به نام خدا

دانشگاه تهران

دانشکده مهندسی

برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین امتیازی

نام و نام خانوادگی	بهراد موسایی شیرمحمد محمد جواد رنجبر کلهرودی
شماره دانشجویی	۸۱۰۱۰۱۲۷۸ ۸۱۰۱۰۱۱۷۳
تاریخ ارسال گزارش	۱۴۰۲.۱۰.۲۷

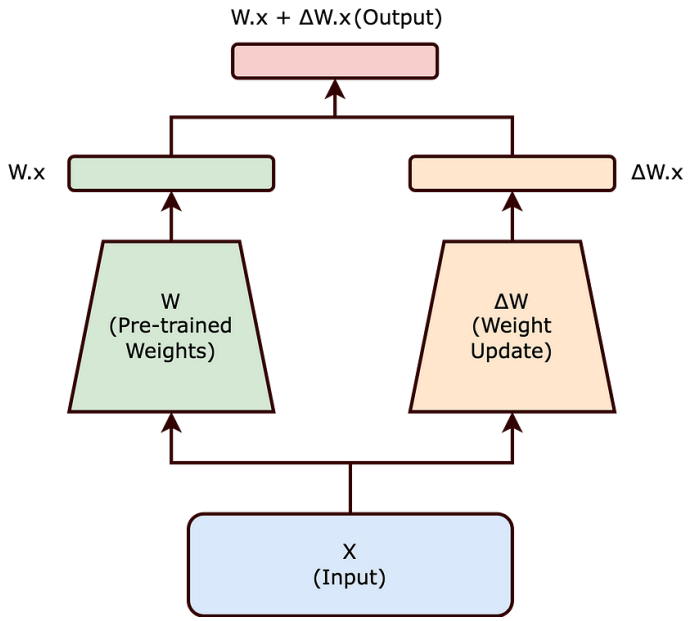
فهرست

۲. LoRA

۲-۱. LoRA چگونه کار می کند:

فاین تون کردن کل پارامترها

در فاین تون کردن کامل، مدل با وزن‌های پیش آموزش داده شده Φ_0 مقداردهی اولیه می‌شود و سپس به $\Phi_0 + \Delta\Phi$ به‌روزرسانی می‌شود. این به‌روزرسانی با دنبال کردن گرادیان برای حداکثر کردن هدف مدل‌سازی زبان شرطی انجام می‌گیرد. برای هر وظیفه جدید، مجموعه‌ای متفاوت از پارامترها $\Delta\Phi$ آموخته می‌شود که ابعاد آن برابر با ابعاد وزن‌های پیش آموزش داده شده $|\Phi_0|$ است. این می‌تواند یکی از معایب اصلی فاین تون کردن کامل باشد چرا که اگر مدل پیش آموزش داده شده بزرگ باشد (مانند GPT۳ با $|\Phi_0|$ تقریباً ۱۷۵ میلیارد)، ذخیره‌سازی و استقرار نمونه‌های مستقل و متعدد از مدل‌های فاین تون شده می‌تواند چالش‌برانگیز باشد، اگر اصلاً ممکن باشد.



شکل ۱ روش سنتی تنظیم دقیق

فاین تون کردن یک یا چند لایه از مدل

فاین تون کردن تنها برخی از لایه‌های مدل یک نوع ساده‌تر از فاین تون کردن کامل است که در آن تنها برخی از لایه‌ها به‌روزرسانی می‌شوند در حالی که بقیه لایه‌ها بدون تغییر باقی می‌مانند. به عنوان مثال، در یک مطالعه قبلی (Li & Liang ۲۰۲۱) درباره GPT۲، فقط دو لایه آخر مدل (FTTop۲) فاین تون شدند. این رویکرد اجازه می‌دهد تا با کاهش تعداد پارامترهایی که نیاز به به‌روزرسانی دارند، نیاز به ذخیره‌سازی و پردازش کمتری داشته باشیم. به عنوان مثال، ذخیره‌سازی ۱۰۰ مدل تطبیق داده شده با

این رویکرد فقط نیاز به ۳۵۴ گیگابایت فضا دارد، در حالی که با فاین تون کردن کامل برای هر مدل نیاز به ۳۵۰ گیگابایت فضا خواهیم داشت، که مجموعاً ۳۵ ترابایت فضا نیاز دارد.

Fine tune with LoRA

حال به صورت کامل داریم که LoRA , روش (LowRank Adaptation)LoRA یک استراتژی فاین تون است که وزن‌های پیش‌آموزش داده شده‌ی مدل را ثابت نگه می‌دارد و ماتریس‌های قابل آموزش تجزیه رتبه پایین را به هر لایه از معماری Transformer تزریق می‌کند. این کار به طور چشمگیری تعداد پارامترهای قابل آموزش برای وظایف پایین‌دست را کاهش می‌دهد. به عنوان مثال، در مقایسه با GPT۳ که با آدام فاین تون شده، LoRA تعداد پارامترهای قابل آموزش را تا ۱۰۰۰۰ برابر و نیاز به حافظه GPU را تا سه برابر کاهش می‌دهد.

وقتی یک مدل زبان را به دقت تنظیم می کنیم، پارامترهای زیربنایی مدل را اصلاح می کنیم. برای ملموس‌تر کردن این ایده، می‌توانیم به‌روزرسانی پارامتر حاصل از تنظیم دقیق را همانطور که در معادله زیر نشان داده شده است، فرموله کنیم.

$$\overbrace{W_{\mathtt{ft}}}^{\text{Finetuned Weights}} = \underbrace{W_{\mathtt{pt}}}_{\text{Pretrained Weights}} + \overbrace{\Delta W}^{\text{Weight Update}}$$

شکل ۲ نحوه بروزرسانی وزن‌های سنتی

ایده اصلی پشت LoRA مدل سازی این به روز رسانی به پارامترهای مدل با تجزیه با رتبه پایین ۹ است که در عمل به عنوان یک جفت پیش بینی خطی اجرا می شود. LoRA لایه های از پیش آموزش دیده LLM را ثابت می گذارد و یک ماتریس تجزیه رتبه قابل آموزش را به هر لایه از مدل تزریق می کند. زیر را ببینید.

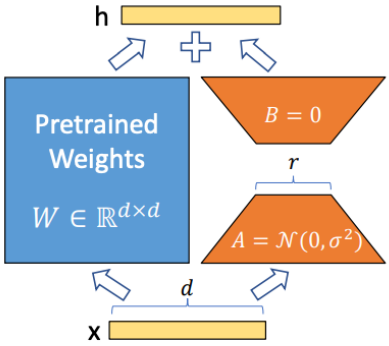


Figure 1: Our reparametrization. We only train A and B.

شکل ۳ نحوه تنظیم دقیق با **LoRa**

حال اجزای LoRA را توضیح می‌دهیم:

ماتریس تجزیه رتبه به زبان ساده، ماتریس تجزیه رتبه فقط دو طرح خطی است که ابعاد ورودی را کاهش داده و بازیابی می کند. خروجی این دو پیش بینی خطی به خروجی حاصل از وزن های از پیش آموزش دیده مدل اضافه می شود. لایه به روز شده ای که با افزودن این دو تبدیل موازی تشکیل می شود، مطابق شکل زیر فرموله شده است. همانطور که می بینیم، افزودن LoRA به یک لایه مستقیماً به روز رسانی وزن لایه زیرین را یاد می گیرد.

$$W_{\text{ft}} = W_{\text{pt}} + \underbrace{\Delta W}_{\text{Approximation}} = W_{\text{pt}} + \overbrace{AB}^{\text{Rank Decomposition Matrix}}$$

$$\text{where} \quad W_{\text{ft}}, W_{\text{pt}}, \Delta W, AB \in \mathbb{R}^{d \times d}$$

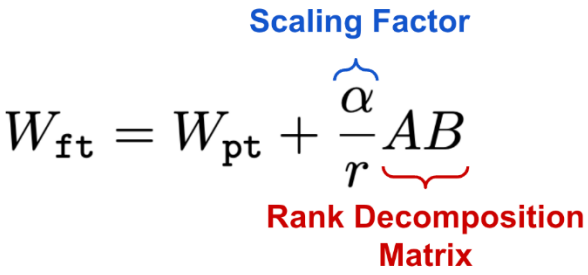
$$\text{and} \quad \underbrace{A \in \mathbb{R}^{d \times r}, B \in \mathbb{R}^{r \times d}}_{\text{Low Rank}}$$

شکل ۴ نحوه بروزرسانی وزن‌ها با **LoRa**

محصول ماتریسی AB همان ابعاد به‌روزرسانی تنظیم دقیق را دارد. تجزیه به روز رسانی به عنوان محصولی از دو ماتریس کوچکتر تضمین می کند که به روز رسانی رتبه پایینی دارد و تعداد پارامترهایی را که باید آموزش دهیم به میزان قابل توجهی کاهش می دهد. به جای تنظیم دقیق پارامترها در لایه های LLM از پیش آموزش دیده، LoRA فقط ماتریس تجزیه رتبه را بهینه می کند و نتیجه ای به دست می دهد که به روز رسانی حاصل از تنظیم دقیق کامل را تقریبی می کند. ما A را با مقادیر تصادفی و کوچک

مقداردهی اولیه می‌کنیم، در حالی که B به صورت صفر مقداردهی می‌شود، و اطمینان حاصل می‌کنیم که فرآیند تنظیم دقیق را با وزن‌های اصلی و از پیش آموزش‌دیده‌شده مدل آغاز می‌کنیم.

افزایش r تقریب LoRA را در مورد به‌روزرسانی کامل تنظیم دقیق بهبود می‌بخشد، اما مقادیر فوق‌العاده کوچک r در عمل کافی است و به ما این امکان را می‌دهد که هزینه‌های محاسباتی و حافظه را با کمترین تأثیر بر عملکرد به میزان قابل توجهی کاهش دهیم. به عنوان مثال، ما می‌توانیم از LoRA برای تنظیم دقیق GPT-۳ با استفاده از تنها ۰.۰۱٪ از کل پارامترها استفاده کنیم و همچنان عملکردی قابل مقایسه با تنظیم دقیق کامل داشته باشیم.



شکل ۵ ضریب مقیاس در **LoRA**

ضریب پوسته پوسته شدن هنگامی که به‌روزرسانی رتبه پایین ماتریس وزن به دست آمد، قبل از اضافه کردن آن به وزن‌های از پیش آموزش‌دیده‌شده مدل، آن را با ضریب α مقیاس می‌کنیم. چنین رویکردی قانون انطباق نشان داده شده در بالا را به همراه دارد. مقدار پیش‌فرض ضریب مقیاس‌بندی یک است، به این معنی که وزن‌های از پیش تمرین‌شده و به‌روزرسانی وزن کم‌تر هنگام محاسبه پاس رو به جلو مدل، به طور مساوی وزن می‌شوند. با این حال، مقدار α را می توان تغییر داد تا اهمیت مدل از پیش آموزش دیده و انطباق ویژه کار جدید را متعادل کند. تجزیه و تحلیل تجربی اخیر نشان می دهد که مقادیر بزرگتر α برای LoRA با رتبه بالاتر (یعنی α la -> r بزرگتر) ضروری است.

