود نداشت و همه‌ی داده‌ها را به کلاس ۰ نسبت می‌دادیم، باز هم accuracy بسیار بالایی را در پیش‌بینی خود مشاهده می‌کردیم. در

نتیجه داشتن accuracy بالا به تنهایی کافی نیست و سه متریک دیگر نیز نیاز است که بالا باشند. اهمیت هر کدام از این متریک‌ها به طور مختصر در زیر توضیح داده شده است:

۱- معیار precision: نشان می‌دهد از تعداد تشخیص‌های مثبت ما، چه درصدی واقعا مثبت بوده است.

۲- معیار recall: نشان می‌دهد از میزان مثبت‌های واقعی چند درصد را توانستیم تشخیص دهیم.

۳- معیار f1score: سعی دارد ترکیبی از precision و recall را برای توصیف کلی به ما عرضه کند.

با توجه به نوع مسئله، اینکه به صورت دقیق همه‌ی آنچه را که ناسالم است را تشخیص بدهیم، بدون آنکه سالم‌ها را به اشتباه ناسالم در نظر بگیریم، بسیار مهم است.

۳-۳. نمونه برداری

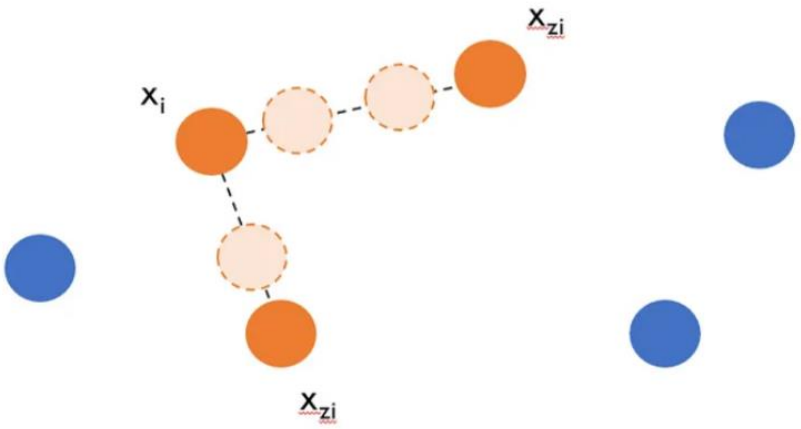
۱.

این روش سعی دارد که با استفاده موقعیت مکانی داده‌های دیتاست به صورت نقاط n بعدی، داده‌های کلاس با نمونه‌ی را به گونه‌ای بیشتر کند که نقاط جدید تولید شده:

۱- در مجاور با داده‌های همان کلاس باشند و توزیع آن‌ها در فضا را تقلید کنند.

۲- تعداد داده‌های تولید شده در اطراف نقاط ایزوله‌تر از کلاس با نمونه‌ی کمتر، بیشتر باشد، تا بتوان آن‌ها را بهتر یاد گرفت.

۳- اضافه کردن نقطه به این شیوه انجام می‌گیرد که سعی می‌شود بین نقطه‌های کلاس مد نظر، نقطه‌ای جدید به صورت تصافی ایجاد شود. (این عمل می‌توان به جای تولید رندوم نقطه روی خط مابین دو نقطه، نقطه‌ها را روی صفحه‌های دارای سه نقطه یا ابرصفحه‌ها انجام گیرد.)



الگوریتم ADASYN به صورت تصویری

مزایا:

+ حفظ توزیع کلی داده‌های با کلاس کمتر در فضای n بعدی

+ استفاده از اطلاعات داده خود کلاس برای تولید داده‌ی جدیدتر

+ جلوگیری از overfit کردن مدل روی نقاط پراکنده و آموزش مدل روی یک ناحیه

معایب:

- این روش به داده‌های نویزی حساس است، چرا که داده‌های نویزی معمولا ایزوله‌تر هستند و این روش برای آن‌ها نیز داده‌ی اضافه تولید می‌کند.

- ممکن است ADASYN موجب کاهش دقت شود، این امر به نوع توزیع داده‌های کلاس‌ها مربوط است.

۲.

