به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

درس پردازش زبان طبیعی

پاسخ تمرین ۴

نام و نام خانودگی: زهرا ریحانیان

شماره دانشجویی: ۸۱۰۱۰۱۱۷۷

خرداد ماه ۱۴۰۳

فهرست

پاسخ سوال اول

کد مربوط به این بخش در مسیر codes/Q1.ipynb موجود است.

دادگان

مجموعه استنتاج زبان طبیعی چند ژانر (MultiNLI) مجموعه ای از ۴۳۳ هزار جفت جمله است که با اطلاعات استنتاج مبتنی بر متن حاشیه نویسی شده است. این corpus طیفی از ژانرهای متن گفتاری و نوشتاری مختلف را پوشش می دهد و از ارزیابی تعمیم متقابل ژانر متمایز پشتیبانی می کند. MultiNLI ساخته شد تا ارزیابی صریح مدل‌ها را هم بر اساس کیفیت بازنمایی جملاتشان در حوزه آموزشی و هم در مورد توانایی آن‌ها برای استخراج بازنمایی‌های معقول در حوزه‌های ناآشنا را ممکن سازد.

این corpus برای تسک NLI (natural language inference) کاربرد دارد و هر رکورد آن دارای ۳ جز اصلی دارد: فرض، فرضیه و برچسب. فرض و فرضیه دو جمله هستند و برچسب برای طبقه بندی ارتباط دو جمله است: 0 دلالت، 1 خنثی و 2 تضاد. داده هایی هم که برچسب ندارند با برچسب 1- مشخص می شوند.

پاسخ بخش اول- روش های فاین تیون

۱- در اینجا توضیح مختصری از دو روش سنتی آورده شده است:

۱. تنظیم دقیق همه پارامترهای مدل: این رویکرد شامل تنظیم تمام پارامترهای یک مدل از پیش آموزش دیده است. این مدل که ویژگی‌های کلی را از یک مجموعه داده بزرگ (مانند ImageNet) آموخته است، بیشتر بر روی یک مجموعه داده کوچکتر و مختص تشک آموزش داده می‌شود. در طول این فرآیند، وزن کل شبکه به‌روزرسانی می‌شود تا با تسک مورد نظر مطابقت بیشتری داشته باشد.

۲. تنظیم دقیق یک یا چند لایه: این روش به جای به روز رسانی تمام پارامترهای مدل، بر تنظیم دقیق تنها چند لایه آخر یا هر لایه خاص از مدل تمرکز می کند. ایده این است که لایه‌های اولیه ویژگی‌های عمومی (مانند لبه‌ها و بافت‌ها) را که برای کارهای مختلف مفید هستند، ثبت می‌کنند، در حالی که لایه‌های آخر ویژگی‌های خاص‌تری را ثبت می‌کنند. تنها با تنظیم دقیق لایه‌های آخر، مدل با تسک جدید با هزینه محاسباتی کمتری سازگار می‌شود.

LoRA رویکرد متفاوتی را برای تنظیم دقیق مدل‌های بزرگ پیشنهاد می‌کند، که مخصوصاً در هنگام برخورد با مدل‌های بسیار بزرگ مانند GPT-3 مفید است. تفاوت بین وزن های مدل از پیش آموزش دیده و تنظیم دقیق شده اغلب دارای رتبه ذاتی پایین می باشند ازین رو لازم نیست تمام وزن های مدل آموزش ببینند و به عبارت دیگر، می توان وزن های مدل تنظیم دقیق شده را به طور مؤثر با یک ماتریس با رتبه پایین تقریب زد. به همین خاطر LoRA به جای تنظیم دقیق همه پارامترها، وزن های مدل از پیش آموزش دیده را freeze می کند و ماتریس های low-rank قابل آموزش را معرفی می کند که به هر لایه از معماری

ترانسفورماتور تزریق می شود. اندازه این ماتریس ها در مقایسه با مجموعه کامل پارامترها بسیار کوچکتر است، بنابراین تعداد پارامترهایی که باید برای انطباق با تسک مورد نظر آموزش داده شوند بسیار کاهش می یابد. این روش نه تنها باعث صرفه جویی در منابع محاسباتی می شود، بلکه عملکرد مدل را در کارهای خاص حفظ یا حتی بهبود می بخشد.