۲۲. قرار است چه کاری بر روی داده ها صورت پذیرد؟

در این سوال قرار است از مدل زبانی بزرگ ROBERT نسخه ی Large استفاده کنید. این مدل را میتوان به صورت آماده و با دستور زیر دریافت کرد میتوانید از هر روش دیگری نیز استفاده کنید ولی در این صورت از سالم بودن مدل دریافت شده اطمینان حاصل نمایید.

```python
from transformers import AutoTokenizer, AutoModel

tokenizer = AutoTokenizer.from_pretrained("roberta-large")
model = AutoModel.from_pretrained("roberta-large")
```

شکل ۶: روش بارگذاری داده ها

با توجه به اطلاعاتی که داده شده، ابتدا به توضیحات کلی در مورد دیتاست پرداخته و سپس روش آموزش مدل با استفاده از LoRA را شرح می‌دهم:

توضیحات دیتاست

مجموعه داده چند سبک تداول زبان طبیعی (MultiNLI) یک مجموعه از ۴۳۳ هزار جفت جمله است که با اطلاعات تضمین متنی ارزیابی شده‌اند و به صورت گروهی به صورت برون‌سپاری جمع‌آوری شده‌اند. این مجموعه داده بر پایه مجموعه داده SNLI مدل شده است، اما با این تفاوت که گستره‌ای از سبک‌های متن‌های گفتاری و نوشتاری را پوشش می‌دهد و ارزیابی تعمیم از سبک متفاوت را پشتیبانی می‌کند. این مجموعه داده به عنوان پایه‌ای برای وظیفه مشترک کارگاه RepEval در کنفرانس EMNLP ۲۰۱۷ در کپنهاگ خدمت کرده است.

Premise	Label	Hypothesis
Fiction		
The Old One always comforted Ca'daan, except today.	*neutral*	Ca'daan knew the Old One very well.
Letters		
Your gift is appreciated by each and every student who will benefit from your generosity.	*neutral*	Hundreds of students will benefit from your generosity.
Telephone Speech		
yes now you know if if everybody like in August when everybody's on vacation or something we can dress a little more casual or	*contradiction*	August is a black out month for vacations in the company.
9/11 Report		
At the other end of Pennsylvania Avenue, people began to line up for a White House tour.	*entailment*	People formed a line at the end of Pennsylvania Avenue.

شکل ۷ نمونه‌هایی از مجموعه داده **MultiNLI**

مجموعه داده شامل جفت جملات از ده ژانر مختلف، مانند داستان، دولتی و مکالمات تلفنی است. هر جفت با یکی از سه دسته برچسب گذاری شده است: "مضاف"، "تضاد" یا "خنثی" که نشان دهنده رابطه بین دو جمله است.

MultiNLI به عنوان معیاری برای ارزیابی عملکرد مدل های مختلف NLP به کار گرفته شده است و نقش مهمی در پیشرفت تحقیقات در زمینه درک زبان طبیعی ایفا کرده است.

محققان و متخصصان اغلب از مجموعه داده MultiNLI برای آموزش و ارزیابی مدل‌ها استفاده می‌کنند و این به یک معیار استاندارد در جامعه NLP تبدیل شده است.

حال برای آموزش مدل باید مراحل زیر را طی کنیم:

مجموعه داده MultiNLI را بارگیری کنید:

مجموعه داده MultiNLI را که معمولاً در کتابخانه Hugging Face Datasets موجود است، بدست آورید.

توکن سازی:

از توکنایزر RoBERTa برای تبدیل داده های متن خام به فرمتی مناسب برای ورودی مدل استفاده کنید.

پیکربندی مدل:

مدل RoBERTa از قبل آموزش دیده را برای طبقه بندی توالی بارگذاری کنید و آن را برای کار خاص خود پیکربندی کنید.

تنظیم دقیق:

راه‌اندازی خط لوله تنظیم دقیق، از جمله تعریف پارامترهای آموزشی، بهینه‌ساز و زمان‌بندی نرخ یادگیری.

حلقه آموزشی:

حلقه آموزشی را پیاده سازی کنید، از طریق مجموعه داده آموزشی تکرار کنید، تلفات را محاسبه کنید، و وزن مدل را با استفاده از پس انتشار به روز کنید.

اعتبار سنجی:

به طور دوره‌ای مدل را بر روی مجموعه اعتبارسنجی ارزیابی کنید تا عملکرد آن در طول آموزش نظارت شود.

تنظیم فراپارامتر:

فراپارامترهایی مانند نرخ یادگیری، اندازه دسته ای و تعداد دوره های آموزشی را بر اساس عملکرد مدل در مجموعه اعتبار سنجی تنظیم کنید.

ارزیابی در مجموعه تست:

پس از آموزش، عملکرد تعمیم مدل را در مجموعه آزمون ارزیابی کنید.

ورودی مدل RoBERTa در مسائل پردازش زبان طبیعی (NLP) به شکل دنباله‌ای از توکن‌ها (واژه‌ها یا زیرکلمات) می‌باشد. این دنباله توکن‌ها به وسیله‌ی توکنایزر مربوط به مدل تولید می‌شود. برای مثال، اگر شما یک جمله دارید، این جمله توسط توکنایزر RoBERTa به یک دنباله از شناسه‌های عددی (توکن‌ها) تبدیل می‌شود که به مدل وارد می‌شود.

خروجی مدل RoBERTa در وظیفه‌های متفاوت می‌تواند متفاوت باشد. در وظیفه تشخیص دسته‌ای (مانند طبقه‌بندی دسته‌ها)، خروجی احتمالات اختصاص داده شده به هر دسته است. برای وظایف دیگر مانند ترجمه یا تولید متن، خروجی ممکن است شامل متن تولیدی باشد.

اگر مدل RoBERTa را به عنوان یک طبقه‌بندی‌کننده در MultiNLI داشته باشید، از آنجا که ما دارای ۳ کلاس هستیم نیاز به اضافه کردن یک لایه با سه نورون به انتهای مدل می‌باشم.

برای اضافه کردن LoRA، به این صورت عمل می‌کنیم که LoRA لایه های از پیش آموزش دیده LLM را ثابت می گذارد و یک ماتریس تجزیه رتبه قابل آموزش را به هر لایه از مدل تزریق می کند.

۳۲. کد نویسی: آموزش مدل

حال شروع به پیاده سازی کد می کنیم:

در این بخش که با استفاده از کتابخانه Hugging Face Transformers و سایر بسته‌های مرتبط، یک محیط یادگیری ماشین را برای وظایف پردازش زبان طبیعی تنظیم و پیکربندی می‌کند. ابتدا کتابخانه های مختلف پایتون مانند PyTorch، Hugging Face Transformers و مجموعه داده ها را وارد می کند که برای توسعه و آموزش مدل های یادگیری ماشین ضروری هستند. بررسی می‌کند که آیا یک GPU CUDA در دسترس است یا نه و دستگاه را بر این اساس اختصاص می‌دهد. در این مورد خاص، اگر یک GPU در دسترس باشد، از آن استفاده خواهد کرد. در غیر این صورت، به استفاده از CPU برمی گردد. این انتخاب دستگاه برای بهینه سازی فرآیند آموزش مدل های یادگیری عمیق مهم است زیرا GPU ها می توانند محاسبات را به طور قابل توجهی تسریع کنند. کد اندازه دسته را روی ۳۲ تنظیم می کند که تعداد نمونه های ورودی پردازش شده در هر تکرار در طول آموزش را تعیین می کند. به طور کلی، این قطعه کد تنظیمات محیط ضروری را برای وظایف یادگیری ماشینی که شامل داده های متنی است، ایجاد می کند.

در ادامه کد هایی که پیاده سازی می کنیم به صورت زیر بیان می شود:

این بخش داده را برای کار MultiNLI MNLI از معیار ارزیابی درک عمومی زبان (GLUE) با استفاده از کتابخانه «مجموعه‌های داده» Hugging Face بارگیری و آماده می‌کند. مجموعه داده‌های MNLI را به سه بخش تقسیم می‌کند: یک مجموعه داده آموزشی ("train_data")، یک مجموعه داده اعتبارسنجی برای داده‌های منطبق ("val_m_data") و یک مجموعه داده اعتبارسنجی برای داده‌های ناهمخوان ("val_mm_data"). سپس اطلاعات مربوط به این مجموعه داده ها را چاپ می کند. علاوه بر این، یک متریک برای ارزیابی وظیفه MNLI به نام "metric" با استفاده از تابع "load_metric" از همان کتابخانه بارگیری می کند. این کد معمولاً برای راه‌اندازی و آماده‌سازی داده‌ها برای آموزش و ارزیابی مدل‌های یادگیری ماشین در تکلیف MNLI در معیار GLUE، که معمولاً برای درک زبان طبیعی و وظایف طبقه‌بندی متن استفاده می‌شود، استفاده می‌شود.

```
Train dataset:
 Dataset({
    features: ['premise', 'hypothesis', 'label', 'idx'],
    num_rows: 392702
})
Validation matched dataset:
 Dataset({
    features: ['premise', 'hypothesis', 'label', 'idx'],
    num_rows: 9815
})
Validation mismatched dataset:
 Dataset({
    features: ['premise', 'hypothesis', 'label', 'idx'],
    num_rows: 9832
})
```

شکل ۸: نمایی از دیتاست

در ادامه پیاده سازی داریم:

برای پردازش داده های متنی، به ویژه برای یک کار پردازش زبان طبیعی با استفاده از یک مدل RoBERTa از پیش آموزش دیده استفاده می شود. در ابتدا، یک نشانه‌ساز ("RobertaTokenizer") مخصوص مدل "robertabase" بارگذاری می‌شود و متن به حروف کوچک تبدیل می‌شود. "عملکرد_tokenize" برای توکن کردن دو فیلد متنی ("Premise" و "Hopothesis") در داده‌های

ورودی تعریف شده است، و اطمینان حاصل می‌کند که آنها به طول حداکثر ۱۲۸ توکن اضافه یا کوتاه شده‌اند. مجموعه داده‌های آموزشی ("train_data")، اعتبارسنجی منطبق ("val_m_data") و اعتبارسنجی نامتناسب ("val_mm_data") برای تکرارپذیری با ۴۲ عدد ترکیب می‌شوند و تنها ۱۰۰۰ نمونه اول از هر کدام انتخاب می‌شوند. سپس این داده‌ها با استفاده از tokenize_function به صورت دسته‌ای توکن می‌شوند. در نهایت، قالب این مجموعه داده‌ها روی تانسورهای PyTorch تنظیم می‌شود که فقط شامل ستون‌های ضروری است: ««input_ids، attention_mask»» و «label». این فرآیند مجموعه‌های داده را برای آموزش و اعتبارسنجی در یک مدل یادگیری ماشینی آماده می‌کند، و اطمینان حاصل می‌کند که در قالب و ساختار صحیح هستند.

```
RobertaForSequenceClassification(
  (roberta): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(50265, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
      (token_type_embeddings): Embedding(1, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): RobertaEncoder(
      (layer): ModuleList(
        (0-11): 12 x RobertaLayer(
          (attention): RobertaAttention(
            (self): RobertaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): RobertaSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): RobertaIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): RobertaOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
```

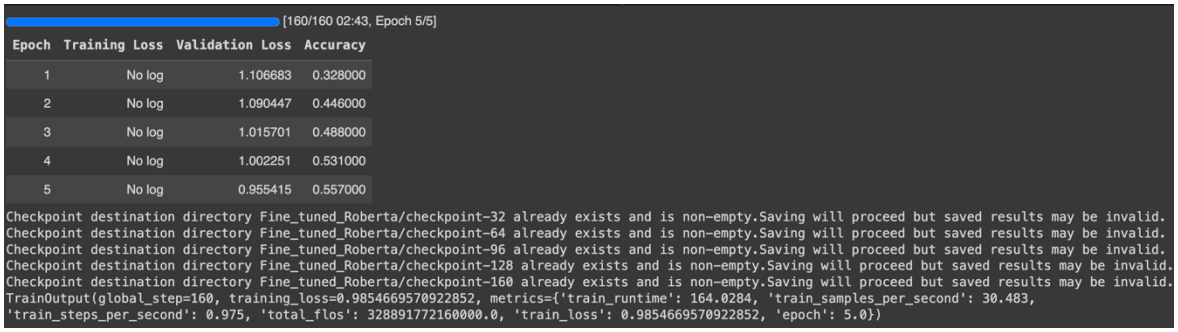
شکل ۹: مدل زبانی **Roberta**

در ادامه این بخش داریم:

