به نام خداوند جان و خرد

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

پردازش زبان طبیعی

تمرین شماره ۴

نام و نام خانوادگی: علی خرم فر

شماره دانشجویی: ۸۱۰۱۰۲۱۲۹

خردادماه ۱۴۰۳

فهرست مطالب

۱_ پاسخ سوال اول

باتوجه به سوال ارائه شده، هدف ما مقایسه عملکرد مدل‌های فاین‌تیون شده با استفاده از روش‌های سنتی و روش LoRA است. باتوجه به اینکه مدل‌های زبانی پارامترهای بسیار زیاده داشته و لازمه‌ی آموزش آن‌ها منابع سخت‌افزاری قوی است، نیاز است که روش‌های انجام فاین‌تیون را که میزان حافظه مورد نیاز را کاهش می‌دهند را بررسی کنیم.

۱-۱_ دادگان

بررسی داده‌ها

در این سوال برای ارزیابی و مقایسه روش‌های مختلف آموزش مدل زبانی، از دادگان MultiNLI استفاده می‌کنیم. این دیتاست شامل جملات و عبارات مختلفی است که برای استنتاج زبانی مورد استفاده قرار می‌گیرد. این مجموعه داده شامل جفت جملاتی است که باید تعیین شود آیا جمله دوم نتیجه منطقی جمله اول است، با آن تناقض دارد و یا هیچ رابطه‌ای ندارد. به طور کلی ۳ برچسب زیر برای هر جفت جمله در این دیتاست وجود دارد:

Entailment (استنتاج): جمله دوم نتیجه منطقی جمله اول است.

Contradiction (تناقض): جمله دوم با جمله اول تناقض دارد.

Neutral (خنثی): جمله دوم هیچ رابطه‌ای با جمله اول ندارد.

برای بررسی تعداد داده‌ها و ویژگی‌های آن‌ها از دیتاست خروجی گرفتیم که تعداد داده‌های هر بخش و ویژگی‌ها به صورت زیر است:

```
print(dataset)
```

```
DatasetDict({
    train: Dataset({
        features: ['promptID', 'pairID', 'premise', 'premise_binary_parse', 'premise_parse', 'hypothesis', 'hypothesis_binary_parse', 'hypo
thesis_parse', 'genre', 'label'],
        num_rows: 392702
    })
    validation_matched: Dataset({
        features: ['promptID', 'pairID', 'premise', 'premise_binary_parse', 'premise_parse', 'hypothesis', 'hypothesis_binary_parse', 'hypo
thesis_parse', 'genre', 'label'],
        num_rows: 9815
    })
    validation_mismatched: Dataset({
        features: ['promptID', 'pairID', 'premise', 'premise_binary_parse', 'premise_parse', 'hypothesis', 'hypothesis_binary_parse', 'hypo
thesis_parse', 'genre', 'label'],
        num_rows: 9832
    })
})
```

ویژگی‌ها: ['promptID', 'pairID', 'premise', 'premise_binary_parse', 'premise_parse', 'hypothesis', 'hypothesis_binary_parse', 'hypothesis_parse', 'genre', 'label']

پیش‌پردازش داده‌ها

همانطور که در متن تمرین هم گفته‌شده، از ۱۰ درصد بخش آموزشی داده‌ها استفاده می‌کنیم تا زمان و منابع محاسباتی بهینه شوند. همچنین داده‌ها باید به فرمتی تبدیل شوند که برای ورودی مدل زبانی مناسب باشد. این پیش‌پردازش‌ها شامل توکنایز کردن جملات و قراردادن برچسب‌های مربوطه به بهترین حالت در جمله ورودی است.

۲-۱_ روش های فاین‌تیون

فاین‌تیون کل پارامترهای مدل

فاین‌تیون کل پارامترهای مدل به معنای به‌روزرسانی تمامی پارامترهای یک مدل از پیش آموزش دیده است. در این روش، تمام وزن‌ها و بایاس‌های شبکه عصبی در طی فرآیند آموزش دوباره تنظیم شده و این امر منجر به افزایش دقت مدل بر روی داده‌های جدید می‌شود، چرا که مدل می‌تواند الگوهای خاص وظیفه جدید را بهتر شناسایی کند. با این حال، این روش نیازمند منابع محاسباتی زیادی است و زمان طولانی برای آموزش نیاز دارد که این امر برای پروژه‌هایی با محدودیت منابع و زمان چالش‌هایی به دنبال خواهد داشت. علاوه بر این، در صورتی که داده‌های آموزشی محدود باشند، احتمال بیش‌برازش افزایش می‌یابد، زیرا که مدل به جای یادگیری الگوهای کلی، جزئیات داده‌های آموزشی را حفظ می‌کند و در نتیجه عملکرد ضعیفی بر روی داده‌های جدید خواهد داشت. در مجموع، اگرچه این روش دقت بالایی ارائه می‌دهد، اما نیاز به منابع قابل توجه و زمان Train طولانی دارد.

فاین‌تیون یک یا چند لایه از لایه‌های مدل

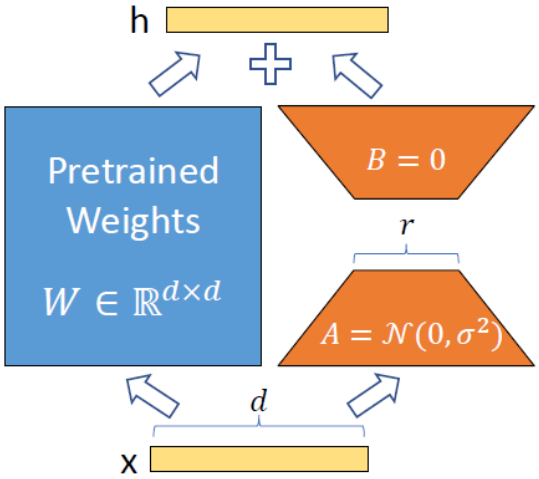
در این روش، تنها یک یا چند لایه از لایه‌های مدل به‌روزرسانی می‌شوند و باقی پارامترها ثابت می‌مانند که معمولاً لایه‌های بالایی (نزدیک به خروجی) برای فاین‌تیون انتخاب می‌شوند، زیرا این لایه‌ها به نتیجه یک تسک خاص حساس‌تر هستند. یکی از مزایای اصلی این روش، کاهش زمان و منابع محاسباتی مورد نیاز است، زیرا تنها بخشی از پارامترهای مدل به روزرسانی می‌شوند. این امر کمک می‌کند که فرآیند آموزش سریع‌تر انجام شده و نیاز به سخت‌افزارهای قدرتمند کاهش یابد. همچنین، با توجه به کاهش تعداد پارامترهای قابل تنظیم، خطر بیش‌برازش نیز کاهش می‌یابد. با این حال، ممکن است دقت مدل به اندازه فاین‌تیونینگ کل پارامترها بهبود نیابد ممکن است برخی از ویژگی‌های مهم نادیده گرفته شوند.

روش LoRA (Low-Rank Adaptation)

روش LoRA یک تکنیک جدید برای فاین‌تیون مدل‌های زبانی است که به جای به‌روزرسانی تمامی پارامترهای مدل، فقط پارامترهای خاصی را بهینه می‌کند. این روش بر پایه کاهش تعداد پارامترهای قابل آموزش و استفاده از low-rank matrices استوار است.(Hu et al., 2021)

در این روش پارامترهای مدل به دو دسته تقسیم می‌شوند: پارامترهای ثابت و پارامترهای قابل آموزش. پارامترهای قابل آموزش با استفاده از ماتریس‌های low-rank بهینه‌سازی می‌شوند. این ماتریس‌ها تعداد پارامترهای قابل آموزش را به طور قابل توجهی کاهش می‌دهند. به جای به‌روزرسانی مستقیم تمامی پارامترهای مدل، LoRA تغییرات کوچکی را در فضای کم‌مرتبه اعمال می‌کند که می‌توانند به راحتی با داده‌های جدید سازگار شوند. این روش امکان افزایش دقت مدل را با مصرف منابع کمتر فراهم می‌کند.

به این منظور از دو ماتریس low-rank A و B استفاده می‌شود. این ماتریس‌ها با اعمال تغییرات کوچک در پارامترهای ورودی و خروجی مدل، به بهینه‌سازی مدل کمک می‌کنند. ماتریس‌های A و B ابعاد کوچکتری نسبت به پارامترهای اصلی مدل دارند و این امر باعث کاهش نیاز به منابع محاسباتی و زمان آموزش می‌شود.



روش LoRA به دلیل کاهش تعداد پارامترهای قابل آموزش، منابع محاسباتی و زمان مورد نیاز برای فاین‌تیونینگ را به شدت کاهش می‌دهد. با این حال، طراحی ماتریس‌های low-rank نیاز به دقت و تخصص دارد و باید به گونه‌ای تنظیم شوند که ویژگی‌های مهم مدل را حفظ کنند. به طور کلی بسیاری از پارامترهای مدل ثابت باقی می‌مانند و تغییر نمی‌کنند که این پارامترها شامل وزن‌ها و بایاس‌های اصلی شبکه عصبی هستند که در مرحله pre-training بهینه شده‌اند. با ثابت نگه داشتن این پارامترها، می‌توان از مزایای مدل از پیش آموزش دیده بهره‌مند شد و در عین حال فرآیند فاین‌تیون را ساده‌تر و سریع‌تر کرد.

۳-۱_ روش‌های مبتنی بر پرامپت

روش Hard Prompt

روش Hard Prompt به معنی استفاده از رشته‌های ثابت و از پیش تعیین‌شده از کلمات یا جملات به عنوان ورودی مدل است. این پرامپت‌ها به گونه‌ای طراحی می‌شوند که مدل را برای انجام یک Task خاص مثل NLI راهنمایی کنند. در این روش، پرامپت‌ها به صورت دستی و با استفاده از زبان طبیعی نوشته شده و ثابت باقی می‌مانند. پرامپت‌های ثابت نیاز به آموزش اضافی ندارند و به صورت مستقیم به مدل داده می‌شوند با این حال ممکن است نتوانند به خوبی با تمامی Taskها و داده‌ها تطبیق پیدا کنند. این مورد برای اهداف پیچیده مشکل‌تر هم خواهدشد. (Brown et al., 2020)

روش Soft Prompt

روش Soft Prompt به معنای استفاده از پرامپت‌های قابل آموزش و پویا است که به جای رشته‌های ثابت، به عنوان پارامترهای قابل یادگیری در مدل قرار گرفته و در طی فرآیند آموزش مدل بهینه می‌شوند. از این جهت به خوبی با داده‌ها و وظایف مختلف سازگار می‌شوند.