تابع زیر عملیات اضافه کردن نمونه به کلاس کوچکتر را انجام می‌دهد. شرح عملکرد این تابع در زیر عکس آورده شده است.

```
[ ]  def ADASYN(x, y, beta, k):
       np.random.seed(50)
       minor_num = np.sum(y)
       major_num = len(y) - minor_num

       minors = x[y==1]
       majors = x[y==0]

       synth_num = ((major_num-minor_num)*beta).astype(int)

       kn_minor_args_list = []
       k_nearest_minors_args = []
       weights = []
       for m in minors:
           d = np.sum((m-x)**2, axis=1)
           kn_args = np.argsort(d)[1:k+1]
           kn_minor_args = kn_args[y[kn_args]==1]
           kn_minor_args_list.append(kn_minor_args)
           weight = k-np.sum(y[kn_args])
           weights.append(weight%k)
       weights = np.array(weights)
       minors_synth_num = (synth_num*weights/np.sum(weights)).astype(int)

       synth_x = []
       synth_y = np.ones(np.sum(minors_synth_num))
       for i, m in enumerate(minors):
         synth_num = minors_synth_num[i]
         for j in range(synth_num):
           first_point = m
           second_point = x[np.random.choice(kn_minor_args_list[i])]
           synth_point = first_point+(second_point-first_point)*np.random.rand()
           synth_x.append(synth_point)
       x = np.concatenate((x, np.array(synth_x)), axis=0)
       y = np.concatenate((y, synth_y))
       return x, y
```

تابع ADASYN

۱- به دست آوردن تعداد نمونه‌های لازم برای ایجاد شدن: این مقدار با استفاده از تعداد داده‌های هر کلاس و پارامتر بتا صورت می‌گیرد.

۲- به دست آوردن kتا از نزدیک‌ترین نقطه‌های داخل دیتاست به هر کدام از نقطه‌های داخل دیتاست اقلیت

۳- به دست آوردن تعداد نقطه‌های مربوط به دیتاست اکثریت در هر مجموعه‌ی k نقطه‌ای مربوط به نقطه‌های اقلیت و به دست آوردن نسبت حضور آن‌ها در دیتاست(weights)

۴- جمع تمامی وزن‌ها و نرمال کردن آن‌ها به شکلی که مجموع تمامی آن‌ها برابر ۱ شود.

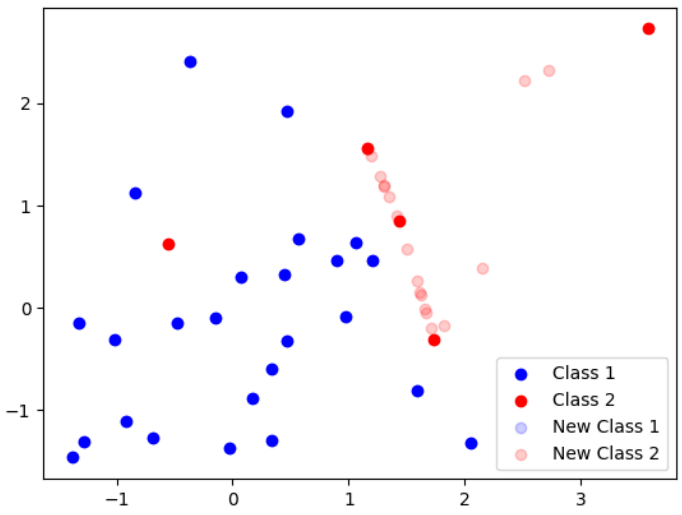
۵- با ضرب مقادیر نرمال در تعداد نمونه‌های لازم برای تولید، می‌توانیم تعداد نقطه‌های تولید شده برای هر کدام از نقطه‌های اقلیت را به دست آوریم.

۶- با داشتن تعداد نمونه‌ی لازم برای ساختن، روی هر کدام از نقطه‌های اقلیت گردش می‌کنیم و برای آن‌ها به تعداد لازم نمونه‌ی جدید می‌سازیم.

۷- عملیات ساخت نمونه‌ی جدید به این شکل است که از نقاط اقلیت مجاور با نقطه‌ی اقلیت فعلی به تعداد نمونه‌ی مورد نیاز بار انتخاب می‌کنیم و هر سری میان نقطه‌ی انتخابی و نقطه‌ی کنونی داده‌ای را به صورت رندوم ایجاد می‌کنیم.