آرگومان های آموزشی را برای تنظیم دقیق یک مدل RoBERTa از پیش آموزش دیده با استفاده از کتابخانه Transformers Hugging Face و تعریف تابعی برای محاسبه معیارها در حین ارزیابی تنظیم

می کند. کلاس TrainingArguments برای تعیین پارامترهای مختلف برای فرآیند آموزش استفاده می شود. این پارامترها شامل فهرست خروجی ("Fine_tuned_Roberta")، استراتژی های ارزیابی و ذخیره مدل ("دوران" برای هر دو)، نرخ یادگیری، اندازه دسته ای برای آموزش و ارزیابی، تعداد دوره های آموزشی، کاهش وزن برای منظم سازی، و دستورالعمل بارگذاری بهترین مدل در پایان آموزش بر اساس یک متریک مشخص ("دقت"). آرگومان «رزومه_از_نقطه_بازرسی» روی «درست» تنظیم شده است تا در صورت وجود، امکان ادامه آموزش از یک ایست بازرسی ذخیره شده را فراهم کند. تابع compute_metrics برای محاسبه عملکرد مدل در طول ارزیابی تعریف شده است. پیش‌بینی‌ها و برچسب‌ها را می‌گیرد، argmax پیش‌بینی‌ها را محاسبه می‌کند تا محتمل‌ترین پیش‌بینی‌های کلاس را به دست آورد، و سپس از یک «متریک» از پیش تعریف‌شده (مانند دقت) برای مقایسه این پیش‌بینی‌ها با برچسب‌های واقعی استفاده می‌کند و متریک محاسبه‌شده را برمی‌گرداند. این تنظیم برای آموزش یک مدل یادگیری ماشینی قوی و با عملکرد خوب ضروری است و امکان نظارت و بهینه سازی عملکرد آن را بر اساس دقت فراهم می کند.

حال در ادامه یک callback سفارشی برای کلاس Hugging Face Trainer تعریف می کند و یک محیط آموزشی برای یک مدل یادگیری ماشینی تنظیم می کند. کلاس «CustomCallback» یک زیر کلاس از «TrainerCallback» است و برای انجام اقدامات اضافی در پایان هر دوره در طول آموزش طراحی شده است. به طور خاص، برای ارزیابی مدل در مجموعه داده های آموزشی در پایان هر دوره، علاوه بر ارزیابی استاندارد در مجموعه داده های اعتبار سنجی تنظیم شده است. این با نادیده گرفتن روش «on_epoch_end» به دست می‌آید، جایی که بررسی می‌کند که آیا ارزیابی باید انجام شود («control.should_evaluate») و سپس ارزیابی روی مجموعه داده آموزشی («self._trainer.train_dataset») انجام می‌دهد. شی «Trainer» با پارامترهای مختلفی از جمله مدل، آرگومان‌های آموزشی («args»)، مجموعه داده‌های آموزش و ارزیابی («train_data» و «val_m_data»)، نشانه‌ساز و تابع «compute_metrics» سفارشی برای محاسبه متریک در طول ارزیابی: اگرچه پاسخ تماس سفارشی ("CustomCallback") تعریف شده است، اما نظر داده می شود و به مربی اضافه نمی شود، که نشان می دهد این یک جزء اختیاری است که در صورت نیاز می تواند فعال شود. متد «trainer.train()» در پایان فرآیند آموزش را شروع می کند. این تنظیم اجازه می دهد تا نظارت دقیقی بر عملکرد مدل، نه تنها بر روی داده های اعتبار سنجی دیده نشده، بلکه بر روی داده های آموزشی، ارائه بینشی در مورد پویایی یادگیری آن باشد.


[160/160 02:43, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	1.106683	0.328000
2	No log	1.090447	0.446000
3	No log	1.015701	0.488000
4	No log	1.002251	0.531000
5	No log	0.955415	0.557000

```
Checkpoint destination directory Fine_tuned_Roberta/checkpoint-32 already exists and is non-empty.Saving will proceed but saved results may be invalid.
Checkpoint destination directory Fine_tuned_Roberta/checkpoint-64 already exists and is non-empty.Saving will proceed but saved results may be invalid.
Checkpoint destination directory Fine_tuned_Roberta/checkpoint-96 already exists and is non-empty.Saving will proceed but saved results may be invalid.
Checkpoint destination directory Fine_tuned_Roberta/checkpoint-128 already exists and is non-empty.Saving will proceed but saved results may be invalid.
Checkpoint destination directory Fine_tuned_Roberta/checkpoint-160 already exists and is non-empty.Saving will proceed but saved results may be invalid.
TrainOutput(global_step=160, training_loss=0.9854669570922852, metrics={'train_runtime': 164.0284, 'train_samples_per_second': 30.483,
'train_steps_per_second': 0.975, 'total_flos': 328891772160000.0, 'train_loss': 0.9854669570922852, 'epoch': 5.0})
```

شکل ۱۰: پیاده سازی کلاس **Callback**

در ادامه پیاده سازی داریم قطعه کد برای ارزیابی عملکرد یک مدل یادگیری ماشینی آموزش دیده (احتمالاً یک مدل طبقه بندی متن با استفاده از Transformers Hugging Face) در یک مجموعه داده آزمایشی ("val_mm_data") است. ابتدا، پیش‌بینی‌های مدل روی مجموعه داده آزمایشی با استفاده از روش «trainer.predict()» به‌دست می‌آیند. سپس پیش‌بینی‌ها برای استخراج محتمل‌ترین برچسب‌های کلاس با استفاده از «np.argmax()» پردازش می‌شوند. این برچسب‌های پیش‌بینی‌شده با برچسب‌های واقعی از مجموعه داده آزمایشی برای محاسبه ماتریس سردرگمی مقایسه می‌شوند، ابزاری که به درک عملکرد مدل در کلاس‌های مختلف کمک می‌کند. سپس ماتریس سردرگمی چاپ می‌شود و تفکیک دقیقی از پیش‌بینی‌های صحیح و نادرست مدل برای هر کلاس ارائه می‌کند. علاوه بر این، یک گزارش طبقه‌بندی با استفاده از «classification_report()» تولید می‌شود که معیارهای مختلفی مانند دقت، یادآوری، و امتیاز F۱ را برای هر کلاس و همچنین میانگین‌های کلی ارائه می‌کند. این معیارها یک نمای کلی جامع از عملکرد طبقه‌بندی مدل ارائه می‌دهند و نقاط قوت و ضعف آن را در پیش‌بینی کلاس‌های مختلف برجسته می‌کنند. این ارزیابی برای درک اثربخشی مدل و هدایت بهبودهای بیشتر بسیار مهم است.

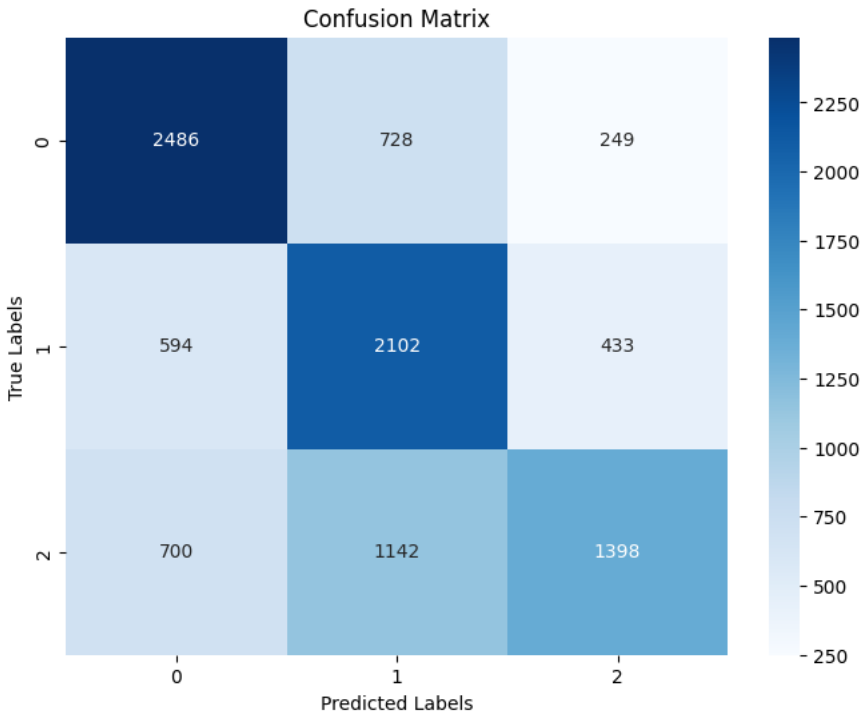
نتایج Roberta

```
Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.72      0.69      3463
           1       0.53      0.67      0.59      3129
           2       0.67      0.43      0.53      3240

    accuracy                           0.61      9832
   macro avg       0.62      0.61      0.60      9832
weighted avg       0.62      0.61      0.60      9832
```

شکل ۱۱: نتایج **Roberta**



شکل ۱۲ : ماتریس در هم آمیختگی **Roberta**

در ادامه این بخش به پیاده سازی مدل با Freeze می پردازیم:

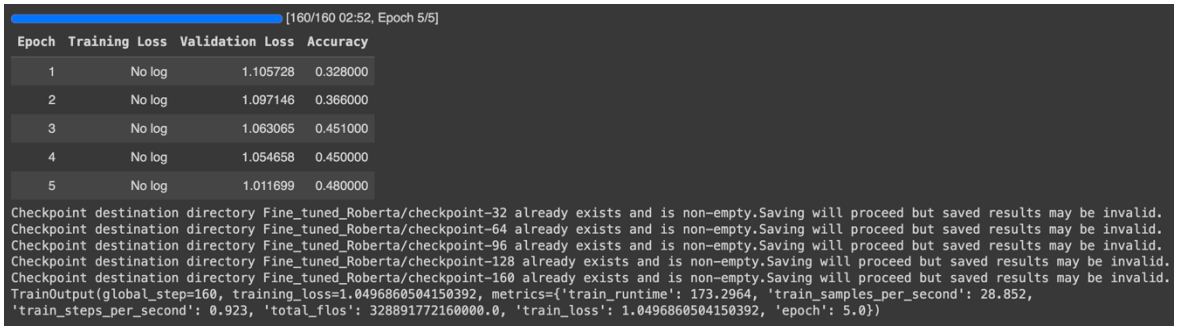
مدل RoBERTa را برای یک کار طبقه بندی دنباله ای با تمرکز خاص بر فریز کردن لایه های خاص در طول آموزش، مقداردهی اولیه و پیکربندی می کند. مدل «RobertaForSequenceClassification» از مدل «robertabase» از پیش آموزش‌دیده Hugging Face بارگیری شده است که برای یک کار طبقه‌بندی با سه برچسب خروجی («تعداد_برچسب‌ها=۳») طراحی شده است. هدف از انجماد لایه‌ها جلوگیری از به‌روزرسانی وزن آن لایه‌ها در طول آموزش است که می‌تواند برای تنظیم دقیق یک مدل در یک کار خاص با داده‌های محدود مفید باشد. در این قطعه، تمام پارامترها در لایه جاسازی مدل ثابت می‌شوند، همانطور که با تنظیم «requires_grad» روی «False» نشان داده می‌شود. به‌علاوه، شش لایه اول رمزگذار («تعداد_لایه‌های_به_فریز = ۶») نیز منجمد شده‌اند. یعنی در حین تمرین فقط وزنه های لایه های بالاتر و سر رده بندی به روز می شود. در نهایت، مدل برای آموزش کارآمد به یک دستگاه مشخص («DEVICE») منتقل می‌شود که می‌تواند یک CPU یا GPU باشد. این رویکرد انجماد جزئی یک مدل اغلب در سناریوهای یادگیری انتقال استفاده می‌شود تا از ویژگی‌های از پیش آموخته‌شده مدل پایه استفاده کند و در عین حال به مدل اجازه می‌دهد تا با ویژگی‌های کار جدید سازگار شود.

```
RobertaForSequenceClassification(
  (roberta): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(50265, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
      (token_type_embeddings): Embedding(1, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): RobertaEncoder(
      (layer): ModuleList(
        (0-11): 12 x RobertaLayer(
          (attention): RobertaAttention(
            (self): RobertaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): RobertaSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): RobertaIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): RobertaOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
```