هرچند که فرآیند آموزش پرامپت‌های قابل یادگیری نیاز به منابع محاسباتی بیشتری داشته و پیچیدگی بیشتری نسبت به پرامپت‌های ثابت دارند.(Li & Liang, 2021)

۱-۴_ آموزش مدل

در این بخش، مدل RoBERTa-large را دریافت کرده و با کمک روش‌های مختلف آن را بر روی دادگان MultiNLI آموزش می‌دهیم. قبل از انجام آموزش نیز داده‌ها را توکنایز کرده و برای تمامی روش‌ها استفاده می‌کنیم.

توکنایزکردن داده‌ها

برای آموزش مدل ، ابتدا باید داده‌ها را به فرمتی تبدیل کنیم که برای ورودی مدل مناسب باشند. به این منظور توکنایزر مدل RoBERTa-large را از Hugging Face دریافت و بارگذاری می‌کنیم. یک تابع برای پیش‌پردازش داده‌ها تعریف می‌کنیم که شامل توکنایز کردن جملات و اعمال پیش‌پردازش‌های لازم است.

```
def preprocess_function(examples):
    return tokenizer(
        examples['premise'],
        examples['hypothesis'],
        truncation=True,
        padding='max_length',
        max_length=128
    )
```

مقدار truncation=True مشخص می‌کند که جملات باید به طول معینی بریده شوند اگر طول آن‌ها از حد مجاز بیشتر باشد. در اینجا، طول جملات به حداکثر مقدار max_length یعنی ۱۲۸ توکن محدود می‌شود. همچنین تمامی جملات ورودی باید پدینگ شوند.

در نهایت داده‌های آموزشی و اعتبارسنجی را با استفاده از تابع پیش‌پردازش توکنایز کرده و به فرمت پایتورچ تبدیل می‌کنیم.

۱-۵_ آموزش با بروزرسانی کل پارامترهای مدل

ابتدا مدل RoBERTa-large را با استفاده از کتابخانه‌های موجود بارگذاری می‌کنیم.

بررسی معماری

مدل RoBERTa-large از چندین لایه تشکیل شده است که هر کدام وظیفه خاصی بر عهده دارند.

```
summary(model, input_size=(1, 128), dtypes=[torch.long])
```

```
==========================================================================================
Layer (type:depth-idx)                                  Output Shape              Param #
==========================================================================================
RobertaForSequenceClassification                        [1, 3]                    --
├─RobertaModel: 1-1                                     [1, 128, 1024]            --
│    └─RobertaEmbeddings: 2-1                           [1, 128, 1024]            --
│    │    └─Embedding: 3-1                              [1, 128, 1024]            51,471,360
│    │    └─Embedding: 3-2                              [1, 128, 1024]            1,024
│    │    └─Embedding: 3-3                              [1, 128, 1024]            526,336
│    │    └─LayerNorm: 3-4                              [1, 128, 1024]            2,048
│    │    └─Dropout: 3-5                                [1, 128, 1024]            --
│    └─RobertaEncoder: 2-2                              [1, 128, 1024]            --
│    │    └─ModuleList: 3-6                             --                        302,309,376
├─RobertaClassificationHead: 1-2                        [1, 3]                    --
│    └─Dropout: 2-3                                     [1, 1024]                 --
│    └─Linear: 2-4                                      [1, 1024]                 1,049,600
│    └─Dropout: 2-5                                     [1, 1024]                 --
│    └─Linear: 2-6                                      [1, 3]                    3,075
==========================================================================================
Total params: 355,362,819
Trainable params: 355,362,819
Non-trainable params: 0
Total mult-adds (M): 355.36
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 281.03
Params size (MB): 1421.45
Estimated Total Size (MB): 1702.48
==========================================================================================
```

RobertaForSequenceClassification:

خروجی نهایی این بخش، یک بردار با اندازه [۱, ۳] است که نشان‌دهنده احتمال هر یک از سه کلاس خروجی است.

RobertaModel:

هسته اصلی مدل RoBERTa

:RobertaEmbeddings

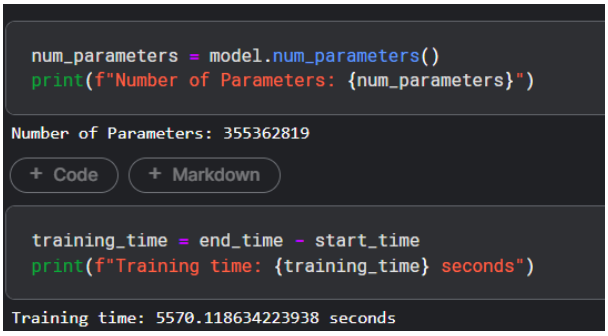
این بخش وظیفه تبدیل توکن‌های ورودی به بردارهای امبدینگ را بر عهده دارد.

:RobertaEncoder

این بخش شامل مجموعه‌ای از لایه‌های از نوع ترنسفورمر است که وظیفه استخراج ویژگی‌های عمیق‌تر از بردارهای امبدینگ شده را بر عهده دارند.

:RobertaClassificationHead

این بخش برای طبقه‌بندی نهایی استفاده می‌شود. از DropOut برای جلوگیری از بیش‌برازش استفاده می‌شود.



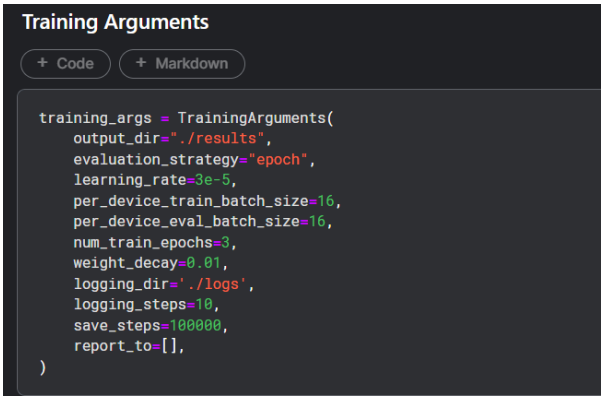
تعداد کل پارامترها: ۳۵۵,۳۶۲,۸۱۹

پارامترهای قابل آموزش: ۳۵۵,۳۶۲,۸۱۹

زمان آموزش ۵۵۷۰ ثانیه یا ۹۲ دقیقه

بررسی هایپرپارامترها

از تنظیمات زیر برای آموزش مدل استفاده شد:



ابتدا هرکدام را به صورت کلی معرفی کرده و سپس به ادامه سوال می‌پردازیم.

output_dir="./results:"

این هایپرپارامتر دایرکتوری را مشخص می‌کند که در آن نتایج آموزش و مدل‌های ذخیره‌شده قرار می‌گیرند.

evaluation_strategy="epoch:"

این پارامتر استراتژی ارزیابی مدل را تعیین می‌کند. با تنظیم آن بر روی "epoch"، مدل در پایان هر ایپاک (epoch) ارزیابی می‌شود.

learning_rate=3e-5:

نرخ یادگیری (learning rate) که باتوجه به تحقیقات انجام شده برای مدل‌های زبانی بزرگ بهتر است مقادیر کم مثل 3e-5 استفاده کنیم.

per_device_train_batch_size=16:

این پارامتر تعداد نمونه‌هایی که در هر مرحله آموزش (بچ) برای GPU ارسال و پردازش می‌شوند را تعیین می‌کند.

num_train_epochs=3:

تعداد ایپاک‌ها همان تعداد کل تکرارهایی است که مدل روی داده‌های آموزشی طی می‌کند.

weight_decay=0.01:

این پارامتر نرخ ضعیف‌شدن را تعیین کرده که به منظور جلوگیری از بیش‌برازش استفاده می‌شود.

logging_dir='./logs:'

مسیری برای ذخیره لاگ‌های آموزش مدل مشخص می‌کند.

logging_steps=10:

تعداد مراحل بین هر ثبت لاگ را تعیین می‌کند. با تنظیم آن بر روی ۱۰، هر ۱۰ مرحله آموزش یک لاگ ثبت می‌شود.

save_steps=100000:

این پارامتر تعداد مراحلی را که بین هر ذخیره‌سازی مدل انجام می‌شود تعیین می‌کند.

نکته‌ای که در این پارامترها وجود دارد مقادیر فعلی بهترین نتیجه را دادند. نرخ‌های یادگیری متفاوتی بررسی شد و یا اینکه save_steps به دلیل محدودیت فضا مقدار بالاتری داده شد تا فضا پر نشود. برای ارزیابی مدل در طول آموزش نیز باتوجه به خواسته مسئله از متریک accuracy یا دقت استفاده می‌کنیم.

در نهایت، با استفاده از Trainer، تمامی این تنظیمات و پارامترها برای مدیریت آموزش مدل استفاده شدند که در ادامه نتیجه را بررسی می‌کنیم.

نتایج آموزش مدل

[7365/7365 1:32:49, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.510700	0.473953	0.840652
2	0.310800	0.449588	0.878859
3	0.218100	0.618880	0.877534

مقدار از Training Loss در طی هر ایپاک کاهش یافته است، که نشان‌دهنده بهبود عملکرد مدل بر روی داده‌های آموزشی است.از طرفی مقدار Validation Loss در دو ایپاک اول کاهش یافته است، اما در ایپاک سوم افزایش یافته که این مورد می‌تواند نشانه‌ی از بیش‌برازش باشد. دقت مدل نیز بر روی داده‌های اعتبارسنجی روند صعودی دارد، هرچند که در ایپاک سوم کمی کاهش یافته است.

دقت مدل بر روی ۲ دسته از داده‌های ارزیابی به صورت زیر است. دسته دوم داده‌هایی هستند که از جنس داده‌های آموزشی نبوده و به اصطلاح mismatched هستند.