۸- در انتها داده‌های جدید را به همراه برچسبشان به داده‌های اصلی اضافه می‌کنیم و به عنوان خروجی بیرون می‌دهیم.

عملکرد این تابع روی یک دیتاست دوبعدی به شکل زیر نمایش داده شده است. (Part Extra 2)



نمایش گرافیکی عملکرد تابع ADASYN روی دیتاست دوبعدی

.۳

با توجه به اینکه هدف ما از نمونه‌برداری تنها کمک به مرحله‌ی آموزش مدل می‌باشد، حق نداریم که داده‌های ارزیابی و تست را نیز نمونه‌برداری کنیم. تنها می‌توانیم مقادیر موجود در هر کلاس را در دیتای آموزش متوازن کنیم.

اگر از کل داده‌ها برای ساخت نمونه‌ی جدید استفاده کنیم، از لحاظ آماری کار غلطی انجام داده‌ایم و فرض دست نخورده بودن داده‌های ارزیابی و تست را زیر پا گذاشته‌ایم و متریک‌های به دست آمده فاقد اعتبار خواهند بود.

.۴

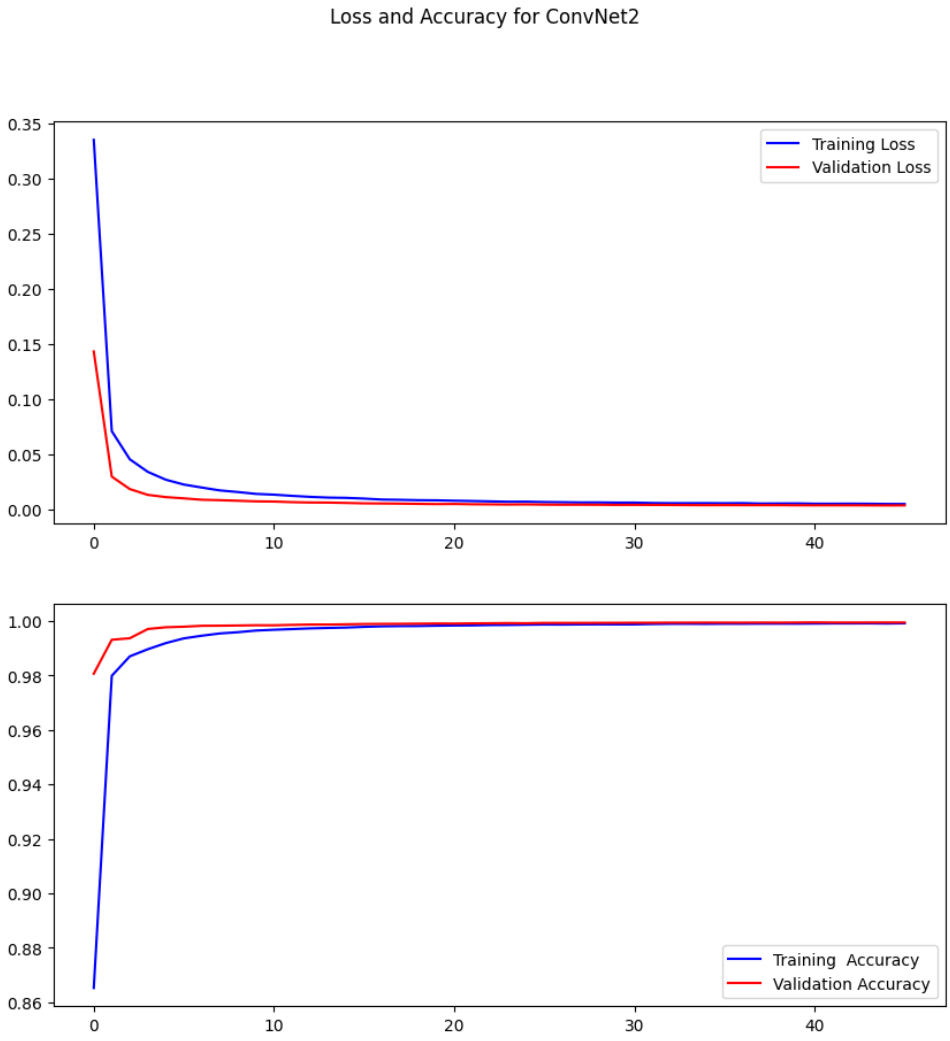
این تابع بر روی کل دیتاست اجرا شد و خروجی هیستوگرام دیتاست جدید در زیر نمایش داده شده است.