شکل ۱۳: مدل **Roberta**

حال مشابه حالت قبل یک callback سفارشی را برای آموزش یک مدل یادگیری ماشینی با استفاده از کلاس Hugging Face Trainer تعریف می‌کند، به‌ویژه به منظور ارزیابی مدل در مجموعه داده‌های آموزشی در پایان هر دوره آموزشی. کلاس «CustomCallback» یک زیر کلاس از «TrainerCallback» است و برای انجام یک مرحله ارزیابی اضافی، روش «on_epoch_end» را لغو می‌کند. در این روش، اگر شرط «control.should_evaluate» درست باشد (که نشان دهنده پایان یک دوره است)، callback یک کپی از شی کنترل ایجاد می کند و سپس روش «ارزیابی» مربی را در مجموعه داده آموزشی فراخوانی می کند. ._trainer.train_dataset). این ارزیابی با «پیشوند_کلید_متریک» برچسب گذاری شده است، که آن را از ارزیابی استاندارد در مجموعه داده اعتبار سنجی متمایز می کند. خود شی «Trainer» با مدل، آرگومان‌های آموزشی، مجموعه داده‌های آموزش و ارزیابی، نشانه‌ساز و یک تابع محاسبات متریک سفارشی پیکربندی شده است. اگرچه

"CustomCallback" نمونه سازی شده است، اما نظر داده می شود و به مربی اضافه نمی شود، که نشان می دهد اختیاری است و در صورت نیاز می توان آن را فعال کرد. در نهایت، متد trainer.train() فرآیند آموزش را شروع می کند. این راه‌اندازی، با تماس سفارشی اختیاری، یک رژیم آموزشی جامع با توانایی نظارت بر عملکرد مدل نه تنها بر روی داده‌های اعتبارسنجی، بلکه بر روی داده‌های آموزشی برای تجزیه و تحلیل بیش‌ازحد فراهم می‌کند.

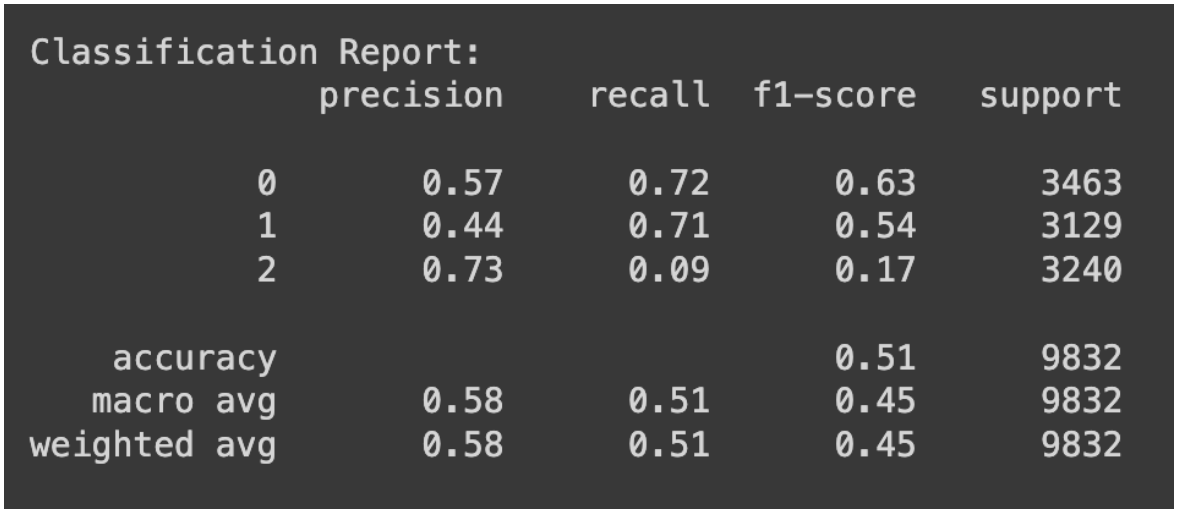

[160/160 02:52, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	1.105728	0.328000
2	No log	1.097146	0.366000
3	No log	1.063065	0.451000
4	No log	1.054658	0.450000
5	No log	1.011699	0.480000

Checkpoint destination directory Fine_tuned_Roberta/checkpoint-32 already exists and is non-empty.Saving will proceed but saved results may be invalid.
Checkpoint destination directory Fine_tuned_Roberta/checkpoint-64 already exists and is non-empty.Saving will proceed but saved results may be invalid.
Checkpoint destination directory Fine_tuned_Roberta/checkpoint-96 already exists and is non-empty.Saving will proceed but saved results may be invalid.
Checkpoint destination directory Fine_tuned_Roberta/checkpoint-128 already exists and is non-empty.Saving will proceed but saved results may be invalid.
Checkpoint destination directory Fine_tuned_Roberta/checkpoint-160 already exists and is non-empty.Saving will proceed but saved results may be invalid.
TrainOutput(global_step=160, training_loss=1.0496860504150392, metrics={'train_runtime': 173.2964, 'train_samples_per_second': 28.852,
'train_steps_per_second': 0.923, 'total_flos': 328891772160000.0, 'train_loss': 1.0496860504150392, 'epoch': 5.0})

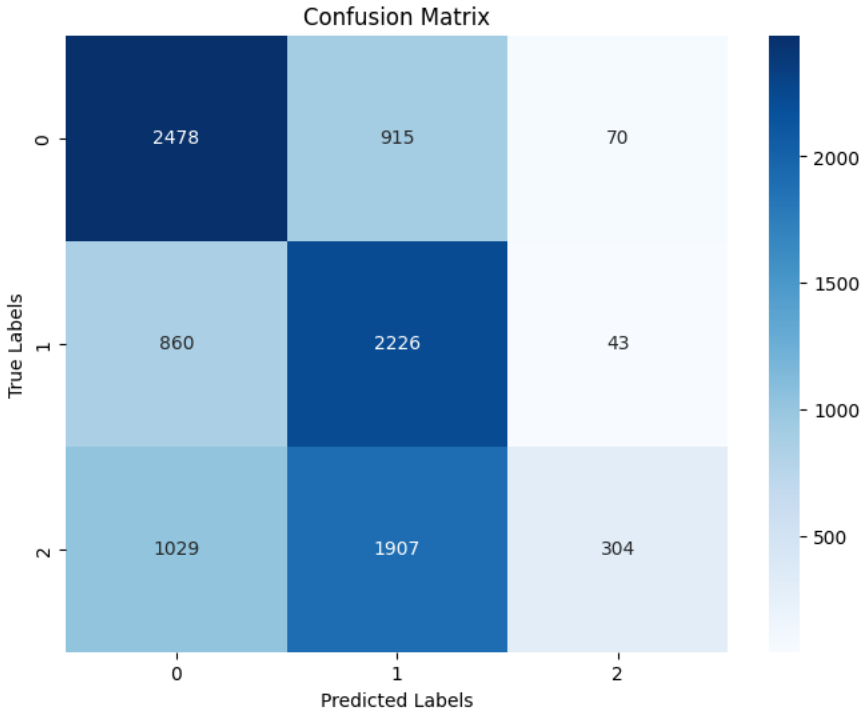
شکل ۱۴: پیاده سازی کلاس **Callback**

نتایج Freeze


Classification Report:

	precision	recall	f1-score	support
0	0.57	0.72	0.63	3463
1	0.44	0.71	0.54	3129
2	0.73	0.09	0.17	3240
accuracy			0.51	9832
macro avg	0.58	0.51	0.45	9832
weighted avg	0.58	0.51	0.45	9832

شکل ۱۵: نتایج **Freeze**



شکل ۱۶:ماتریس در هم آمیختگی **Freeze**

حال به توضیح بخش LoRA می‌پردازیم:

ابتدا با استفاده از کتابخانه‌ی peft بخش‌های LoRA را به کد اضافه می‌کنیم، این مدل LoRA برای وظیفه دسته‌بندی سری هست و می‌توانیم مقدار rank آن را مشخص کنیم که در تعداد پارامترهای یادگیری تاثییر دارید.

سپس LoRA را به مدل خود اضافه می‌کنیم، تعداد پارامترهای مدل قبل و بعد از اضافه کردن LoRA به شکل زیر می‌باشد:

```
trainable params: 355362819 || all params: 355362819 || trainable%: 100.0
trainable params: 4198403 || all params: 359561222 || trainable%: 1.167646215197255
```

شکل ۱۷ تعداد پارامترهایقابل آموزش مدل قبل و بعد از **LoRA**

حال مدل را finetune می‌کنیم و نتایج آن به صورت زیر می‌باشد:

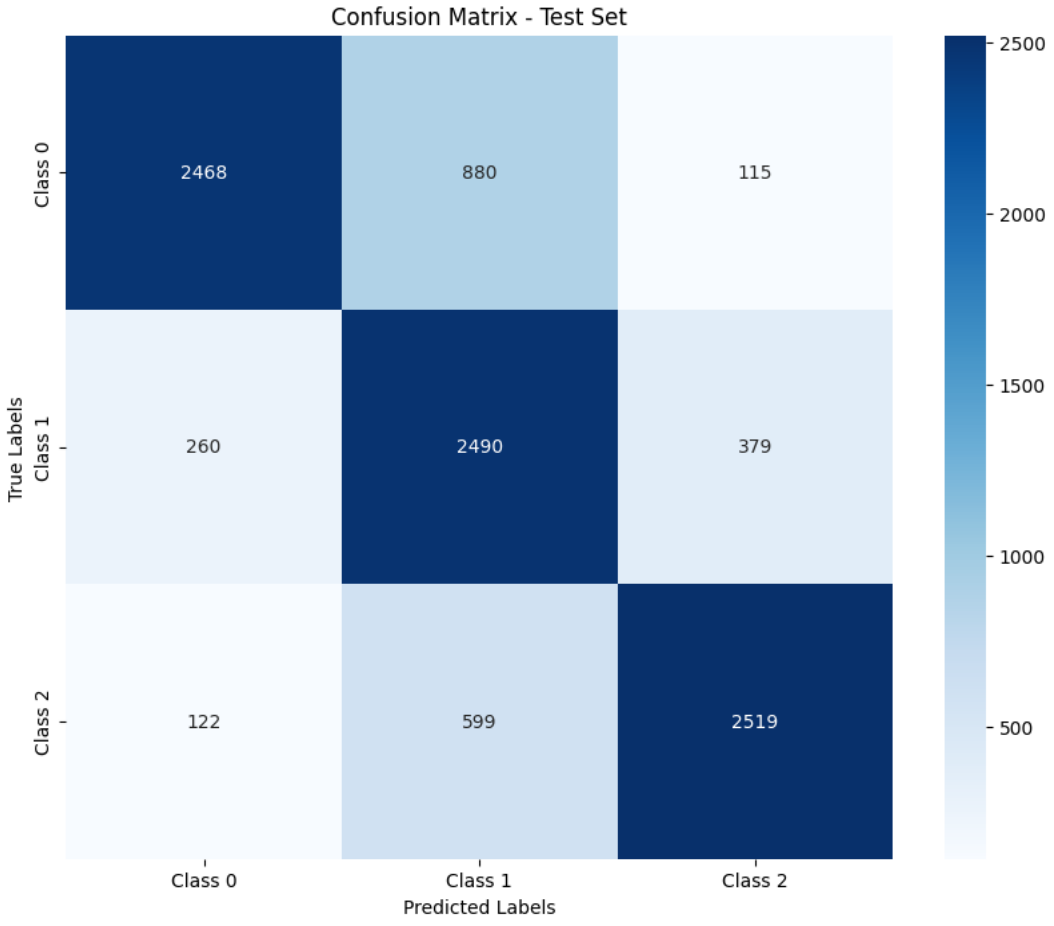
```
Classification Report - Test Set:
              precision    recall  f1-score   support

     Class 0       0.87      0.71      0.78      3463
     Class 1       0.63      0.80      0.70      3129
     Class 2       0.84      0.78      0.81      3240

    accuracy                           0.76      9832
   macro avg       0.78      0.76      0.76      9832
weighted avg       0.78      0.76      0.76      9832

Test Accuracy: 0.7605
```

شکل ۱۸: نتایج **LoRA**



شکل ۱۹:ماتریس در هم آمیختگی **LoRA**

مقایسه کلی

مفاهیم "LoRA" (انطباق با رتبه پایین) و لایه های "انجماد" در یک شبکه عصبی، به ویژه در زمینه مدل های ترانسفورماتور مانند BERT یا RoBERTa، نشان دهنده دو روش متمایز برای انطباق و تنظیم دقیق مدل های از پیش آموزش دیده برای خاص است. وظایف یا مجموعه داده ها

LoRA (انطباق با رتبه پایین)

مفهوم: LoRA شامل اصلاح یک مدل از پیش آموزش‌دیده با قرار دادن ماتریس‌های با رتبه پایین در لایه‌های آن است که در طول تنظیم دقیق یاد می‌شوند. به‌جای به‌روزرسانی وزن‌های اصلی مدل، LoRA ماتریس‌های کوچک و قابل آموزش را معرفی می‌کند که وزن‌های موجود را به روشی با رتبه پایین تطبیق می‌دهند. این رویکرد با هدف گرفتن سازگاری های خاص کار با پارامترهای آموزش پذیر کمتر است.

مزایا: LoRA امکان تطبیق کارآمد یک مدل را با افزایش نسبتاً کمی در تعداد پارامترها فراهم می کند. می‌تواند با ثبت نکات ظریف و ظریف و بدون نیاز به آموزش مجدد کل مدل، منجر به عملکرد بهتر در وظایف خاص شود.

سناریوهای استفاده: LoRA به ویژه در سناریوهایی مفید است که در آن شخص می‌خواهد یک مدل بزرگ از پیش آموزش دیده را با یک کار خاص بدون هزینه محاسباتی آموزش پارامترهای زیادی تطبیق دهد. همچنین زمانی مفید است که داده‌های مربوط به کار محدود باشد و تطبیق بیش از حد یک نگرانی باشد.

لایه های انجماد:

مفهوم: انجماد لایه‌ها در یک شبکه عصبی به معنای تنظیم وزن لایه‌های خاص غیرقابل آموزش است. این اغلب با مدل های از پیش آموزش دیده انجام می شود که در آن تنها چند لایه بالایی (یا فقط لایه طبقه بندی) برای یک کار خاص به خوبی تنظیم شده اند. لایه‌های پایین‌تر، که ویژگی‌های عمومی بیشتری را ثبت می‌کنند، منجمد می‌شوند.

مزایا: انجماد تعداد پارامترهایی را که نیاز به آموزش دارند کاهش می‌دهد، هزینه‌های محاسباتی و خطر بیش از حد برازش را کاهش می‌دهد. این به مدل اجازه می دهد تا دانش عمومی آموخته شده در طول آموزش را حفظ کند و فقط ویژگی های سطح بالاتر را برای کار خاص تطبیق دهد.

سناریوهای استفاده: فریز کردن معمولاً در یادگیری انتقال استفاده می‌شود، به‌ویژه زمانی که مدل‌های بزرگ از پیش آموزش‌دیده را روی یک مجموعه داده کوچک تنظیم می‌کنید. زمانی موثر است که مدل پایه از قبل روی یک مجموعه داده بزرگ و متنوع به خوبی آموزش داده شده باشد و تنها به انطباق های جزئی برای کار مورد نیاز نیاز است.

مقایسه:

هدف: هر دو روش با هدف تطبیق مدل های از پیش آموزش دیده با وظایف خاص به طور موثر انجام می شوند، اما آنها این کار را به روش های مختلف انجام می دهند. LoRA حداقل ظرفیت سازگاری را اضافه می کند، در حالی که انجماد سازگاری را به لایه های خاصی محدود می کند.

آموزش پارامتر: LoRA شامل آموزش ماتریس‌های با رتبه پایین اضافی است، در حالی که فریز کردن شامل آموزش تنها پارامترهای لایه‌های منجمد نشده است.

انعطاف پذیری: LoRA تعادلی بین حفظ دانش از پیش آموزش دیده و انطباق با وظایف جدید ارائه می دهد، در حالی که انجماد بیشتر در مورد استفاده از دانش موجود با حداقل سازگاری است.

کارایی محاسباتی: هر دو روش در مقایسه با تنظیم دقیق مدل کامل از نظر محاسباتی کارآمد هستند، اما کارایی خاص به تعداد پارامترهای ثابت شده یا رتبه ماتریس ها در LoRA بستگی دارد.

زمان محاسباتی: از آنجا که با استفاده از LoRa فقط تعداد کمی از پارامترهای مدل آموزش میبنیند با استفاده از لورا توقع داریم در مدت زمان کمتری آموزش به اتمام برسد.

با توجه به نتایج مشخص است که مدل اصلی ۲۰ دقیقه در ۵ اپاک آموزش دیده است ولی مدل لورا فقط ۱۰ دقیقه طول کشیده است و نتایج به اندازه کافی مناسبی گرفته است.

مدل Lora با وجود مدت زمان کمتر آموزش و همچنین تعداد پارامتر کمتر، توانسته دقتی نزدیک دقت مدل اصلی (در دفعات مختلف ممکن بود تا ۵ درصد بهتر نیز بشود) گرفته است.

دقت مدل اصلی: ۷۵

دقت LoRA: ۷۳

یعنی مشخصا Lora یک روش تنظیم دقیق بسیار خوب می‌باشد که با وجود مقداری کم شدن دقت می‌تواند تا حد زیادی عملکردی در حد تنظیم دقیق کل مدل داشته باشد و ما با منابع و زمان کمتر به این عملکرد می‌رسیم.

به طور خلاصه، LoRA و انجماد هر دو استراتژی‌های مؤثری برای تنظیم دقیق مدل‌های از پیش آموزش‌دیده هستند، اما نیازها و محدودیت‌های محاسباتی کمی متفاوت را برآورده می‌کنند. انتخاب بین آنها به نیازهای خاص کار، اندازه مجموعه داده و منابع محاسباتی موجود بستگی دارد.

۲-۴. چرا LoRA ؟

روش Lora به شدت efficient تر از finetune کردن کل مدل است و نتیجه تقریبا یکسانی با آن می‌گیرد، این روش با اضافه کردن تعداد کمی پارامتر به مدل در مدت زمان کمتر و با دقت و خطای مناسب همان نتایج مدل را به ما خواهد داد.

Finetune کردن کل مدل به شدت هزینه بر است و باید توانایی و داده‌های زیاد برای این کار داشته باشیم تا مدل مدل هم وظیفه جدید را یاد بگیرد و هم وظیفه قبلی را فراموش نکند، همینطور finetune کردن بخشی از مدل راه حلی منطقی تر و بهینه‌تر می‌باشد ولی با این حال بهترین نتایج را لزوما نخواهد داد، روش Lora به طور کلی برتری‌های زیر را دارد.

- LoRA نسبت به تنظیم دقیق (fine-tuning) بسیار سریعتر است، زیرا فقط یک تعداد کوچک از پارامترها را به‌روزرسانی می‌کند به جای تمام پارامترهای مدل زبان بزرگ
- LoRA در مصرف حافظه و فضای ذخیره‌سازی به مراتب کارآمدتر از تنظیم دقیق است، زیرا تنها نیاز به ذخیره ماتریس‌های تجزیه رتبه برای هر وظیفه دارد به جای کل مدل تنظیم دقیق شده
- LoRA در کیفیت مدل در مقایسه با تنظیم دقیق بر روی مدل‌های زبان بزرگ مختلف مانند RoBERTa، DeBERTa، GPT-۲ و GPT-۳ بهتر یا مساوی عمل می‌کند، با این‌که تعداد کمتری پارامتر قابل آموزش دارد
- LoRA هیچ تاخیر استنتاجی ایجاد نمی‌کند، زیرا ماتریس‌های قابل آموزش می‌توانند در هنگام استقرار با وزن‌های یخ‌زده ترکیب شوند، به عنوان مقابل مانند آداپتورها که لایه‌های اضافی به مدل اضافه می‌کنند.

- LoRA یک روش ارتجاعی است که می‌تواند با تکنیک‌های دیگر تنظیم دقیق مانند تنظیم پیشوندی (prefix-tuning) ترکیب شود.

حال اگر قصد finetune کردن مدل برای چند وظیفه را داشته باشیم راه‌های زیر را داریم:

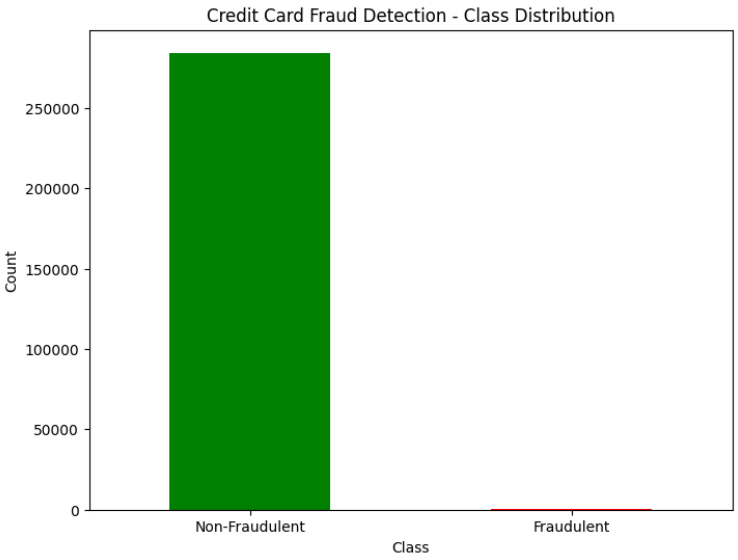
- مجموعه‌داده‌ای از این وظایف جدید و وظایفی که مدل قبلا به آن آگاه بوده جمع کنیم و مدل را برای این هر دو آن‌ها آموزش بدهیم تا وظایف جدید را یاد بگیرد و وظایف قبلی را فراموش نکند.
- اگر مدل به اندازه کافی بزرگ باشد و قبلا با داده‌های کافی آموزش دیده باشد، می‌توانیم با استفاده از روش‌های few shot این مساله را حل کنیم(که fine tuning نیست)
- می‌توانیم Lora را برای وظیفه اول آموزش بدهیم و آپدیت‌های آن را با وزن‌ها جمع کنیم، سپس Lora را برای وظیفه دوم نیز آموزش بدهیم و آپدیت‌ها را با وزن‌ها جمع کنیم. به طور کلی برای وظایف مختلف می‌توانیم آپدیت‌های lora را با هم جمع کرده و به وزن‌های مدل اضافه کنیم.