Validation matched accuracy: 0.8775343861436576

Validation mismatched accuracy: 0.8746948738812043

۶-۱_ آموزش مدل RoBERTa-large با استفاده از روش LoRA

در بخش قبلی با مدل LoRA آشنا شدیم. در این قسمت قصد داریم که این مدل را با کمک روش LoRA فاین تیون کنیم. در قسمت قبلی دلیل استفاده و توضیح هرکدام از هایپرپارامترها داده شد. در این بخش موارد درخواستی را فقط گزارش می‌کنیم.

بررسی معماری

```
==========================================================================================================================
Layer (type:depth-idx)                                                  Output Shape              Param #
==========================================================================================================================
PeftModelForSequenceClassification                                      [1, 3]                    --
├─LoraModel: 1-1                                                        [1, 3]                    --
│    └─RobertaForSequenceClassification: 2-1                            --                        --
│    │    └─RobertaModel: 3-1                                           [1, 128, 1024]            355,096,576
│    │    └─ModulesToSaveWrapper: 3-2                                   [1, 3]                    2,105,350
==========================================================================================================================
Total params: 357,201,926
Trainable params: 1,839,107
Non-trainable params: 355,362,819
Total mult-adds (M): 356.15
==========================================================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 331.75
Params size (MB): 1424.60
Estimated Total Size (MB): 1756.35
==========================================================================================================================
```

:PeftModelForSequenceClassification

این بخش کل مدل SequenceClassification را شامل می‌شود که با استفاده از LoRA تنظیم شده است. خروجی نهایی این بخش، یک بردار با اندازه [۱, ۳] است که نشان‌دهنده احتمال هر یک از سه کلاس خروجی است.

:LoraModel

این لایه به مدل اصلی اضافه شده است تا از روش LoRA برای کاهش تعداد پارامترهای قابل آموزش استفاده کند.

:RobertaForSequenceClassification

این بخش مدل اصلی RoBERTa-large را شامل می‌شود که برای SequenceClassification استفاده می‌شود.

:RobertaModel

هسته اصلی مدل RoBERTa

:ModulesToSaveWrapper

این لایه برای ذخیره ماژول‌های اضافه شده به مدل استفاده می‌شود. این ماژول‌ها شامل پارامترهای LoRA هستند که قابل آموزش می‌باشند.

تعداد کل پارامترها: ۳۵۷,۲۰۱,۹۲۶

پارامترهای قابل آموزش: ۱,۸۳۹,۱۰۷

پارامترهای غیرقابل آموزش: ۳۵۵,۳۶۲,۸۱۹

زمان آموزش ۴۰۶۴ ثانیه یا ۶۸ دقیقه

بررسی هایپرپارامترها

هایپرپارامترهای LoRA:

```
lora_config = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    r=8,
    lora_alpha=32,
    lora_dropout=0.1
)
```

task_type=TaskType.SEQ_CLS:

این پارامتر نوع Task را مشخص می‌کند که در اینجا Sequence Classification است.

r=8 (Rank of the low-rank decomposition):

این پارامتر درجه تجزیه Rank of the low-rank decomposition را تعیین می‌کند. مقدار r=8 نشان می‌دهد که ماتریس‌های low-rank دارای درجه ۸ هستند که این مقدار تعیین می‌کند که چه تعداد ابعاد در ماتریس‌های جدید باید حفظ شوند و به طور کلی یک تعادل بین دقت و کاهش تعداد پارامترها است.

lora_alpha=32 (Scaling factor for the low-rank decomposition):

این پارامتر ضریب تجزیه low-rank را تعیین کرده و به تنظیم شدت تأثیر ماتریس‌ها بر روی پارامترهای مدل اصلی کمک می‌کند.

lora_dropout=0.1 (Dropout for LoRA layers):

این پارامتر نرخ Dropout برای لایه‌های LoRA را تعیین می‌کند.

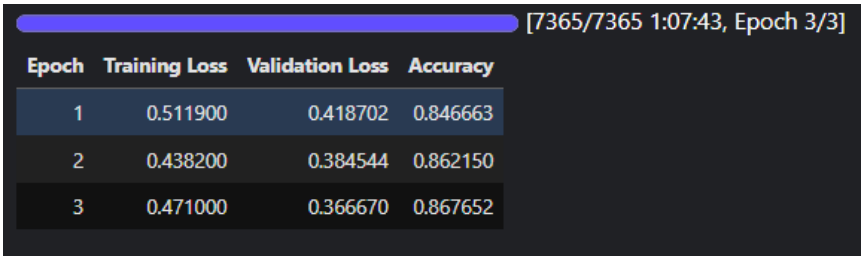
همچنین از از تنظیمات زیر برای آموزش مدل استفاده شد:

```
training_args_lora = TrainingArguments(
    output_dir="./results-lora",
    evaluation_strategy="epoch",
    learning_rate=3e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir='./logs-lora',
    logging_steps=10,
    save_steps=100000,
    report_to=[],
)
```

توضیحات کامل هر کدام از هایپرپارامترها در قسمت قبلی داده شد.

نتایج آموزش مدل

تصویر زیر نتایج آموزش مدل RoBERTa-large با استفاده از روش LoRA را در طی سه ایپاک نشان می‌دهد:


[7365/7365 1:07:43, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.511900	0.418702	0.846663
2	0.438200	0.384544	0.862150
3	0.471000	0.366670	0.867652

مقدار از Training Loss در ایپاک اول برابر ۰.۵۱۱۹۰۰ بود و در ایپاک دوم به ۰.۴۳۸۲۰۰ کاهش یافت. با این حال، در ایپاک سوم مقدار آن به ۰.۴۷۱۰۰۰ افزایش یافت. مقدار Validation Loss نیز در طول سه ایپاک کاهش یافته است. دقت مدل بر روی داده‌های اعتبارسنجی نیز در طول سه ایپاک بهبود یافته است.

در مقایسه با نتایج آموزش قبلی که تمامی پارامترها به‌روزرسانی می‌شدند، روش LoRA نتایج قابل توجهی ارائه می‌دهد. این مدل با استفاده از روش LoRA توانسته است به دقت مشابه با اختلاف حدود ۱ درصد در مقایسه با روش سنتی دست یابد، اما با منابع محاسباتی کمتر و زمان کوتاه‌تر.

Validation matched accuracy (LoRA): 0.8676515537442689

Validation mismatched accuracy (LoRA): 0.8668633034987795

۷-۱_ آموزش با استفاده از روش P-Tuning

در این بخش، مدل RoBERTa-large را با استفاده از روش P-Tuning که نوعی روش Soft Prompting به حساب می‌آید، آموزش می‌دهیم.

بررسی معماری

```
==========================================================================================
Layer (type:depth-idx)                                  Output Shape              Param #
==========================================================================================
PeftModelForSequenceClassification                      [1, 3]                    --
├─Embedding: 1-1                                        [1, 128, 1024]            51,471,360
├─ModuleDict: 1-2                                       --                        --
│    └─PromptEmbedding: 2-1                             [1, 30, 1024]             --
│    │    └─Embedding: 3-1                              [1, 30, 1024]             (30,720)
├─RobertaForSequenceClassification: 1-3                 [1, 3]                    --
│    └─RobertaModel: 2-2                                [1, 158, 1024]            --
│    │    └─RobertaEmbeddings: 3-2                      [1, 158, 1024]            52,000,768
│    │    └─RobertaEncoder: 3-3                         [1, 158, 1024]            (302,309,376)
│    └─ModulesToSaveWrapper: 2-3                        [1, 3]                    1,052,675
│    │    └─ModuleDict: 3-4                             --                        (1,052,675)
==========================================================================================
Total params: 407,917,574
Trainable params: 102,942,720
Non-trainable params: 304,974,854
Total mult-adds (M): 355.39
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 346.89
Params size (MB): 1421.57
Estimated Total Size (MB): 1768.47
==========================================================================================
```

PeftModelForSequenceClassification:

کل مدل SequenceClassification را شامل شده که با استفاده از روش Tuning-P تنظیم شده است. خروجی نهایی این بخش، یک بردار با اندازه [۱, ۳] است که نشان‌دهنده احتمال هر یک از سه کلاس خروجی است.

Embedding:

این لایه وظیفه تبدیل توکن‌های ورودی به بردارهای امبدینگ را دارد.

ModuleDict و PromptEmbedding:

این بخش شامل توکن‌های مجازی است که به ورودی مدل اضافه می‌شوند. این توکن‌ها به عنوان پارامترهای قابل آموزش به مدل اضافه شده و به بهبود عملکرد مدل کمک می‌کنند.

RobertaForSequenceClassification:

این بخش مدل اصلی RoBERTa-large را شامل می‌شود که برای SequenceClassification استفاده می‌شود.

RobertaModel:

هسته اصلی مدل RoBERTa

RobertaEncoder:

این بخش شامل مجموعه‌ای از لایه‌های از نوع ترنسفورمر است که وظیفه استخراج ویژگی‌های عمیق‌تر از بردارهای امبدینگ شده را بر عهده دارند.

ModulesToSaveWrapper و ModuleDict:

این لایه برای ذخیره ماژول‌های اضافه شده به مدل استفاده می‌شوند.

در مقایسه با معماری مدل اصلی و LoRA، مدل با استفاده از Tuning-P دارای تعداد بیشتری پارامتر قابل آموزش است. با این حال، این پارامترها به طور خاص برای بهبود کارایی مدل با استفاده از توکن‌های مجازی تنظیم شده‌اند.

بررسی هایپرپارامترها

پارامترهای P-Tuning :

```
prompt_tuning_config = PromptTuningConfig(
    task_type=TaskType.SEQ_CLS,
    num_virtual_tokens=30,
)

model = get_peft_model(model, prompt_tuning_config)

for param in model.parameters():
    param.requires_grad = False
for param in model.get_input_embeddings().parameters():
    param.requires_grad = True
```

task_type=TaskType.SEQ_CLS:

این پارامتر Taskرا مشخص می‌کند که در اینجا Sequence Classification است.

num_virtual_tokens=30

این پارامتر تعداد توکن‌های مجازی را تعیین می‌کند که به ورودی مدل اضافه می‌شوند.