* اجرای عملیات نمونه‌برداری باید تنها روی دیتاست ترین انجام می‌شد، اما این مرحله مشابه آنچه در مقاله انجام شده بود صورت گرفت واز کل دیتاست نمونه‌برداری و ساخت نمونه‌ی جدید صورت گرفت.

تعداد داده‌های موجود از هر کلاس پس از نمونه‌برداری

۳-۴. آموزش مدل با داده های جدید

در قسمت پیش‌پردازش مشابه آنچه در قسمت قبل کرده بودیم عمل می‌کنیم. سپس داده‌ها را با همان هایپرپارامترهای سابق به همان شبکه می‌دهیم تا عملیات آموزش صورت پذیرد.



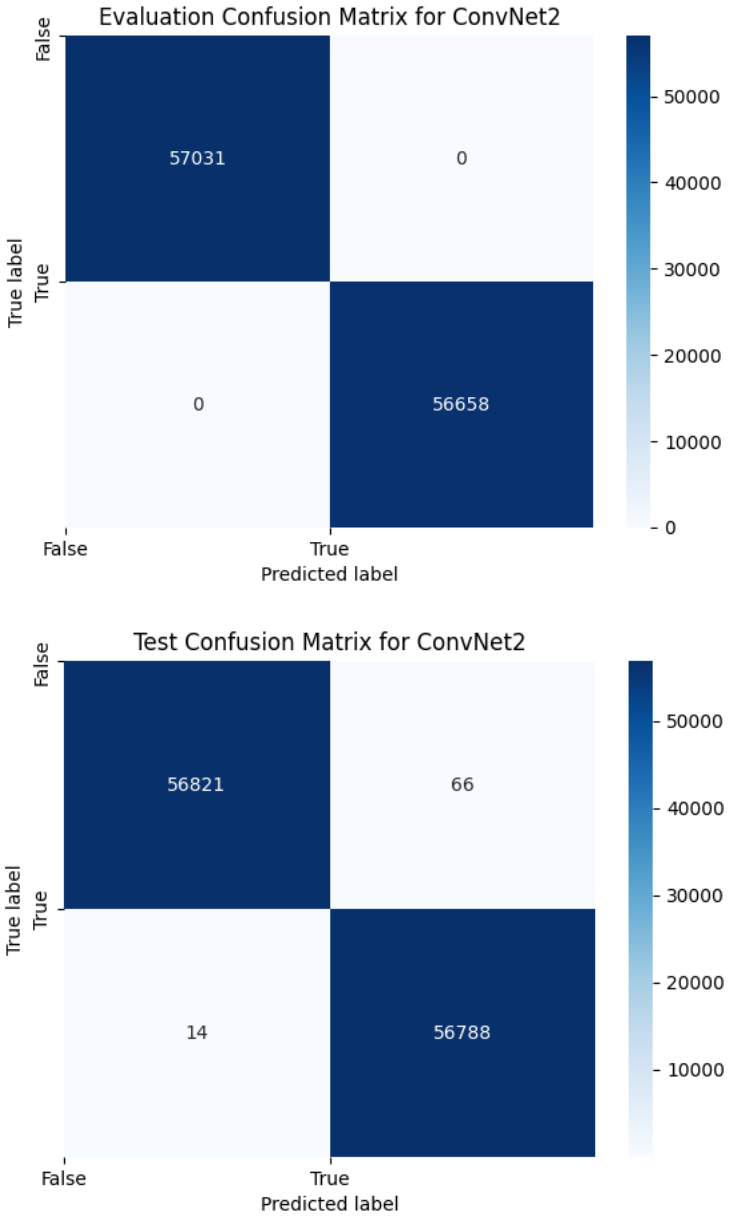
نمودارهای تابع هزینه و دقت برای دو دیتاست train و evaluation به ازای هر epoch برای مدل جدید