۳. تشخیص تقلب

۳-۱. آشنایی با دیتاست

این دیتاست مربوط به تراکنش‌های بانکی می‌باشد که آیا دارای کلاه‌برداری بوده‌اند یا خیر. این مجموعه‌داده شامل تراکنش‌هایی با ویژگی‌های مختلفی است که از V۱ تا V۲۸ برچسب‌گذاری شده‌اند، که احتمالاً نتیجه تغییر تجزیه و تحلیل مؤلفه اصلی (PCA) به دلایل حفظ حریم خصوصی، و همچنین «زمان» و «مقدار» است. همچنین یک ستون 'Class' وجود دارد که متغیر هدف است و نشان می دهد که آیا تراکنش تقلبی است (۱) یا خیر (۰).

هیستوگرام توزیع کلاس در مجموعه داده، عدم تعادل قابل توجهی را بین دو کلاس نشان می دهد: غیر متقلبان (کلاس ۰) و متقلب (کلاس ۱). تعداد تراکنش های غیر متقلبانه بسیار بیشتر از معاملات متقلبانه است.



شکل ۲۰ هیستوگرام داده‌ها

این عدم تعادل یک چالش مهم برای مدل های یادگیری ماشینی است. اگر مدلی بر روی این مجموعه داده همانطور که هست آموزش داده شود، ممکن است نسبت به پیش‌بینی تراکنش‌ها به‌عنوان غیرجعلی تعصب داشته باشد، زیرا آن کلاس بر مجموعه داده تسلط دارد. در نتیجه، این مدل ممکن است در تشخیص تراکنش‌های جعلی، که کلاس اقلیت هستند، اما از منظر کشف تقلب، علاقه بالاتری دارند، عملکرد ضعیفی داشته باشد.

دقت مدل ممکن است بالا به نظر برسد، زیرا می‌تواند طبقه اکثریت را به درستی پیش‌بینی کند، اما توانایی آن در تشخیص طبقه اقلیت - که برای سیستم تشخیص تقلب بسیار مهم است - احتمالاً ناکافی است. به عبارت ساده تر دقت کلی مدل ممکن است ۹۹ درصد باشد اما روی داده‌های کلاس اکثریت دقت ۱۰۰ داشته باشیم و روی داده‌های کلاس اقلیت دقت ۰ که به این معنی است همه را کلاس اکثریت پیش‌بینی میکند و عملا مدل چیزی یاد نگرفته است. این یک مشکل رایج در مجموعه‌های داده با کلاس‌های بسیار نامتعادل است و به استراتژی‌های خاصی برای رسیدگی نیاز دارد، مانند نمونه‌گیری مجدد از مجموعه داده، استفاده از تکنیک‌های تشخیص ناهنجاری، یا اعمال معیارهای مختلف برای ارزیابی مدل مانند فراخوان دقیق AUC-PR، F۱-. امتیاز، یا استفاده از الگوریتم هایی که حساسیت کمتری نسبت به عدم تعادل کلاس دارند.

۳-۲. پیاده سازی معماری مقاله

در این بخش از کتابخانه TensorFlow و مدلی از نوع Sequential برای ساخت یک شبکه عصبی کانولوشنال ۱D استفاده می‌کند. این شبکه برای یادگیری از داده‌های ورودی با سه بعد (X_train.shape[۱], X_train.shape[۲]) استفاده می‌شود که در اینجا X_train به عنوان ورودی شبکه تعیین شده است.

در زیر توضیح داده‌ام که هر لایه از مدل چه کارهایی انجام می‌دهد:

۱. ۱ Conv۱D Layer: این لایه یک لایه کانولوشنال ۱D با ۳۲ فیلتر، هر کدام با یک کرنل به طول ۲ و تابع فعال‌سازی ReLU دارد. این لایه برای استخراج ویژگی‌های مختلف از داده‌های ورودی استفاده می‌شود.

۱. ۲ BatchNormalization: این لایه برای استانداردسازی میانگین و واریانس خروجی‌های لایه قبلی (۱ Conv۱D Layer) استفاده می‌شود، که به بهبود آموزش مدل کمک می‌کند.

۱. ۳ MaxPool۱D: این لایه از عملگر MaxPooling برای کاهش ابعاد خروجی لایه قبلی استفاده می‌کند. این باعث کاهش تعداد ویژگی‌ها و افزایش مقیاس‌پذیری شبکه می‌شود.

۱. ۴ Dropout: این لایه برای جلوگیری از بر Overfitting در مدل استفاده می‌شود. با احتمال ۰.۲، تعدادی از واحدهای خروجی لایه قبلی را به صورت تصادفی خاموش می‌کند.

۲. ۵ Conv۱D Layer: این لایه یک لایه کانولوشنال دیگر با ۶۴ فیلتر و تابع فعال‌سازی ReLU است.

۶. ۲ BatchNormalization: مانند ۱ BatchNormalization، این لایه برای استانداردسازی میانگین و واریانس خروجی‌های لایه قبلی (۲ Conv۱D Layer) استفاده می‌شود.

۷. ۲ MaxPool۱D: مانند ۱ MaxPool۱D، این لایه از عملگر MaxPooling برای کاهش ابعاد خروجی لایه قبلی استفاده می‌شود.

۸. ۲ Dropout: مانند ۱ Dropout، این لایه با احتمال ۰.۵ برای جلوگیری از Overfitting از طریق خاموش کردن واحدهای تصادفی استفاده می‌شود.

۹. Flatten: این لایه به داده‌های خروجی لایه قبلی اعمال می‌شود و آنها را به یک بردار یک بعدی تبدیل می‌کند.

۱۰. ۱ Dense: این لایه با ۶۴ واحد و تابع فعال‌سازی ReLU برای اعمال یادگیری ژرف بر روی ویژگی‌های استخراج شده توسط لایه‌های کانولوشنال استفاده می‌شود.

۱۱. ۳ Dropout: مانند ۱ Dropout و ۲، این لایه با احتمال ۰.۵ برای جلوگیری از Overfitting استفاده می‌شود.

۱۲. ۲ Dense: این لایه دیگر با ۶۴ واحد و تابع فعال‌سازی ReLU استفاده می‌شود.

۱۳. ۳ Dense: این لایه نهایی با یک واحد و تابع فعال‌سازی sigmoid برای انجام مسئله دسته‌بندی دودویی استفاده می‌شود.

این مدل به صورت یک لایه‌ی کانولوشنال و دو لایه‌ی میانی کاملاً متصل (Dense) به علاوه‌ی لایه‌های BatchNormalization، MaxPooling و Dropout برای ایجاد یک معماری عصبی کامل در

شبکه‌های عصبی عمیق استفاده شده است. این مدل به منظور یادگیری از داده‌های دوره‌ای D۱ مورد استفاده قرار می‌گیرد.

تصویر زیر تمام مدل را نشان می‌دهد که به صورت کامل از مقاله پیاده‌سازی شده است:

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)

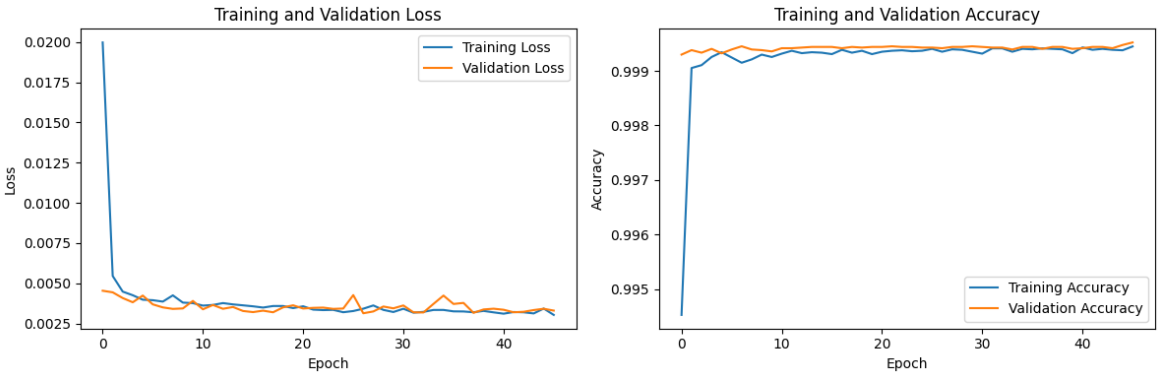
شکل ۲۱ معماری مدل

همچنین هایپرپارامترهای مدل به شرح زیر می‌باشند:

Hyperparameter	Value
epochs	46
optimizer	Adam
learning_rate	0.0001
loss	binary_crossentropy
metrics	accuracy

شکل ۲۲ هایپر پارامترهای مدل

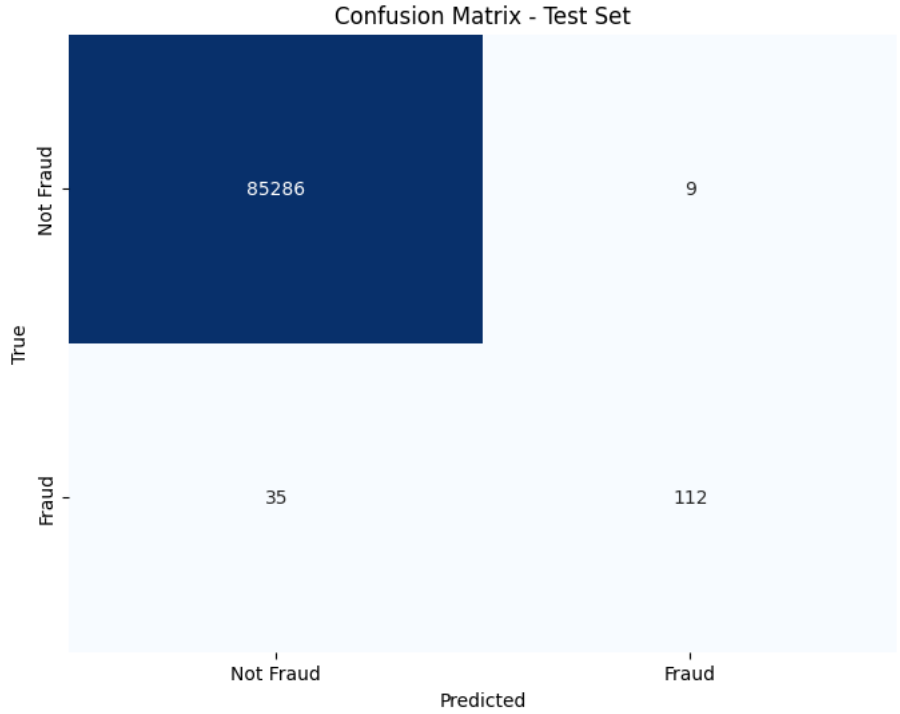
حال مدل را برای ۴۶ تا ایپاک آموزش می‌دهیم که نتایج به صورت زیر شد:



شکل ۲۳: نتایج پیاده سازی مدل

با توجه به نمودار loss به نظر می‌آید مدل overfit نشده است و عملکرد روی داده‌های آموزش و validation به اندازه کافی خوب هست، با این حال دلیل این موضوع می‌تواند این باشد که داده‌های validation نیز شامل داده‌های کلاس اکثریت هست و مدل روی آن کلاس خوب عمل کرده، به طور کلی دقت برای نظر دادن راجع به این نوع از داده‌ها معیار مناسبی نیست.