فریز کردن تمامی پارامترها:

```
for param in model.parameters():
    param.requires_grad = False
for param in model.get_input_embeddings().parameters():
    param.requires_grad = True
```

برای کاهش تعداد پارامترهای قابل آموزش و بهبود کارایی مدل، تمامی پارامترهای مدل به جز پارامترهای مربوط به Tuning-P فریز می‌شوند. این کار باعث می‌شود که تنها توکن‌های مجازی قابل آموزش باشند و سایر پارامترهای مدل ثابت باقی بمانند.

تعداد کل پارامترها: ۴۰۷,۹۱۷,۵۷۴

پارامترهای قابل آموزش: ۱۰۲,۹۴۲,۷۲۰

پارامترهای غیرقابل آموزش: ۳۰۴,۹۷۴,۸۵۴

زمان آموزش ۴۴۶۶ ثانیه یا ۷۵ دقیقه

نتایج آموزش مدل

تصویر زیر نتایج آموزش مدل RoBERTa-large با استفاده از روش Prompt-Tuning را در طی سه ایپاک نشان می‌دهد:

Epoch	Training Loss	Validation Loss	Accuracy
1	1.193300	1.126348	0.354457
2	1.173000	1.124661	0.354457
3	1.134500	1.124166	0.354457

مقدار Training Loss در طول ایپاک‌ها به تدریج کاهش یافته است، اما میزان کاهش چندان زیاد نیست. دقت مدل بر روی داده‌های اعتبارسنجی در طول سه ایپاک ثابت مانده است. دقت در هر سه ایپاک برابر ۰.۳۵۴۴۵۷ بود که نشان می‌دهد مدل نتوانسته است به خوبی الگوهای موجود در داده‌های آموزشی را یاد بگیرد و آن‌ها را به داده‌های اعتبارسنجی تعمیم دهد. در مقایسه با نتایج آموزش کامل پارامترها و LoRA، روش P-Tuning نتوانسته است به دقت و کارایی مشابهی دست یابد. باتوجه به دقت پایین این مدل را با پارامترهای مختلف نیز آموزش دادیم ولی در نهایت بهبودی حاصل نشد و بعضا دقت کاهش یافت

۸-۱_ مقایسه روش LoRA با روش‌های سنتی‌تر

روش‌های سنتی

Fine-tuning کامل	
در این روش، تمام پارامترهای مدل اصلی برای هر تسک خاص به‌روز می‌شوند. این کار نیازمند منابع محاسباتی زیاد و زمان طولانی برای آموزش است.	
مزایا:	دقت بالا برای Task خاص
معایب:	نیازمند منابع زیاد احتمال بیش‌برازش (Overfitting) روی داده‌های آموزش خاص نیاز به مدل جداگانه برای هر وظیفه، که فضای ذخیره‌سازی بیشتری نیاز دارد

Fine-tuning چندلایه	
تنها برخی از لایه‌های مدل به‌روز می‌شوند. معمولاً لایه‌های بالایی مدل که ویژگی‌های عمومی‌تر را یاد گرفته‌اند، ثابت نگه داشته می‌شوند.	
مزایا:	کاهش نیاز به منابع محاسباتی حفظ برخی از ویژگی‌های عمومی
معایب:	همچنان نیاز به منابع قابل توجه دقت پایین‌تر نسبت به Fine-tuning کامل

روش LoRA

روش LoRA	
در LoRA تنها پارامترهای اضافی به‌صورت ماتریس‌های با Low-rank به مدل اضافه می‌شوند. پارامترهای اصلی مدل ثابت می‌مانند.	
مزایا:	کاهش نیاز به منابع محاسباتی امکان استفاده از یک مدل پایه برای چندین وظیفه کاهش زمان آموزش
معایب:	پیچیدگی نسبی در پیاده‌سازی ممکن است برای وظایف بسیار متفاوت کارایی کمتری داشته باشد.

استفاده از مدل بدون Fine-tuning مجدد

یک روش برای پیاده‌سازی این مورد، استفاده از مدل ثابت با Headهای خروجی جداگانه است. Output Heads به طور مجزا برای هر وظیفه خاص آموزش داده می‌شوند. مثلا یک Head خروجی برای تحلیل احساسات و یک Head دیگر برای پرسش و پاسخ آموزش داده می‌شود. از مزایای این روش می‌توان به نیاز به منابع کمتر نسبت به Fine-tuning کامل و امکان استفاده مجدد از مدل اصلی برای وظایف مختلف اشاره کرد هرچند که دقت ممکن است بهینه نباشد، زیرا که مدل اصلی برای هیچ‌کدام از وظایف بهینه‌سازی نمی‌شود.

روش دیگر نیز استفاده از LoRA به نحوی است که پارامترهای اضافی که به مدل اصلی اضافه می‌شوند به صورت ماتریس‌های low-rank هستند و پارامترهای اصلی مدل ثابت نگه داشته می‌شوند. برای هر تسک خاص می‌توان پارامترهای LoRA جداگانه‌ای داشت که به‌صورت موازی با پارامترهای اصلی پیاده‌سازی می‌شوند. در سوال اول که پیاده‌سازی شد، استفاده از روش LoRA برای مدل RoBERTa به خوبی نشان داد که می‌توان بدون نیاز به Fine-tuning کامل پارامترهای اصلی، عملکرد بالایی برای یک تسک خاص داشت.

این مورد در مقاله هم توضیح داده شده که این روش با ثابت نگه داشتن پارامترهای اصلی و به‌روزرسانی تنها ماتریس‌های low-rank، می‌تواند به‌طور موثری منابع محاسباتی و زمان آموزش را کاهش دهد و به همین دلیل مناسب برای کاربردهای چند وظیفه‌ای است.(Hu et al., 2021)

۲_پاسخ سوال دوم

۲-۱_ بارگذاری Llama

ابتدا مدل LLaMA را بارگذاری می‌کنیم تا بتوانیم از آن برای پیش‌بینی برچسب‌های داده‌ها استفاده کنیم. پس از نصب کتابخانه‌ها، مدل LLaMA و توکنایزر مرتبط با آن را بارگذاری می‌کنیم. حال باتوجه به محدود بودن منابع باید از روش‌های کوانتیزه‌کردن مدل زبانی استفاده کنیم تا بتوانیم مدل را بر روی کگل اجرا کنیم. از پارامترهای زیر برای Quantization استفاده می‌کنیم:

Quatization Parameters

```
bnb_config = BitsAndBytesConfig(
    load_in_4bit= True,
    bnb_4bit_quant_type= "nf4",
    bnb_4bit_compute_dtype= torch.float16,
    bnb_4bit_use_double_quant= False,
)
```

استفاده از Quantization ۴ بیتی باعث کاهش حجم مدل و افزایش سرعت پردازش می‌شود، بدون اینکه تاثیر خیلی زیادی بر دقت مدل داشته باشد. استفاده از torch.float16 به معنای استفاده از half precision است. همچنین Quantization دوگانه ممکن است دقت مدل را بهبود بخشد، اما در اینجا برای ساده‌سازی و کاهش پیچیدگی، این ویژگی را استفاده نکردیم.

پس از این مقداردهی این پارامترها، مدل را بارگذاری می‌کنیم. همچنین تابع زیر برای آماده‌سازی مدل به منظور آموزش با دقت کمتر (مثلاً ۴ بیت یا ۸ بیت) استفاده می‌شود. این تابع تغییراتی در ساختار مدل ایجاد می‌کند تا مدل بتواند با تعداد بیت‌های کمتر کار کند:

K-bit training

+ Code + Markdown

```
model = prepare_model_for_kbit_training(model)
model
```

پس از این مرحله به سراغ روش‌های مختلف آموزش از طریق ICL می‌رویم.

۲-۲_ بخش اول – ICL[1]

In-Context Learning رویکردی است که در آن مدل‌های زبان بزرگ مانند Llama از ورودی‌های متنی (پرامپت‌ها) برای یادگیری و انجام وظایف خاص بدون نیاز به آموزش مجدد استفاده می‌کنند. این مدل‌ها با استفاده از نمونه‌هایی در متن ورودی می‌توانند الگوها را تشخیص داده و پاسخ‌های مناسب تولید کنند. این روش کمک می‌کند که مدل‌ها با استفاده از مثال‌های ارائه شده در پرامپت‌ها، تسک‌های مختلفی مانند ترجمه، تکمیل متن، طبقه‌بندی و ... را انجام دهند.

۲-۳_ بررسی روش‌های مختلف

در یک سناریوی Zero-Shot، مدل بدون دیدن هیچ مثالی از وظیفه مورد نظر، تلاش می‌کند تا بر اساس اطلاعاتی که در پرامپت داده شده، پاسخ دهد.

برای ارزیابی این سوال ۲ تابع نوشته شده است. یکی از آن‌ها پرامپت نوشته شده را به مدل می‌دهد و پاسخی که دریافت کرده را برمی‌گرداند. همچنین یک تابع برای اجرای پرامپت بر روی تمام دیتاست ارزیابی نوشته شده که تمامی پاسخ‌ها را دریافت کرده و دقت و ماتریس آشفتگی را خروجی می‌دهد.

در این سوال از از ۲ رویکرد متفاوت برای پرامپت‌نویسی استفاده کرده‌ایم و بر اساس هرکدام خروجی را ارزیابی کرده‌ایم. به طور کلی ۵ پرامپت متفاوت نوشته شد و سعی شد بهترین خروجی ممکن دریافت شود. در ادامه به بررسی هر رویکرد پرداخته و پس از آن تک‌تک پرامپت‌ها را بررسی می‌کنیم.

رویکرد اول

در این رویکرد قصد داریم مدل در پاسخ لیبل دقیق را برگرداند. این لیبل‌ها به صورت زیر هستند:

"Entailment", "Neutral", "Contradiction"

رویکرد دوم

در این رویکرد قصد داریم مدل در پاسخ جواب Yes یا No و یا Maybe برگرداند. در این رویکرد ما از مدل می‌پرسیم که آیا می‌توان از جمله اول، جمله دوم را نتیجه گرفت؟ که Yes همان معادل Entailment و No معادل Contradiction و Maybe معادل با Neutral است.