۲- روش های پرامپت:

Hard prompt: پرامپت های متنی دست ساز با توکن های ورودی گسسته هستند. نکته منفی این است که برای ایجاد یک پرامپت خوب به تلاش زیادی نیاز دارد.

Soft prompt: تنسورهای قابل یادگیری هستند که با جاسازی‌های ورودی به هم پیوسته‌اند که می‌توانند به یک مجموعه داده بهینه شوند. نکته منفی این است که آنها برای انسان قابل خواندن نیستند زیرا شما این "نشانه های مجازی" را با جاسازی یک کلمه واقعی مطابقت نمی دهید.

پاسخ بخش دوم- آموزش مدل

برای پیاده سازی کد های این بخش ابتدا ماژول های مورد نیاز را نصب و import کردم. نام مدل و دیتاست مورد نیاز را ذخیره کردم و در هاگینگ فیس login کردم. تابع print_number_of_trainable_model_parameters را تعریف کردم برای این که تعداد پارامتر های قابل آموزش مدل را برگرداند. در مرحله بعد tokenizer مدل را از هاگینگ فیس دانلود کردم و آن را برای tokenize دیتاست، آماده کردم. برای این مراحل از این لینک[1] کمک گرفته ام. با این پیاده سازی tokenize_function و البته تغییراتی دیگر، در کل نتیجه ی بهتری برای قسمت هایی که در ادامه توضیح خواهم داد، گرفتم.

دیتاست را دانلود کردم. پیش پردازش های لازم را روی آن انجام دادم. آن را tokenize کردم، ستون های اضافه را حذف کردم، اندازه batch را ۲ گذاشتم و ستون label را به labels تغییر دادم. چون مدل Roberta با labels کار می کند. این کار را برای دیتاست train و validation mismatched انجام دادم چون این دیتاست از منبعی متفاوت از منبع آموزش است و نتیجه ی حاصل از آن عملکرد مدل را منصفانه بیان خواهد کرد.

۱- به روز رسانی کل پارامتر های مدل

در این بخش ابتدا مدل را از کلاس RobertaForSequenceClassification متد from_pretrained دانلود کردم و تعداد پارامتر های قابل آموزش آن را چاپ کردم:

```
trainable model parameters: 355363844
all model parameters: 355363844
percentage of trainable model parameters: 100.00%
```

همان طور که میبینید تمام پارامتر های مدل را آموزش خواهیم داد که حدود ۳۵۵ میلیون پارامتر است.

[1] https://huggingface.co/docs/peft/main/en/task_guides/ptuning-seq-classification

در ادامه معماری مدل را چاپ کردم:

```
RobertaForSequenceClassification(
  (roberta): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(50265, 1024, padding_idx=1)
      (position_embeddings): Embedding(514, 1024, padding_idx=1)
      (token_type_embeddings): Embedding(1, 1024)
      (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): RobertaEncoder(
      (layer): ModuleList(
        (0-23): 24 x RobertaLayer(
          (attention): RobertaAttention(
            (self): RobertaSelfAttention(
              (query): Linear(in_features=1024, out_features=1024, bias=True)
              (key): Linear(in_features=1024, out_features=1024, bias=True)
              (value): Linear(in_features=1024, out_features=1024, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): RobertaSelfOutput(
              (dense): Linear(in_features=1024, out_features=1024, bias=True)
              (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): RobertaIntermediate(
            (dense): Linear(in_features=1024, out_features=4096, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): RobertaOutput(
            (dense): Linear(in_features=4096, out_features=1024, bias=True)
            (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
  (classifier): RobertaClassificationHead(
    (dense): Linear(in_features=1024, out_features=1024, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=1024, out_features=4, bias=True)
  )
)
```

اجزای اصلی این مدل شامل لایه embedding، یک انکدر و یک classifier است.