عملکرد مدل با استفاده از ماتریس درهمریختگی به شکل زیر می‌باشد:



شکل ۲۴: ماتریس در هم آمیختگی مدل

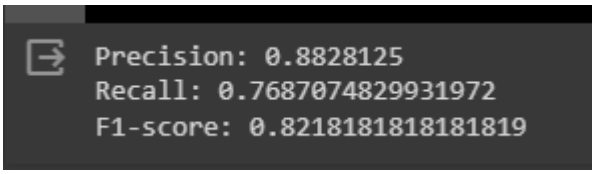
```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85295
           1       0.93      0.76      0.84       147

    accuracy                           1.00     85442
   macro avg       0.96      0.88      0.92     85442
weighted avg       1.00      1.00      1.00     85442
```

شکل ۲۵: نتایج مدل

مشخص است که دقت مدل به صورت کلی خوب به نظر می‌رسد اما روی کلاس ۱ که داده‌های اقلیت هست عملکرد ضعیفی دارد. این موضوع را با استفاده از معیارهای دیگر می‌توان فهمید:



شکل ۲۶ معیارهایی صحت و **recall**

۳-۳. نمونه برداری

۱. توضیح روش ADASYN و مقایسه با روش‌های دیگر نمونه‌برداری

ADASYN یک روش نمونه‌برداری هوشمندانه است که برای مقابله با مشکلات عدم تعادل در داده‌های آموزشی در مسائل طبقه‌بندی ماشینی استفاده می‌شود. در مسائلی که تعداد نمونه‌های هر کلاس در داده‌ها به طور نسبی ناهمسان است، ممکن است مدل طبقه‌بندی با مشکل روبرو شود و به کلاس‌های کم‌نمونه‌ای توجه نکند. ADASYN با تولید نمونه‌های مصنوعی برای کلاس‌های کم‌نمونه‌ای و تعادل تعداد نمونه‌ها، به بهبود عملکرد مدل کمک می‌کند.

توضیح عملکرد ADASYN:

۱. تعیین وزن برای هر نمونه: این روش وزن مختص به هر نمونه را تعیین می‌کند. نمونه‌های کمیتی که برای تعادل در داده‌ها نیاز دارند، وزن بیشتری دریافت می‌کنند.

۲. محاسبه میزان نیاز به نمونه جدید: بر اساس اختلاف تعداد نمونه‌ها بین کلاس‌ها، میزان نیاز به نمونه جدید برای هر کلاس محاسبه می‌شود.

۳. تولید نمونه جدید: برای هر نمونه کمیتی که بر اساس محاسبات به آن نیاز است، نمونه‌های جدید تولید می‌شوند.

۴. افزودن نمونه‌های تولید شده به داده‌های آموزشی: نمونه‌های تولید شده به داده‌های آموزشی افزوده می‌شوند تا تعداد نمونه‌ها برای هر کلاس بهبود یابد.

مزایا:

- بهبود تعادل کلاس‌ها در داده‌های آموزشی.

- افزایش دقت مدل در کلاس‌های کم‌نمونه‌ای.

معایب:

- افزایش حجم داده‌های آموزشی به دلیل تولید نمونه‌های مصنوعی.

- حساسیت به نویز در داده‌ها.

- زمان‌بر بودن فرآیند تولید نمونه‌های مصنوعی.

- برای نمونه های اقلیت که به صورت پراکنده توزیع شده اند، هر محله ممکن است فقط دارای ۱ نمونه اقلیت باشد.

- دقت ADASYN ممکن است به دلیل ماهیت سازگاری آسیب ببیند.

نسبت به روش‌های دیگر نمونه‌برداری، ADASYN به دلیل توجه به نمونه‌های کم‌نمونه‌ای بهبود قابل توجهی می‌دهد. اما هر روشی مزایا و معایب خود را دارد و انتخاب بهترین روش بستگی به مشخصات خاص مسئله و داده‌های آموزشی دارد.

۳. روش پیاده‌سازی:

مراحل پیاده‌سازی به شرح زیر می‌باشد:

مرحله ۱

نسبت اقلیت به اکثریت را با استفاده از مثال‌های زیر محاسبه کنید:

$$d = \frac{m_s}{m_l}$$

که در آن m_s و m_l به ترتیب # نمونه های کلاس اقلیت و اکثریت هستند. اگر d کمتر از یک آستانه مشخص است، الگوریتم را مقداردهی اولیه کنید.

گام ۲

تعداد کل داده های اقلیت مصنوعی برای تولید را محاسبه کنید.

$$G = (m_l - m_s)\beta$$

شکل ۲۷ تعداد کل دادههای اقلیت

در اینجا، G تعداد کل داده های اقلیت برای تولید است. ß نسبت داده های اقلیت: اکثریت مورد نظر پس از ADASYN است. ۱= ß به معنای مجموعه داده های کاملاً متعادل پس از ADASYN است.

مرحله ۳

k-نزدیکترین همسایگان هر یک از نمونه های اقلیت را پیدا کنید و مقدار r_i را محاسبه کنید. پس از این مرحله، هر نمونه اقلیت باید با یک محله متفاوت مرتبط شود.

$$r_i = \frac{\#majority}{k}$$

شکل ۲۸ محاسبه **r**

مقدار r_i نشان دهنده تسلط طبقه اکثریت در هر محله خاص است. محلههای r_i بالاتر حاوی نمونههای کلاس اکثریت بیشتری هستند و یادگیری آنها دشوارتر است. برای تجسم این مرحله به زیر مراجعه کنید. در مثال، ۵ = K (به دنبال ۵ همسایه نزدیک می گردد).

مرحله ۴

مقادیر r_i را عادی کنید تا مجموع همه مقادیر r_i برابر با ۱ باشد.

$$\hat{r}_i = \frac{r_i}{\sum r_i}$$

$$\sum \hat{r}_i = 1$$

شکل ۲۹ محاسبه شعاع

این مرحله عمدتاً مقدمه ای برای آسان کردن مرحله ۵ است.

مرحله ۵

مقدار نمونه های مصنوعی برای تولید در هر محله را محاسبه کنید.

$$G_i = G\hat{r}_i$$

شکل ۳۰ تعداد نمونه‌ی تصادفی

از آنجایی که r_i برای محله‌هایی که نمونه‌های طبقه اکثریت بر آنها غالب است، بالاتر است، نمونه‌های کلاس اقلیت مصنوعی بیشتری برای آن محله‌ها تولید می‌شود. از این رو، این به الگوریتم ADASYN ماهیت تطبیقی آن می دهد. داده های بیشتری برای محله های "سخت تر برای یادگیری" تولید می شود.

مرحله ۶

داده های G_i را برای هر مرحله ایجاد کنید. ابتدا مثال اقلیت را برای محله، x_i در نظر بگیرید. سپس، به طور تصادفی نمونه اقلیت دیگری را در آن محله، xz_i انتخاب کنید. مثال مصنوعی جدید را می توان با استفاده از:

$$s_i = x_i + (x_{zi} - x_i)\lambda$$

شکل ۳۱ تولید نمونه تصادفی

در معادله بالا، λ یک عدد تصادفی بین ۰-۱ است، Si نمونه مصنوعی جدید است، Xi و XZi دو نمونه اقلیت در یک همسایگی هستند. تصویری از این مرحله در زیر ارائه شده است. به طور شهودی، نمونه های مصنوعی بر اساس ترکیب خطی Xi و XZi ایجاد می شوند.

در کد یک کلاس به نام ADASYN` را تعریف می‌کنیم که الگوریتم ADASYN را برای افزایش نمونه‌ها در زمینه مسائل طبقه‌بندی نامتوازن پیاده‌سازی می‌کند. در زیر توضیحی از قسمت‌های اصلی کد آمده است:

در پیاده سازی این بخش داریم:

یک کلاس پایتون به نام ADASYN(Adaptive Synthetic Sampling) را تعریف می کند که مخفف Adaptive Synthetic Sampling است. الگوریتم ADASYN یک تکنیک نمونه برداری بیش از حد است که برای مقابله با عدم تعادل کلاس در مجموعه داده های یادگیری ماشین استفاده می شود. بر روی تولید نمونه های مصنوعی برای طبقه اقلیت بر اساس نقاط داده ای که یادگیری آنها دشوار است تمرکز دارد.

در اینجا جزئیاتی از اجزای اصلی کلاس ADASYN آورده شده است:

سازنده (__init__): این متد شی ADASYN را با چندین پارامتر مقداردهی اولیه می کند:

نسبت: نسبت مورد نظر تعداد نمونه در کلاس اقلیت پس از نمونه گیری مجدد به تعداد نمونه در کلاس اکثریت.

imb_threshold: آستانه ای برای تعیین اینکه آیا یک کلاس نامتعادل در نظر گرفته می شود.

k: تعداد نزدیک‌ترین همسایه‌ها برای استفاده در هنگام تولید نمونه‌های مصنوعی.

random_state: دانه ای برای مولد اعداد تصادفی که امکان تکرارپذیری را فراهم می کند.

verbose: پرچمی برای فعال یا غیرفعال کردن چاپ اطلاعات در حین پردازش.

fit: این روش ویژگی های X و هدف y را می پذیرد و اقدامات زیر را انجام می دهد:

آرایه های ورودی را بررسی و قالب بندی می کند.

یک مولد اعداد تصادفی را با دانه ارائه شده راه اندازی می کند.

طبقه اکثریت را شناسایی می کند و آمار کلاس را محاسبه می کند.

transform (تبدیل): این روش به طور کامل اجرا نمی شود زیرا X و y را نمی پذیرد و چیزی را بر نمی گرداند. به نظر می رسد در نظر گرفته شده است که نمونه برداری بیش از حد در مجموعه داده اعمال شود، اما ناقص است.

fit_transform: این روش مراحل تناسب و تبدیل را به یک مرحله ترکیب می کند. ابتدا نمونه ADASYN را با داده ها مطابقت می دهد و سپس نمونه برداری بیش از حد را انجام می دهد. سپس داده‌های بیش‌نمونه‌شده با داده‌های اصلی الحاق می‌شوند تا یک مجموعه داده جدید با توزیع کلاس متعادل ایجاد شود.

Generate_samples: این روش نمونه های مصنوعی را برای یک کلاس اقلیت معین تولید می کند. برای هر نقطه داده کلاس اقلیت، k نزدیکترین همسایه را پیدا می کند و با درون یابی بین نقطه داده و همسایگان آن، نمونه های جدیدی تولید می کند.