حال به بررسی پرامپت‌های نوشته شده بر اساس هرکدام از این رویکردها پرداخته و نتیجه را بررسی می‌کنیم.

---

[1] In-Context Learning

هایپرپارامترها

Temperature: 0.1

پارامتر temperature میزان خلاقیت مدل در تولید پاسخ‌ها را تنظیم می‌کند. مقادیر پایین‌تر باعث می‌شود مدل محافظه‌کارتر عمل کند و پاسخ‌های پیش‌بینی‌شده را با اطمینان بیشتری تولید کند. انتخاب مقدار ۰.۱ به این دلیل است که می‌خواهیم مدل پاسخ‌های دقیق‌تری تولید کند و از تولید پاسخ‌های خلاقانه یا غیرمرتبط جلوگیری کنیم. برای مقادیر بالاتر مثل ۰.۵ دقت تمامی پرامپت‌ها کاهش می‌یابد.

Max New Tokens: 1-2

این پارامتر تعداد توکن‌های جدیدی را که مدل می‌تواند تولید کند، محدود می‌کند. محدود کردن تولید به یک یا دو توکن برای اطمینان از اینکه مدل فقط برچسب مورد نظر (entailment, contradiction, neutral) را تولید کند.pad_token_id و eos_token_id نیز توکن‌هایی هستند که برای پایان دادن و پدینگ ورودی استفاده می‌شوند.

پرامپت اول – روش Zero shot – رویکرد اول

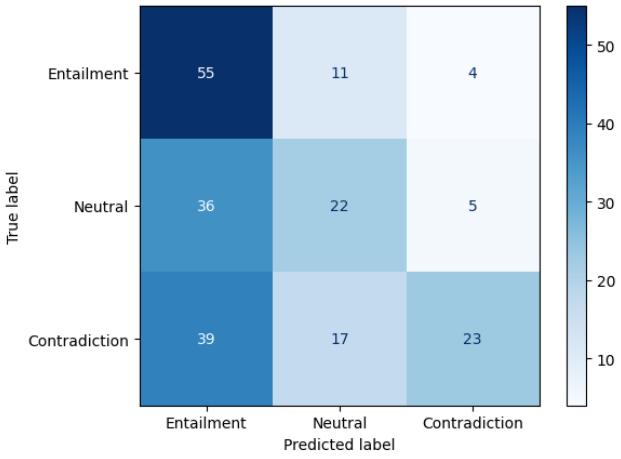
متن پرامپت:

```python
def create_zero_shot_prompt(premise, hypothesis):
    prompt = (f"NIL Task : premise: {premise}\n"
              f"hypothesis: {hypothesis}\n"
              f"Label (entailment, contradiction, neutral):")
    return prompt
```

دقت:

Accuracy: 0.4716981132075472

ماتریس آشفتگی:



با استفاده از پرامپت Zero-Shot، مدل توانسته است با دقت ۰.۴۷ پاسخ دهد. این دقت نشان می‌دهد که مدل در پیش‌بینی برچسب‌ها عملکرد متوسطی دارد هرچند که برای zero shot بسیار مناسب است. مدل در تمایز بین برچسب‌ها به خصوص در مورد Neutral و Contradiction مشکل دارد.

پرامپت دوم روش Zero shot – رویکرد دوم

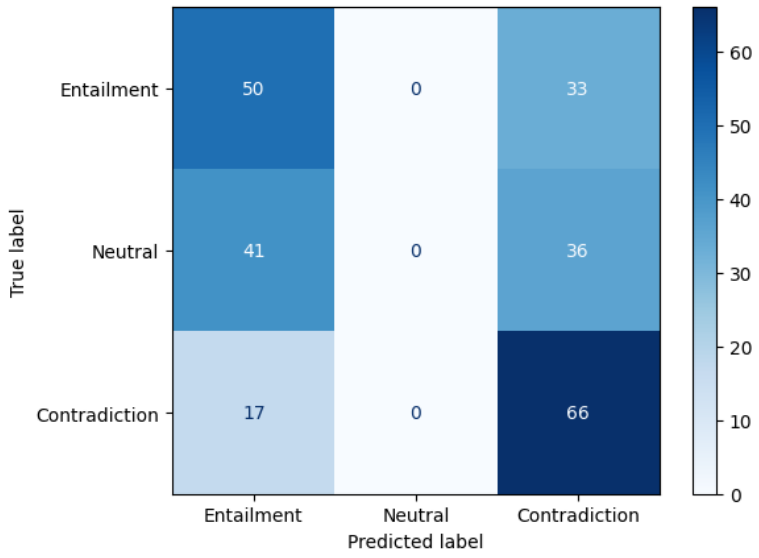
متن پرامپت:

```
def create_zero_shot_prompt(premise, hypothesis):
    prompt = (f"premise: {premise}\n"
              f"hypothesis: {hypothesis}\n"
              f"Does {premise} follow logically from the {hypothesis}?. Answer:")
    return prompt
```

دقت:

Accuracy: 0.4773662551440329

ماتریس آشفتگی:



با استفاده از پرامپت Zero-Shot جدید، مدل توانسته است به دقت ۰.۴۷۷ دست یابد که کمی بهتر از پرامپت قبلی است. اما همچنان مدل در پیش‌بینی برچسب‌های Neutral چالش دارد، به طوری که هیچ برچسب Neutral را به درستی پیش‌بینی نکرده است.

یکی از دلایل اصلی رویکرد پرامپت است که مبتنی بر Yes و No می‌باشد. در پرامپ بعدی سوال را با Maybe شروع می‌کنیم تا بتوانیم میانگین وزنی دقت را افزایش دهیم.

پرامپت سوم –روش Zero shot – رویکرد دوم

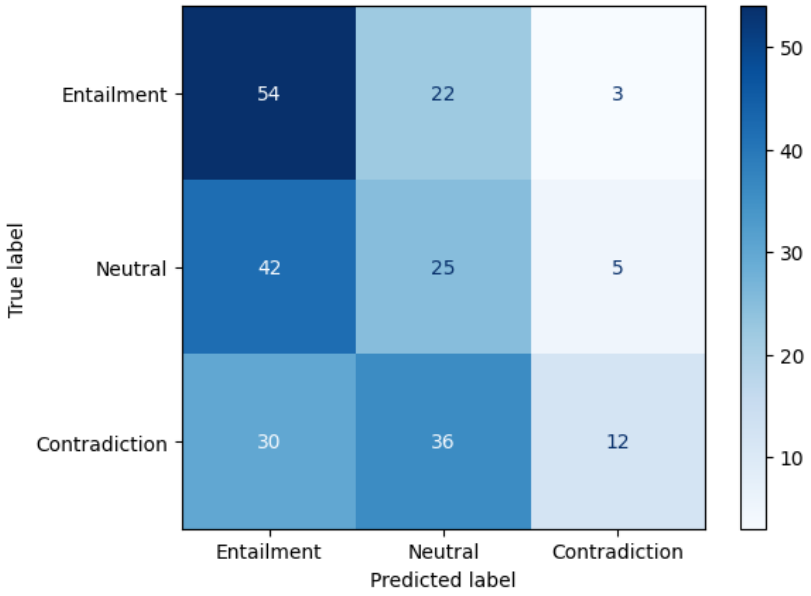
متن پرامپت:

```
def create_zero_shot_prompt(premise, hypothesis):
    prompt = (f"premise: {premise}\n"
              f"hypothesis: {hypothesis}\n"
              f"Maybe {premise} follow logically from the {hypothesis}.Maybe or Yes? No ? Answer:")
    return prompt
```

دقت:

Accuracy: 0.39737991266375544

ماتریس آشفتگی:



دقت مدل در این رویکرد کاهش یافته است ولی نسبت به پرامپت قبلی کلاس Neutral را بهتر تشخیص می‌دهد.

پرامپت چهارم – روش One-shot – رویکرد اول

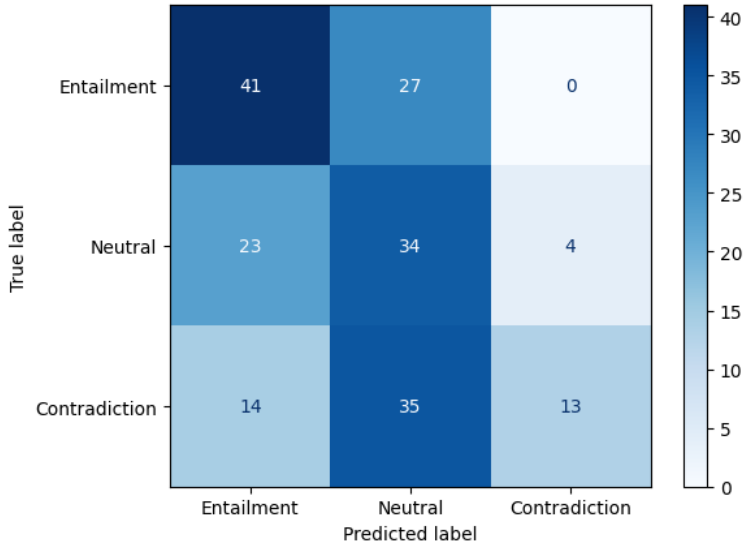
متن پرامپت:

```
def create_one_shot_prompt(premise, hypothesis):
    example_premise = "I did not mention Monica in my lecture, but the first question I was asked was how President Clinton could do his job with all
    example_hypothesis = "They wanted to get through the lecture without any problems."
    example_answer = "neutral"
    prompt = (f"Example:\n"
              f"Premise: \"{example_premise}\"\n"
              f"Hypothesis: \"{example_hypothesis}\"\n"
              f"Label: {example_answer}\n\n"
              f"Now, classify the following:\n"
              f"Premise: \"{premise}\"\n"
              f"Hypothesis: \"{hypothesis}\"\n"
              f"Label ( entailment, contradiction, neutral ):")
    return prompt
```

دقت:

Accuracy: 0.4607329842931937

ماتریس آشفتگی:



اگرچه دقت مدل در این روش افزایش نیافته است ولی یکی از چالش‌های اصلی یعنی تشخیص Neutral به خوبی حل شده است. دلیل استفاده از آن نمونه آموزشی هم همین مورد بود تا بتوانیم ضعفی که در رویکرد ZeroShot داشتیم را پوشش دهیم. در نمونه بعدی سعی می‌کنیم علاوه بر حل مشکلی قبلی، مشکل فعلی هم حل کنیم.