برای آموزش آن از trainer ماژول transformers استفاده کردم با این آرگومان ها:

```
training_args = TrainingArguments(
    output_dir=output_dir,
```

```
    save_strategy="epoch",
    num_train_epochs=EPOCHS,
    per_device_train_batch_size=2,
    per_device_eval_batch_size=2,
    logging_steps= 1000,
    learning_rate= 2e-4,
    weight_decay= 0.01,
    fp16= False,
    bf16= False,
    max_grad_norm= 0.3,
    warmup_ratio= 0.3,
    group_by_length= True,
    lr_scheduler_type= "linear",
)
```

که تعداد ایپاک هم برابر ۱ بود. با weight_decay=0.01 نتیجه بهتری نسبت به 0.001 گرفتم. در نهایت مدل را آموزش دادم:

```
trainer = transformers.Trainer(
    model=base_model,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=tokenized_datasets,
    eval_dataset=val_tokenized_datasets,
    data_collator=transformers.DataCollatorWithPadding(tokenizer),
)
base_model.train()
trainer.train()
```

آموزش آن حدود یک ساعت و ۱۶ دقیقه طول کشید. نتیجه ارزیابی مدل:

```
{'eval_loss': 1.0936452150344849,
 'eval_accuracy': 0.3540183112919634,
 'eval_runtime': 11.9329,
 'eval_samples_per_second': 82.377,
 'eval_steps_per_second': 41.231,
 'epoch': 1.0}
```

دقت آن حدود ۳۵ درصد شد. این یعنی مدل نیاز به آموزش بیشتری دارد و ۱ ایپاک کافی نبود. در ادامه با روش دیگری مدل را فاین تیون می کنیم.

۲- استفاده از LORA برای فاین تیون مدل

در این بخش دوباره مدل را لود کردم. برای اعمال LORA، از کتابخانه peft استفاده کردم و آن را این گونه تنظیم کردم:

```
peft_config = LoraConfig(
    lora_alpha= 8,
    lora_dropout= 0.1,
```

```
    r= 16,
    bias="none",
    task_type="SEQ_CLS"
)
```

تعداد پارامتر های مدل بعد از اعمال LORA که قرار است آموزش ببینند:

```
trainable model parameters: 2626564
all model parameters: 357990408
percentage of trainable model parameters: 0.73%
```

همان طور که میبینید تعداد پارامتر های قابل آموزش خیلی کاهش یافت و به 0.73 درصد رسید!

معماری مدل:

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

```
              )
              (lora_A): ModuleDict(
                (default): Linear(in_features=1024, out_features=16, bias=False)
              )
              (lora_B): ModuleDict(
                (default): Linear(in_features=16, out_features=1024, bias=False)
              )
              (lora_embedding_A): ParameterDict()
              (lora_embedding_B): ParameterDict()
            )
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): RobertaSelfOutput(
            (dense): Linear(in_features=1024, out_features=1024, bias=True)
            (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): RobertaIntermediate(
          (dense): Linear(in_features=1024, out_features=4096, bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): RobertaOutput(
          (dense): Linear(in_features=4096, out_features=1024, bias=True)
          (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
  (classifier): ModulesToSaveWrapper(
    (original_module): RobertaClassificationHead(
      (dense): Linear(in_features=1024, out_features=1024, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
      (out_proj): Linear(in_features=1024, out_features=4, bias=True)
    )
    (modules_to_save): ModuleDict(
      (default): RobertaClassificationHead(
        (dense): Linear(in_features=1024, out_features=1024, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (out_proj): Linear(in_features=1024, out_features=4, bias=True)
      )
    )
  )
 )
)
```

در ادامه با همان تنظیمات trainer (با این تفاوت که مدل و مسیر خروجی آن متفاوت است) مدل را فاین تیون کردم. حدود ۲۸ دقیقه برای آموزش آن صرف شد. نتیجه ارزیابی مدل:

```
{'eval_loss': 0.7070769667625427,
 'eval_accuracy': 0.8677517802644964,
 'eval_runtime': 14.3591,
 'eval_samples_per_second': 68.458,
 'eval_steps_per_second': 34.264,
 'epoch': 1.0}
```

همان طور که میبینید، دقت مدل به طرز قابل توجهی بهتر شد و به ۸۶ درصد رسید.