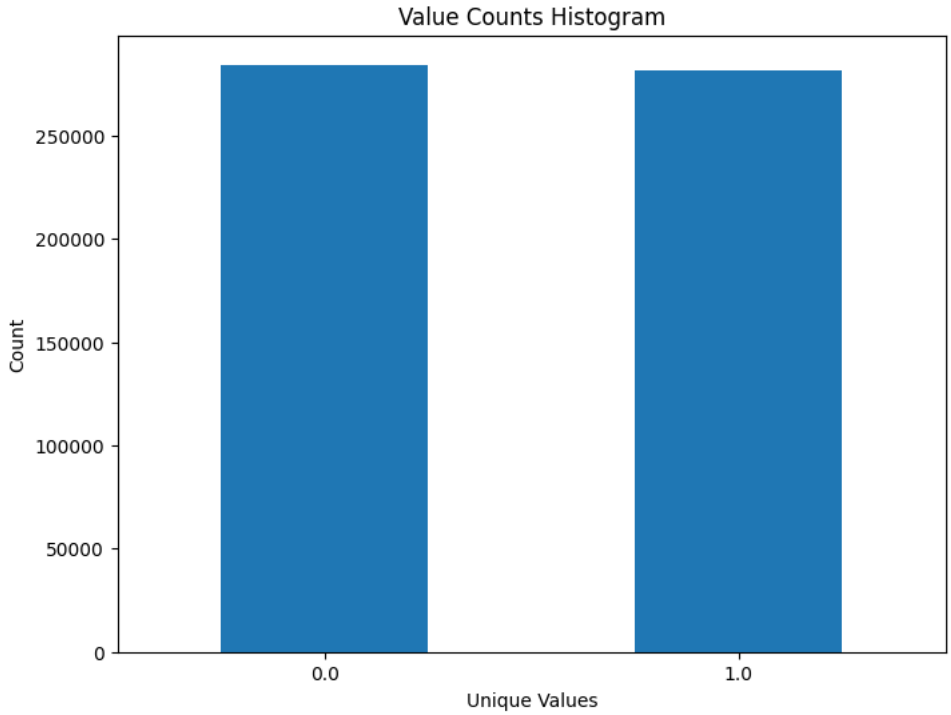
oversample: این روش نمونه برداری واقعی را انجام می دهد.

۴. زمان انجام نمونه‌برداری

نمونه برداری معمولاً باید قبل از تقسیم کردن داده‌ها به دسته های آموزش و تست انجام شود. این کار مهم است زیرا انجام نمونه‌برداری بعد از تقسیم‌بندی می‌تواند منجر به نشت اطلاعات بین دسته‌های آموزش و تست شود و در نتیجه ارزیابی نادرستی از عملکرد مدل را به دنبال داشته باشد.

۴. تحلیل داده‌های نمونه‌برداری شده

پس از انجام نمونه‌برداری، می‌توانید هیستوگرام‌هایی را برای مقایسه توزیع کلاس‌ها قبل و بعد از نمونه‌برداری ترسیم به صورت زیر می‌باشد.



شکل ۳۲: هیستوگرام بعد از نمونه برداری

۳-۴. آموزش مدل با داده های جدید

طبق پیاده سازی مقاله

حال برای پیاده سازی مدل عصبی با استفاده از TensorFlow و Keras ایجاد می‌کند که برای یادگیری از داده‌های دوره‌ای D۱ طراحی شده است. این مدل شامل یک سری از لایه‌های مختلف است. دستورات مربوط به هر لایه به شرح زیر است:

۱. ۱ Conv۱D Layer: این لایه یک لایه کانولوشنال D۱ با ۳۲ فیلتر، هر کدام با یک کرنل به طول ۲ و تابع فعال‌سازی ReLU دارد. این لایه برای استخراج ویژگی‌های مختلف از داده‌های ورودی با ساختار (X_train.shape[۱], X_train.shape[۲]) استفاده می‌شود.

۲. ۱ BatchNormalization: این لایه برای استانداردسازی میانگین و واریانس خروجی‌های لایه قبلی (۱ Conv۱D Layer) استفاده می‌شود. این کمک می‌کند تا آموزش مدل بهبود یابد و از مشکلات مربوط به مقیاس‌پذیری و واریانس داده جلوگیری شود.

۳. ۱ MaxPool۱D: این لایه از عملگر MaxPooling برای کاهش ابعاد خروجی لایه قبلی استفاده می‌کند. این باعث کاهش تعداد ویژگی‌ها و افزایش مقیاس‌پذیری شبکه می‌شود.

۴. ۱ Dropout: این لایه برای جلوگیری از اورفیتینگ (Overfitting) در مدل استفاده می‌شود. با احتمال ۰.۲، تعدادی از واحدهای خروجی لایه قبلی را به صورت تصادفی خاموش می‌کند.

۵. ۲ Conv۱D Layer: این لایه یک لایه کانولوشنال دیگر با ۶۴ فیلتر و کرنل به طول ۲ و تابع فعال‌سازی ReLU است.

۶. ۲ BatchNormalization: مانند ۱ BatchNormalization، این لایه برای استانداردسازی میانگین و واریانس خروجی‌های لایه قبلی (۲ Conv۱D Layer) استفاده می‌شود.

۷. ۲ MaxPool۱D: مانند ۱ MaxPool۱D، این لایه از عملگر MaxPooling برای کاهش ابعاد خروجی لایه قبلی استفاده می‌کند.

۸. ۲ Dropout: این لایه برای جلوگیری از اورفیتینگ با احتمال ۰.۵ استفاده می‌شود. این باعث می‌شود تا تعداد بیشتری از واحدها در طی آموزش خاموش شوند و مدل بهبود پذیرشود.

۹. Flatten: این لایه به داده‌های خروجی لایه قبلی (۲ MaxPool۱D) اعمال می‌شود و آنها را به یک بردار یک بعدی تبدیل می‌کند.

۱۰. ۱ Dense: این لایه با ۶۴ واحد و تابع فعال‌سازی ReLU برای اعمال یادگیری ژرف بر روی ویژگی‌های استخراج شده توسط لایه‌های قبلی استفاده می‌شود.

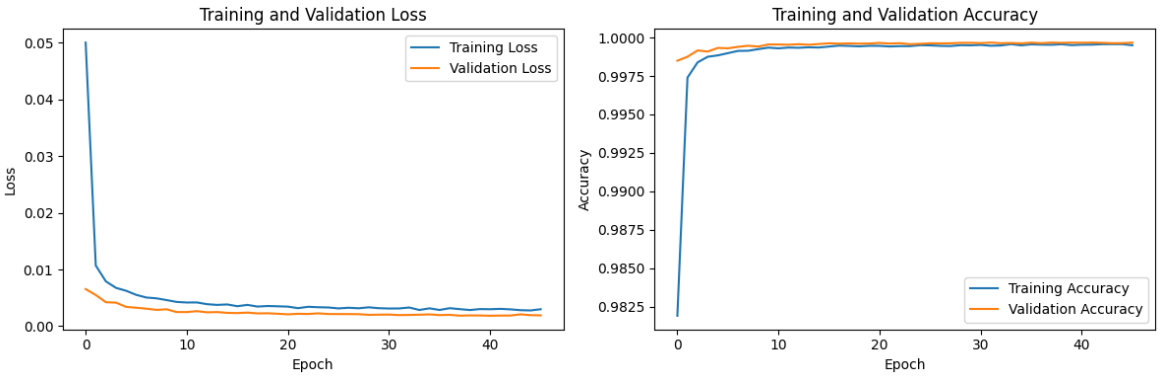
۱۱. ۳ Dropout: این لایه برای جلوگیری از اورفیتینگ با احتمال ۰.۵ استفاده می‌شود.

۱۲. ۲ Dense: این لایه با ۶۴ واحد و تابع فعال‌سازی ReLU استفاده می‌شود.

۱۳. ۳ Dense: این لایه نهایی با یک واحد و تابع فعال‌سازی sigmoid برای انجام مسئله دسته‌بندی دودویی (binary classification) استفاده می‌شود.

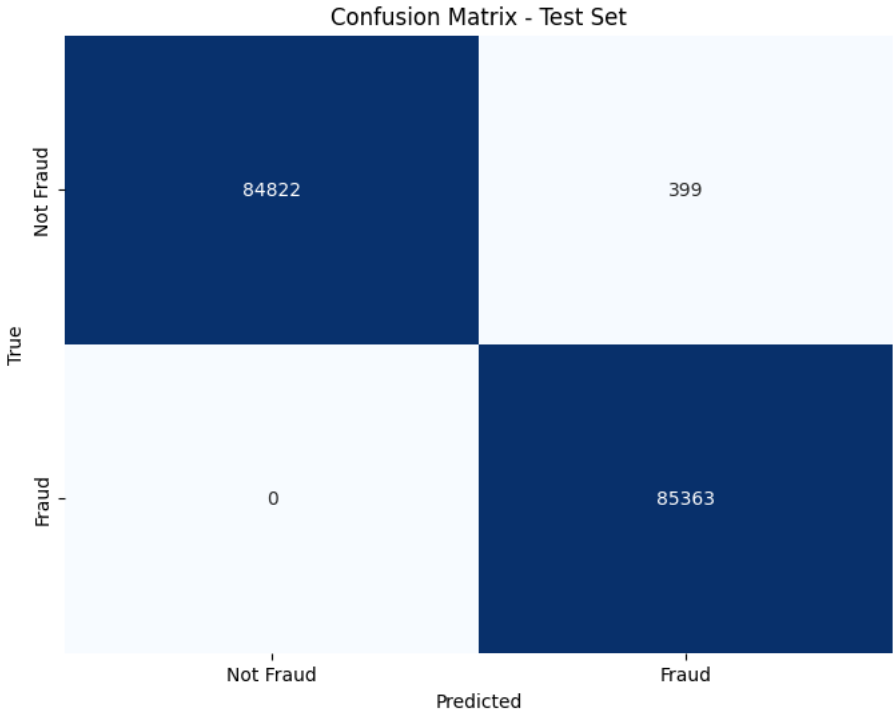
به این ترتیب، مدل نهایی دارای یک لایه ورودی (۱ Conv۱D Layer)، یک سری از لایه‌های میانی (BatchNormalization، MaxPooling، Dropout، ۲ Conv۱D Layer، ۲ BatchNormalization، ۲ MaxPooling۱D، ۲ Dropout، Flatten، ۱ Dense، Dropout ۳، ۲ Dense) و یک لایه خروجی (۳ Dense) است. این مدل مناسب برای مسائل دسته‌بندی دودویی با داده‌های دوره‌ای D۱ می‌باشد.

حال نتایج را به صورت زیر به دست می آوریم:



شکل ۳۳: نتایج با نمونه برداری

خطای هر هم داده‌های تست هم داده‌های ارزیابی در حال کم شدن است و تا حد زیادی مدل overfit نشده است و برای داده‌های خارج از آموزش نیز دارای عمومیت می‌باشد.



شکل ۳۴: ماتریس در هم ریختگی با نمونه برداری

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85221
           1       1.00      1.00      1.00     85363

    accuracy                           1.00    170584
   macro avg       1.00      1.00      1.00    170584
weighted avg       1.00      1.00      1.00    170584
```

شکل ۳۵: نتایج مربوط به نمونه برداری

همچنین عملکرد این مدل برای معایرهایی غیر دقت به شرح زیر می‌باشد:

```
Precision: 0.9997390613324477
Recall: 0.9995138147752876
F1-score: 0.9996264253651248
```

شکل ۳۶ معیارهای صحت و **recall** با نمونه‌برداری

مشخص است که این مدل هم برای کلاس اقلیت هم برای کلاس اکثریت به بهترین نحو عمل کرده است و نوع توزیع هر دو کلاس را به درستی یاد گرفته است.