پرامپت پنجم – روش One-shot – رویکرد دوم

متن پرامپت:

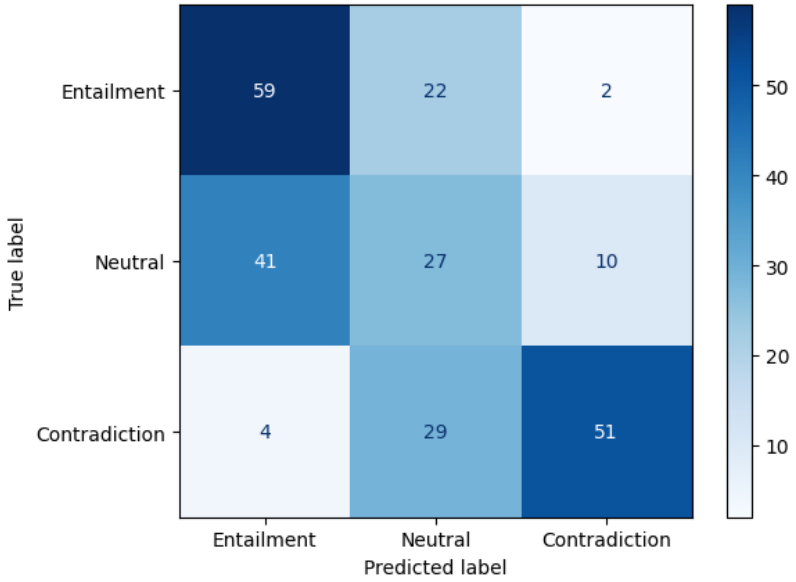
```python
def create_one_shot_prompt(premise, hypothesis):
    example_premise = "I did not mention Monica in my lecture, but the first question I was asked
    example_hypothesis = "They wanted to get through the lecture without any problems."
    example_answer = "Maybe"

    prompt = (f"Premise: {example_premise}\n"
              f"Hypothesis: {example_hypothesis}\n"
              f"Maybe the hypothesis follow logically from the premise. No, Yes or Maybe ? \n"
              f"Answer: {example_answer}\n\n"
              f"Premise: {premise}\n"
              f"Hypothesis: {hypothesis}\n"
              f"Maybe the hypothesis follow logically from the premise. No, Yes or Maybe ? \n"
              f"Answer:")
    return prompt
```

دقت:

Accuracy: 0.5591836734693878

ماتریس آشفتگی:



پرامپت One-Shot با رویکرد Yes, No, Maybe و استفاده از یک نمونه آموزشی توانسته است مدل را به سمت پیش‌بینی دقیق‌تری هدایت کرده و دقت قابل قبولی را نسبت به بقیه موارد ارائه دهد. استفاده از نمونه آموزشی در پرامپت One-Shot با رویکرد دوم می‌تواند به بهبود دقت مدل کمک کند، اما همچنان نیاز به بهبود بیشتر برای تشخیص دقیق‌تر برچسب‌ها وجود دارد.

در جدول زیر نتایج کلی هر ۵ پرامپت را مشاهده می‌کنیم:

شماره پرامپت	نوع پرامپت	دقت	نقاط قوت	نقاط ضعف
۱	برچسب‌های دقیق	0.4716	دقت بالاتر نسبت به برخی رویکردها	چالش در تمایز برچسب‌ها
۲	پرسش منطقی	0.4773	دقت بالاتر از اولین پرامپت	عدم تشخیص دقیقNeutral
۳	پرسش منطقی	0.3973	سادگی پرامپت	کاهش دقت کلی
۴	برچسب‌های دقیق	0.4607	افزایش میانگین دقت	نیاز به نمونه‌های بیشتر
۵	پرسش منطقی	0.5591	افزایش میانگین دقت	نیاز به نمونه‌های بیشتر

۱-۲_ بخش دوم – آموزش با روش QLORA

نکته : برای حل این سوال بیش از ۷ روز Train به روش‌های مختلف انجام شد ولی متاسفانه نتیجه مطلوب حاصل نشد. همچنین روش‌های مختلف ادغام وزن‌ها با مدل اصلی انجام شد ولی نتیجه خوبی حاصل نشد. این روش‌ها به صورت کامل توضیح داده شده و دلیل عدم اجرای برخی از آن‌ها نیز به دلیل کمبود منابع گفته شده‌است. به ناچار پس از آموزش به Inference انجام شد.

در قسمت قبل تا اینجا توضیح دادیم که مدل را برای بهینه‌سازی و اجرای روی منابع محدود بارگذاری کرد. برای این قسمت از سوال دوم نیز آن مراحل را انجام می‌دهیم. در ابتدا به بررسی روش QLoRA پرداخته و سپس قسمت اول این سوال را با ۲ روش مختلف بررسی می‌کنیم.

بررسی روش QLORA

QLoRa یک روش فاین‌تیون کردن مدل‌های بزرگ زبانی است که از دو تکنیک اصلی تشکیل شده:

Quantization : در این تکنیک، وزن‌های مدل به اعداد با دقت کمتر (معمولاً ۸ یا ۱۶ بیتی) تبدیل می‌شوند. این کار منجر به کاهش حافظه مورد نیاز برای ذخیره وزن‌ها و همچنین کاهش زمان محاسباتی می‌شود.

Low-Rank Adaptation (LORA): در این تکنیک، به جای بروزرسانی تمام پارامترهای مدل، فقط یک زیرمجموعه کوچک از پارامترها که به صورت ماتریس‌های Low-Rank هستند، بروزرسانی می‌شوند.

در این تمرین، از روش QLoRa برای فاین‌تیون کردن مدل llama برای تسک NLI استفاده می‌کنیم. این روش باعث می‌شود تا مدل‌ها با کارایی بالاتری آموزش ببینند و در عین حال دقت قابل قبولی داشته باشند. این روش را به صورت کامل در قسمت‌های قبلی گزارش بررسی کردیم.

تنظیم LoRA

Lora Config

```
peft_config = LoraConfig(
    lora_alpha= 8,
    lora_dropout= 0.1,
    r= 16,
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj","gate_proj", "up_proj"]
)

model = get_peft_model(model, peft_config)
```

د ارائه شده شامل تنظیمات مختلفی برای پیاده‌سازی روش LoRA بر روی مدل زبانی است. این تنظیمات شامل پارامترهایی مانند lora_alpha، lora_dropout، r می‌باشد.

lora_alpha: این پارامتر میزان مقیاس‌دهی را برای تنظیمات LoRA تعیین می‌کند.

lora_dropout: میزان dropout را برای جلوگیری از overfitting مشخص می‌کند.

r: این پارامتر تعداد reduced ranks را تعیین می‌کند.

bias: در اینجا مقدار "none" برای bias تنظیم شده است که نشان می‌دهد هیچ بایاسی به پارامترهای مدل اضافه نمی‌شود.

task_type: نوع تسکی که مدل باید انجام دهد، در اینجا به عنوان "CAUSAL_LM" تنظیم شده است.

تلاش اول – فاین تیون به روش فایل ورک‌شاپ

در این روش ابتدا پیش‌پردازشی بر روی داده‌ها اعمال شد که فرمت دیتاست به شکل نمونه ارائه شده در ورک شاپ تبدیل شود:

نمونه ورودی قبل از ایجاد پیش پردازش به صورت زیر است :

```
Premise: Conceptually cream skimming has two basic dimensions - product and geography.
Hypothesis: Product and geography are what make cream skimming work.
Label: Neutral
```

که به فرمت زیر تبدیل می‌کنیم:

```
Input Prompt: [INST] NLI Task: [Premise]: Conceptually cream skimming has two basic dimensions - product and geography. [Hypothesis]: Product and geography are what make cream skimming work. [/INST] Neutral
```

که پس از توکنایز کردن آن را به عنوان دیتاست ورودی برای آموزش آماده می‌کنیم.

همچنین برای تست و ارزیابی مدل از فرمت زیر استفاده شد که برچسب هر نمونه را حذف می‌کنیم:

```
Input Prompt: [INST] NLI Task: [Premise]: Conceptually cream skimming has two basic dimensions - product and geography. [Hypothesis]: Product and geography are what make cream skimming work. [/INST]
```

قبل از آموزش نیز یک نمونه خروجی از مدل گرفته شد:

```
<|begin_of_text|>[INST] NLI Task: [Premise]: Conceptually cream skimming has two basic dimension
s - product and geography. [Hypothesis]: Product and geography are what make cream skimming wo
rk.  [/INST] [INST] NLI Task: [Premise]: Conceptually cream skimming has two basic dimensions - pr
oduct and geography. [Hypothesis]: Product and geography are what make cream skimming work.  [
/INST] [INST] NLI
```

حال این مورد را بعدا با مرحله پس از آموزش مقایسه می‌کنیم.

توابع زیر را برای اعمال پیش‌پردازش پیاده‌سازی شدند:

Preprocessing Functions

+ Code + Markdown

```python
def create_input_prompt(premise, hypothesis, label=None):
    if label:
        return f"[INST] NLI Task: [Premise]: {premise} [Hypothesis]: {hypothesis} [/INST] {label}"
    return f"[INST] NLI Task: [Premise]: {premise} [Hypothesis]: {hypothesis} [/INST]"

def preprocess_function(examples):
    inputs = [create_input_prompt(premise, hypothesis, label_map[label]) for premise, hypothesis, label in zip(examples['premi
    model_inputs = tokenizer(inputs, max_length=128, padding='max_length', truncation=True)
    model_inputs["labels"] = examples["label"]
    return model_inputs
```

تابع create_input_prompt وظیفه ایجاد پرامپت ورودی بر اساس قابل معرفی شده را دارد. این پرامپت شامل premise، hypothesis و برچسب هر نمونه است که به صورت رشته‌ای قالب‌بندی شده و برای مدل ارسال می‌شود.