۳- مقایسه روش ۱ و ۲

دقت و سرعت روش دوم بالاتر بود، چون مثل روش ۱، تمام پارامتر های مدل را دوباره آموزش نمی دهد بلکه کار آموزش برای تسک جدید را روی ماتریس های رتبه پایین تر انجام می دهد که چون تعداد آن کمتر است، سریع تر آموزش میبینند. (مقایسه بیشتر در قسمت چرا LORA)

۴- استفاده از P-Tuninng

در این قسمت از روش P-Tuning که نوعی روش soft-prompt محسوب می شود، استفاده شده است و از لینک داده شده برای پیاده سازی آن استفاده شده است. تنظیمات peft برای این مدل:

```
peft_config = PromptEncoderConfig(task_type="SEQ_CLS",
                                  num_virtual_tokens=20,
                                  encoder_hidden_size=128,
                                  encoder_dropout=0.1)
model_p = AutoModelForSequenceClassification.from_pretrained(
    model_name,
    return_dict=True,
    num_labels=4
)

model_p = get_peft_model(model_p, peft_config)
```

تعداد پارامتر هایی که قرار است آموزش ببینند:

```
trainable model parameters: 1353988
all model parameters: 356717832
percentage of trainable model parameters: 0.38%
```

همان طور که مشاهده می شود، تعداد پارامتر های مدل حتی از روش قبل هم کمتر شد و تعداد 0.38 درصد از پارامتر های مدل قرار است آموزش ببینند.

معماری مدل:

```
PeftModelForSequenceClassification(
 (base_model): RobertaForSequenceClassification(
  (roberta): RobertaModel(
   (embeddings): RobertaEmbeddings(
    (word_embeddings): Embedding(50265, 1024, padding_idx=1)
    (position_embeddings): Embedding(514, 1024, padding_idx=1)
    (token_type_embeddings): Embedding(1, 1024)
    (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
   )
   (encoder): RobertaEncoder(
    (layer): ModuleList(
     (0-23): 24 x RobertaLayer(
      (attention): RobertaAttention(
       (self): RobertaSelfAttention(
        (query): Linear(in_features=1024, out_features=1024, bias=True)
        (key): Linear(in_features=1024, out_features=1024, bias=True)
        (value): Linear(in_features=1024, out_features=1024, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
       )
       (output): RobertaSelfOutput(
        (dense): Linear(in_features=1024, out_features=1024, bias=True)
        (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
       )
      )
      (intermediate): RobertaIntermediate(
       (dense): Linear(in_features=1024, out_features=4096, bias=True)
       (intermediate_act_fn): GELUActivation()
      )
      (output): RobertaOutput(
       (dense): Linear(in_features=4096, out_features=1024, bias=True)
       (LayerNorm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
       (dropout): Dropout(p=0.1, inplace=False)
      )
     )
    )
   )
  )
  (classifier): ModulesToSaveWrapper(
   (original_module): RobertaClassificationHead(
    (dense): Linear(in_features=1024, out_features=1024, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=1024, out_features=4, bias=True)
   )
   (modules_to_save): ModuleDict(
    (default): RobertaClassificationHead(
     (dense): Linear(in_features=1024, out_features=1024, bias=True)
     (dropout): Dropout(p=0.1, inplace=False)
     (out_proj): Linear(in_features=1024, out_features=4, bias=True)
```

```
    )
   )
  )
 )
 (prompt_encoder): ModuleDict(
  (default): PromptEncoder(
   (embedding): Embedding(20, 1024)
   (mlp_head): Sequential(
    (0): Linear(in_features=1024, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=128, bias=True)
    (3): ReLU()
    (4): Linear(in_features=128, out_features=1024, bias=True)
   )
  )
 )
 (word_embeddings): Embedding(50265, 1024, padding_idx=1)
)
```

در ادامه مدل را با همان تنظیمات قبلی trainer یعنی با همان هایپرپارامتر ها آموزش دادم که حدود ۲۰ دقیقه برای ۱ ایپاک زمان برد.