همانگونه که دیده می‌شود آموزش به خوبی انجام گرفته و به خاطر همگرایی دو نمودار مربوط به مقدار loss دیتاست‌های train و evaluation، شاهد overfit نمی‌باشیم.

```
Evaluation Accuracy:   1.0
Evaluation Precision:  1.0
Evaluation Recall:     1.0
Evaluation F1-score:   1.0

 Test Accuracy:   0.9993
 Test Precision:  0.99884
 Test Recall:     0.99975
 Test F1-score:   0.9993
```

مقادیر متریک‌ها پس از آموزش شبکه روی دیتاست‌های test و evaluation برای مدل جدید

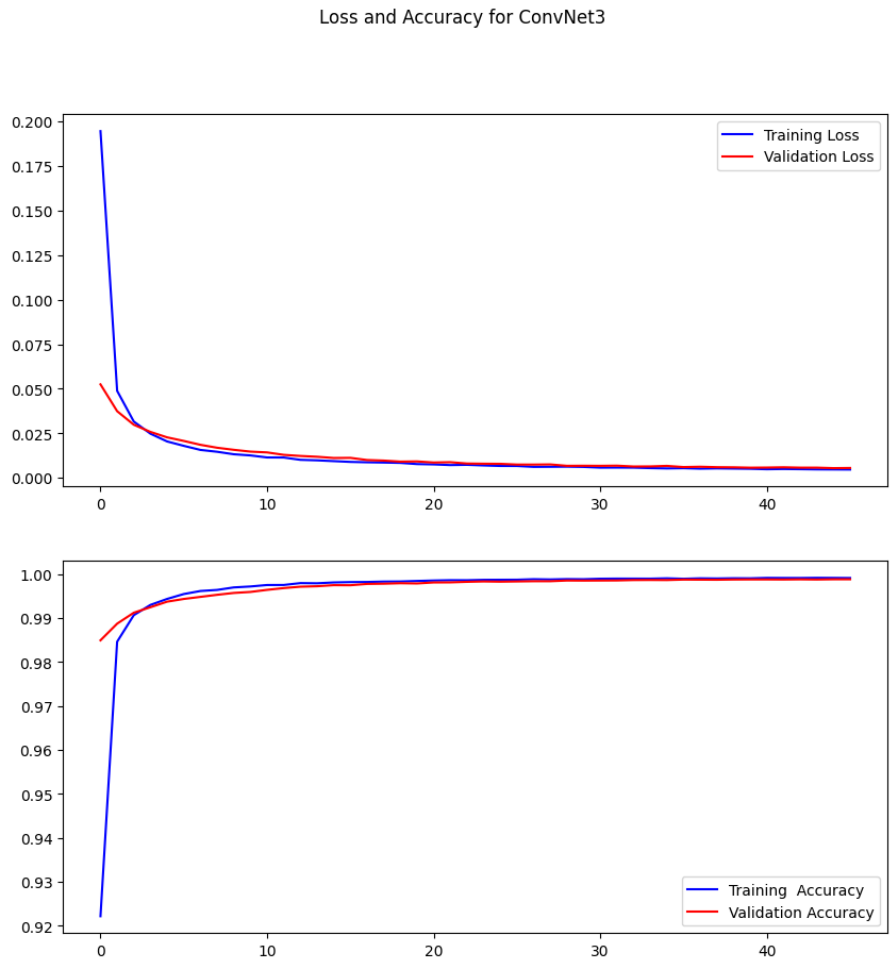


ماتریس‌های در هم‌ریختگی برای دیتاست‌های test و evaluation برای مدل جدید

همانگونه که مشاهده می‌شود، متریک‌ها در مقایسه با قسمت اول بسیار بهتر شده و در اینجا نه تنها accuracy بالا می‌باشد، بلکه متریک‌های دیگر نیز بسیار بالاتر شده‌اند که نشان‌دهنده‌ی دقت بالای شبکه از هر نظری می‌باشد.

* در بخش بعد، نمونه‌برداری به شکل صحیح انجام شده و نتایج قابل اتکا در آن آمده است.