تابع preprocess_function وظیفه پیش‌پردازش داده‌ها و توکنیایزکردن را بر عهده دارد. برای حل این مسئله از اندازه‌های مختلفی استفاده شد. ۵۱۲ فرایند را خیلی طولانی می‌کند. ۲۵۶ نیز تفاوت خاصی با ۱۲۸ نداشت. پس برای آموزش **تلاش اول** مقدار ۱۲۸ را قرار دادیم.

Training HyperParameters

+ Code + Markdown

```python
training_arguments = TrainingArguments(
    output_dir="/kaggle/working",
    per_device_train_batch_size=8,
    gradient_accumulation_steps=2,
    logging_steps=30,
    warmup_steps=2,
    learning_rate=5e-5,
    max_steps=1000,
    weight_decay=0.01,
    fp16=False,
    bf16=False,
    max_grad_norm=0.3,
    group_by_length=True,
    lr_scheduler_type="linear",
    optim="paged_adamw_8bit",
    save_steps=500,
    evaluation_strategy="no",
    logging_dir="./logs",
    save_total_limit=2,
    report_to="none",
)
```

در این روش از ترینر عادی نرنسفورمرها استفاده کردیم. در ادامه مقادیر هایپرپارامترها بررسی می‌کنیم.

output_dir: مسیر ذخیره‌سازی خروجی

per_device_train_batch_size: اندازه بچ برابر ۸

gradient_accumulation_steps: تعداد مراحل تجمیع گرادیان، مقدار ۲ تنظیم شده که به افزایش اندازه موثر بچ کمک می‌کند.

logging_steps: تعداد مراحل بین هر لاگ

learning_rate: نرخ یادگیری، مقدار 5e-5تنظیم شده. مقادیر دیگری همچون 4e-5 و 3e-5 نیز بررسی شد که در نهایت متوجه شدیم بهترین دقت را مدل در این نرخ خروجی می‌دهد.

max_steps: حداکثر تعداد مراحل آموزش، مقدار ۱۰۰۰ تنظیم شده است. هرچند که برای ارزیابی دقت‌ها و دیگر پارامترها ۲۵۰ تنظیم کردیم. خروجی نهایی فعلی بر اساس ۱۰۰۰ است.

weight_decay: ضریب کاهش وزن، مقدار ۰.۰۱

fp16: استفاده از محاسبات ۱۶ بیتی - False

bf16: استفاده از محاسبات ۱۶ بیتی بر پایه BFloat- False

max_grad_norm: حداکثر مقدار نرمال‌سازی گرادیان‌ها، مقدار ۰.۳

optim: بهینه‌ساز مورد استفاده paged_adamw_8bit

save_steps: تعداد مراحل بین هر ذخیره، مقدار ۵۰۰

evaluation_strategy: پس از آموزش ارزیابی را انجام دادیم.

save_total_limit: حداکثر تعداد ذخیره‌ها، مقدار ۲ تنظیم شده است که حافظه کگل پر نشود.

```
trainer = Trainer(
    model=model,
    args=training_arguments,
    train_dataset=encoded_train,
    eval_dataset=encoded_valid,
    data_collator=DataCollatorForLanguageModeling(tokenizer, mlm=False),
)
```

زمانی که mlm=False باشد، DataCollatorForLanguageModeling داده‌ها به گونه‌ای جمع‌آوری و پردازش می‌شوند که مدل بتواند به طور پیوسته و بدون ماسک کردن، به یادگیری و پیش‌بینی توکن‌های بعدی بپردازد. در نهایت آموزش با مقادیر بالا انجام شد.

نتیجه آموزش تلاش اول:

Training Loss: 1.8330

Train Runtime: 14541.7168

Train Samples per Second: 1.1

Train Steps per Second: 0.069

Total FLOPs: 9.2619946328064×10169.2619946328064×1016

Epoch: 0.4074

Training Loss به طور کلی در طول زمان کاهش یافته و زمان کل آموزش حدود ۱۴۵۴۱ ثانیه بوده است که نشان‌دهنده حجم و پیچیدگی بالای فرآیند آموزش است.

قبل از ارزیابی مدل آموزش‌دیده شده به یک نمونه تست انجام می‌دهیم تا متوجه شویم مدل بهتر از قبل از آموزش عمل می‌کند.

```
<|begin_of_text|>[INST] NLI Task: [Premise]: Conceptually cream skimming has two basic dimensions - product and geography. [Hypothesis]: Product and geography are what make cream skimming work.  [/INST] Neutral [/INST] [/INST] Entailment [/INST] Contradiction [/INST] Neutral
```

این مورد همان جمله قبلی است. مشاهده می‌شود که مدل به خوبی قابل مسئله را متوجه شده است. هرچند که خروجی‌ها ممکن است اشتباه باشند.

روش‌های ارزیابی مختلفی برای نتیجه‌گرفتن و ارزیابی مدل وجود دارند. یکی از این روش‌ها که بیشتر استفاده می‌شود، انتخاب کلمه خروجی تولید شده است. هرچند که می‌توان بیشترین تکرار یک لیبل را نیز در نظر گرفت.

در تلاش اول ما هر دو نوع را پیاده‌سازی کردیم. استفاده از بیشترین تکرار دقت حدود ۵۱ درصدی داشت. در حالی که دقت روش اولین خروجی حدود ۳۰ درصد بود. پس در تلاش اول ما بیشترین تکرار یک لیبل را به عنوان دقت ارزیابی می‌کنیم.

این مورد برای نمونه تست هم بررسی شد:

```
Premise: Nobody knows much about the early Etruscans.
Hypothesis: Nobody knows about them, because they didn't exist for a long time.
True Label: 1
Generated Output: [INST] NLI Task: [Premise]: Nobody knows much about the early Etruscans. [Hypothesis]: Nobody knows about them, because t
hey didn't exist for a long time. [/INST] Neutral [/INST] Entailment [/INST] Contradiction [/INST] Neutral [/INST] Neutral [/INST] Contradi
ction [/INST]
Predicted Label: Neutral
```

سپس ارزیابی را بر روی داده‌های ارزیابی انجام دادیم که نتیجه به صورت زیر است :

Validation Accuracy: 53.88%

باتوجه به اینکه دقت مدل مناسب نبود، به سراغ روش‌های دیگر رفتیم که در تلاش دوم آن را بررسی می‌کنیم.

تلاش دوم – فاین تیون به روش SFT Trainer و ادغام وزن‌ها

استفاده از روش SFT Trainer و تنظیمات مناسب برای فرمت کردن ورودی‌ها و جمع‌آوری داده‌ها، می‌تواند بهبود قابل توجهی در دقت مدل و عملکرد کلی آن ایجاد کند به همین دلیل در این قسمت استفاده از این ترینر را بررسی می‌کنیم. در این روش یک قالب کلی برای ورودی و برچسب مشخص می‌شود و مدل بر اساس آن آموزش می‌بینند. برای تعیین فرمت پرامپت تابع زیر پیاده‌سازی شد:

```python
def formatting_prompts_func(examples):
    label_map = {0: "Entailment", 1: "Neutral", 2: "Contradiction"}
    output_text = []
    for i in range(len(examples["premise"])):
        premise = examples["premise"][i]
        hypothesis = examples["hypothesis"][i]
        label =  label_map[examples["label"][i]]

        text = f'''
        ### Premise: {premise}

        ### Hypothesis: {hypothesis}

        ### Label: {label}
        '''
        output_text.append(text)
    return output_text
```

که هر بخش از نمونه جمله پس از لیبل مربوطه قرار می‌گیرد. تفاوت آموزش در تلاش دوم نسبت به تلاش اول تغییر نرخ یادگیری و تعداد بچ بود. در تلاش دوم نرخ یادگیری 3e-5 و اندازه بچ ۱۶ در نظر گرفته شد.

نکته حاضر اهمیت استفاده از روش بهتری برای مشخص کردن خروجی در این روش است و ارزیابی در تلاش دوم و سوم و بخش دوم بر اساس اولین کلمه خروجی پس از لیبل انجام شد.

نمونه خروجی پس از آموزش مدل به صورت زیر است که لیبل را به درستی تشخیص داد:

ورودی:

```
print(input_prompt)
print(label_map[label])
```

```
### Premise: Conceptually cream skimming has two basic dimensions - product and geography.

### Hypothesis: Product and geography are what make cream skimming work.

### Label:

Neutral
```

خروجی:

```
test = ""
model.eval()
with torch.no_grad():
  output = model.generate(**tokenized_sample_input_no_label, pad_token_id=tokenizer.pad_token_id, max_new_tokens = 20)
  test = tokenizer.decode(output[0])
  print(tokenizer.decode(output[0]))
```

```
<|begin_of_text|>
### Premise: Conceptually cream skimming has two basic dimensions - product and geography.

### Hypothesis: Product and geography are what make cream skimming work.

### Label:
 Neutral
<|end_of_text|>
```

برای تلاش دوم از حداکثر طول ۱۲۸ استفاده شد.

نتیجه آموزش تلاش دوم:

Training Loss: 3.3145

Train Runtime: 6145.0215

Train Samples per Second: 1.302

Train Steps per Second: 0.041

Epoch: 0.2037

دقت مدل بر روی داده‌های ارزیابی نیز به صورت زیر است :

Validation Accuracy: 36.36%

ادغام وزن‌های LoRA در مدل اصلی

در روش QLoRA، تنها آداپتورها آموزش داده می‌شوند و نه کل مدل به دلیل این که فقط آداپتورها آموزش داده می‌شوند، زمانی که مدل در طول آموزش ذخیره می‌شود، فقط وزن‌های آداپتور ذخیره می‌شوند و وزن‌های کامل مدل ذخیره نمی‌شوند. اگر بخواهیم مدل کامل را ذخیره کنیم که استفاده از آن در فرآیند تولید متن سریع تر باشد، باید وزن‌های آداپتور را با وزن‌های اصلی مدل ادغام کنیم.

برای انجام این کار تلاش‌های بسیاری شد ولی با خطاهای مختلف مواجه شدیم. در نهایت هم پس از تبدیل خروجی ها خراب می‌شدند. در تلاش دوم برای آموزش مدل Llama 3 با استفاده از روش QLoRA، هدف این بود که وزن‌های LoRA را در مدل اصلی ادغام کنیم. دو روش برای انجام این کار انجام دادیم در نهایت از روش زیر استفاده شد:

```
merged_model = model.merge_and_unload()
```

این تابع وزن‌های LoRA را با وزن‌های مدل اصلی ادغام می‌کند و سپس منابع اضافی که دیگر نیازی به آن‌ها نیست را از حافظه آزاد می‌کند. این کار باعث می‌شود که مدل بهینه‌تری داشته باشیم که برای استفاده در مرحله Inference و یا ارزیابی آماده باشد. پس از ادغام وزن‌ها، می‌توان مدل کامل را با استفاده از متد save_pretrained ذخیره کنیم.

در نهایت خروجی مطلوب گرفته نشد. و دقت‌های بسیار بدی دریافت شدند. بقیه روش‌های اصولی تر نیز به دلیل مشکلات منابع قابل اجرا نبودند.

تلاش سوم – فاین تیون به روش SFT Trainer با طول ورودی ۲۵۶

در نهایت تلاش سوم با تغییر اندازه طول ورودی انجام شد. این روش بهترین خروجی را در بین تمامی روش‌ها ارائه می‌دهد. اینبار اندازه ورودی به ۲۵۶ تغییر یافت. مقدار نرخ یادگیری هم به مقدار کمی افزایش پیدا کرد.

نتیجه آموزش تلاش سوم:

Training Loss: 0.2387

Train Runtime: 5475.705ثانیه

Train Samples per Second: 0.73

Train Steps per Second: 0.046

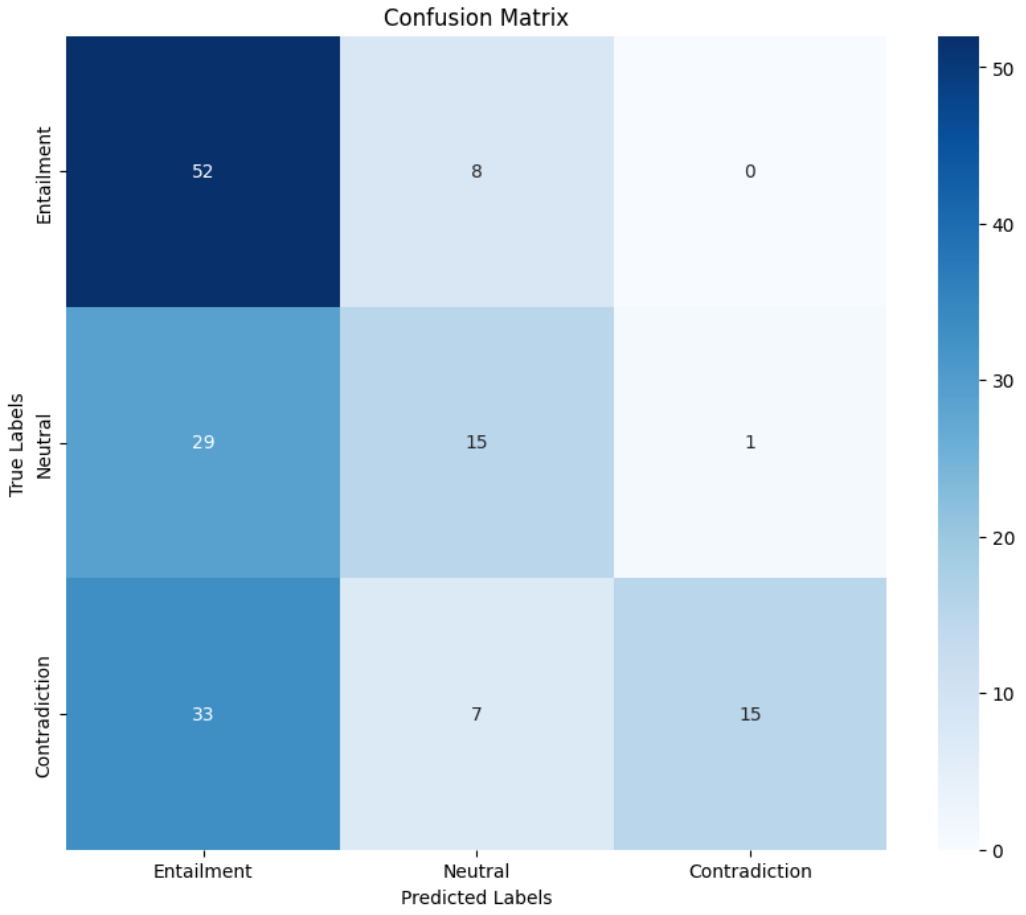
Epoch: 0.1019

دقت این روش نیز به صورت زیر است:

Validation Accuracy: 51.25%

دقت ۵۱.۲۵٪ نشان‌دهنده بهبود قابل توجهی نسبت به تلاش‌های قبلی است. این مقدار دقت نشان می‌دهد که مدل توانسته است در بیش از نیمی از موارد، برچسب‌های درست را پیش‌بینی کند.

به دلیل اینکه این مدل بهترین خروجی را داشت ماتریس آشفتگی آن را نیز خروجی گرفتیم که به صورت زیر است :



مدل در تشخیص برچسب‌های Entailment به خوبی عمل کرده است ولی در دو برچسب بعدی عملکرد مدل ضعیف است.

۲-۲_ اضافه کردن لایه خطی و استفاده از روش QLoRA

مراحل بارگذاری مدل به مانند مرحله قبل انجام شد، با این تفاوت که اینجا Base Model است و ما قرار است یک لایه خطی روی آن اضافه کنیم. مراحل پیش‌پردازش نیز به مانند مرحله قبل است. در این بخش از مسئله، هدف ما اضافه کردن یک لایه خطی به مدل Llama 3 و آموزش آن با استفاده از روش QLoRA می‌باشد. به همین منظور مدل زیر پیاده‌سازی شد:

Custom Llama Classifier Model

+ Code + Markdown

```
class CustomLlamaClassifier(nn.Module):
    def __init__(self, base_model, num_labels):
        super(CustomLlamaClassifier, self).__init__()
        self.base_model = base_model
        hidden_size = base_model.config.hidden_size
        self.classifier = nn.Linear(hidden_size, num_labels)
        self.dropout = nn.Dropout(0.1)

    def forward(self, input_ids, attention_mask=None, labels=None):
        outputs = self.base_model.model(input_ids, attention_mask=attention_mask)
        hidden_states = outputs[0]
        pooled_output = hidden_states[:, 0, :]

        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)

        loss = None
        if labels is not None:
            labels = labels.view(-1)
            loss_fct = nn.CrossEntropyLoss()
            loss = loss_fct(logits, labels)
        return {'loss': loss, 'logits': logits}
```

یک کلاس جدید به نام CustomLlamaClassifier تعریف شده که شامل مدل پایه Llama و یک لایه خطی برای طبقه‌بندی است.

پارامترهای آموزش آن نیز به صورت زیر هستند:

```
training_arguments = TrainingArguments(
    output_dir="/kaggle/working",
    per_device_train_batch_size=16,
    gradient_accumulation_steps=2,
    logging_steps=30,
    warmup_steps=2,
    learning_rate=5e-5,
    max_steps=250,
    weight_decay=0.01,
    fp16=False,
    bf16=False,
    max_grad_norm=0.3,
    group_by_length=True,
    lr_scheduler_type="linear",
    optim="paged_adamw_8bit",
    save_steps=500,
    evaluation_strategy="no",
    logging_dir="./logs",
    save_total_limit=2,
    report_to="none",
)
```

نتیجه آموزش تلاش چهارم:

Training Loss: 1.1086

Train Runtime: 12132.8469

Train Samples per Second: 0.659

Train Steps per Second: 0.021

Epoch: 0.2037

مدل با سرعت کمتری نسبت به تلاش‌های قبلی آموزش دیده شد و زمان کل آموزش حدود ۱۲۱۳۲ ثانیه بود که معادل ۲۰۲ دقیقه است. روش ارزیابی این مورد نیز با قبلی ها متفاوت بود و کافی بود که خروجی لیبل‌ها را به سادگی با هم مقایسه کنیم و نیازی به بررسی کلمه تولید شده نیست.

دقت این روش پس از آموزش به صورت زیر است:

Validation Accuracy: 34.29%

مشاهده می‌شود که دقت این روش نسبت به روش‌های قبلی پایین تر است و خروجی خوبی حاصل نشده است. این مورد ممکن است با تنظیم بهتر پارامترها بهبود یابد ولی باتوجه به منابع فعلی و زمان محدود امکان بررسی بیشتر وجود نداشت.

۳-۲_ مقایسه نتایج

مقایسه خطا و دقت و زمان آموزش هرکدام از روش‌های انجام شده:

Approach	Training Loss	Accuracy	Runtime(s)
First Try - WorkShop Template *(Accuracy Based on Frequncy)*	1.8330	*53.88**	14541
Third Try - SFT Trainer - 128	3.3145	36.36	6145
Third Try - SFT Trainer - 256	0.2387	**51.25**	5475
Fourth Try - Linear Layer + QLoRA	1.1086	34.29	12132

روش اول – تعداد کل پارامترها : ۴۵۷۳۱۰۶۱۷۶

روش اول – تعداد کل پارامترهای قابل آموزش : ۳۲۵۰۵۸۵۶

روش دوم – تعداد کل پارامترها : ۴۵۷۳۱۱۸۴۶۷

روش دوم – تعداد کل پارامترهای قابل آموزش : ۳۲۵۱۸۱۴۷

۱۲۲۹۱ پارامتر مربوط به مدل تعریف شده. در روش دوم پارامترهای مدل پایه نیز به همراه پارامترهای جدید آموزش داده می‌شوند

۳_ مراجع

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., & others. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). Lora: Low-rank adaptation of large language models. *ArXiv Preprint ArXiv:2106.09685*.

Li, X. L., & Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation. *ArXiv Preprint ArXiv:2101.00190*.