۳-۵. بخش اضافه‌ی ۱

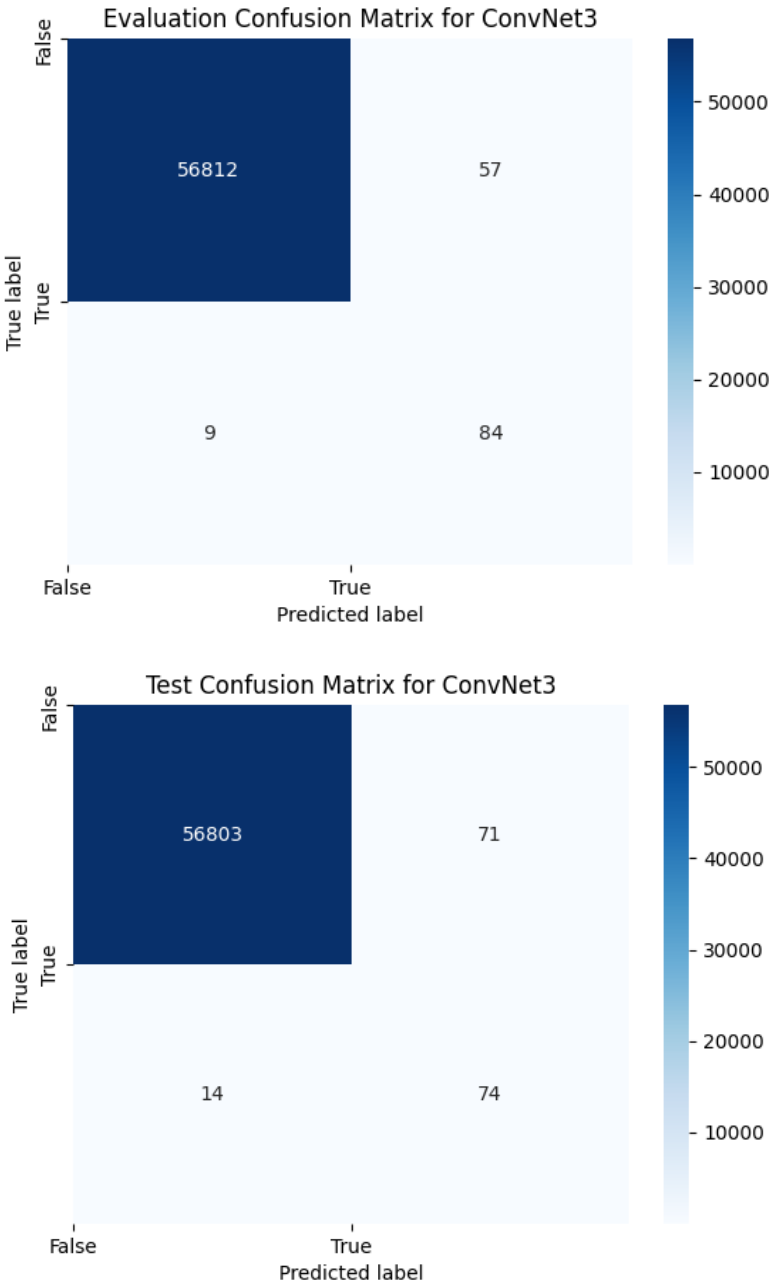


نمودارهای تابع هزینه و دقت برای دو دیتاست train و evaluation به ازای هر epoch برای مدل نهایی

```
Evaluation Accuracy:   0.99884
Evaluation Precision:  0.59574
Evaluation Recall:     0.90323
Evaluation F1-score:   0.71795
```

Test Accuracy:	0.99851
Test Precision:	0.51034
Test Recall:	0.84091
Test F1-score:	0.31759

مقادیر متریک‌ها پس از آموزش شبکه روی دیتاست‌های evaluation و test



ماتریس‌های در هم‌ریختگی برای دیتاست‌های test و evaluation برای مدل نهایی

بنا به نتیاج به دست آمده دیده می‌شود انجام عملیات نمونه‌برداری به شدت مقدار precision را کم کرده است که خوب نیست، ولی تا حد خوبی recall را افزایش داده است. این به این معناست که مدل نسبت به قبل راحتتر تراکنش‌ها را ناسالم در نظر می‌گیرد. f1-score این مدل از مدل اولیه بسیار بدتر است.