نتیجه ارزیابی مدل:

```
{'eval_loss': 1.1008752584457397,
 'eval_accuracy': 0.3326551373346897,
 'eval_runtime': 14.296,
 'eval_samples_per_second': 68.76,
 'eval_steps_per_second': 34.415,
 'epoch': 1.0}
```

دقت مدل ۳۳ درصد شد که این حتی از دقت روش اول هم کمتر شد. احتمالا باید برای تعداد ایپاک های بیشتری آموزش ببیند تا دقت آن بهتر است. در اینجا برای این که بتوان نتایج را مقایسه کرد، برای همه ی روش ها تعداد ایپاک برابر ۱ در نظر گرفتم که با این تعداد ایپاک، مدل با روش LORA بهتر فاین تیون شد و از همه دقت بالاتری گرفت.

پاسخ بخش سوم- چرا LORA؟

- در تنظیم دقیق سنتی، کل پارامترهای مدل در طول آموزش یک کار خاص به روز می شوند اما در LORA به جای به روز رسانی تمام پارامترهای مدل در حین تنظیم دقیق، وزن های مدل اصلی منجمد می شود و تغییرات را در مجموعه جداگانه ای از وزن ها (به نام "آداپتورها") اعمال می شود. LoRA پارامترهای مدل را به بعد رتبه پایین تر تبدیل می کند و تعداد پارامترهایی را که نیاز به آموزش دارند کاهش می دهد. این باعث سرعت بخشیدن به فرآیند و کاهش هزینه های محاسباتی می شود.

- روش های سنتی معمولاً شامل بارگذاری یک مدل از پیش آموزش دیده (به عنوان مثال، RoBERTa) و سپس آموزش بیشتر آن بر روی یک مجموعه داده خاص برای یک کار خاص، مانند تجزیه و تحلیل احساسات یا پاسخ به سؤال است اما در روش LORA تنها با به روز رسانی تعداد کمی از پارامترهای اضافی، امکان اصلاح رفتار مدل را فراهم می کند.

- تنظیم دقیق در روش سنتی برای هر کار به منابع محاسباتی قابل توجهی نیاز دارد اما در روش LORA تعداد پارامترهایی که باید برای هر کار آموزش داده و ذخیره شوند را به میزان قابل توجهی کاهش می دهد.

اجرای چند وظیفه بدون فاین تیون:

- تنظیم دقیق سنتی: بدون تنظیم مجدد برای هر وظیفه، عملی نیست. یادگیری چند وظیفه ای می تواند کمک کند، اما همچنان شامل آموزش تمام پارامترهای مدل به طور مشترک است.

- روش LoRA عملی و کارآمد است. با استفاده از آداپتورهای ویژه وظیفه، پارامترهای مدل اصلی دست نخورده باقی می‌مانند و تنها ماتریس‌های کوچک رتبه پایین، مختص وظیفه هستند.

به طور کلی، این کار در روش سنتی امکان پذیر نیست چون استفاده از یک مدل برای کارهای مختلف بدون تنظیم دقیق احتمالاً منجر به عملکرد ضعیف خواهد شد زیرا مدل برای هیچ کاری تخصصی نخواهد بود و در این روش پارامتر های اصلی مدل دوباره آموزش میبینند اما روش LOAR به گونه ای طراحی شده است که چنین انعطاف پذیری را فراهم کند. مدل پایه به عنوان یک پایه مشترک عمل می کند و آداپتورهای مخصوص وظیفه، تخصص لازم را فراهم می کنند.

پاسخ سوال دوم

کد مربوط به این بخش در مسیر codes/Q2.ipynb موجود است.

قبل از حل این سوال نیاز به پیش پردازش هایی بود که در اینجا توضیح خواهم داد.

اول از همه ماژول ها و کتابخانه های مورد نیاز را نصب و import کردم. نام مدل و دیتاست را تعیین کردم و در هاگینگ فیس لاگین کردم. در قدم بعدی tokenizer مربوطه را دانلود کردم و بعد از آن، دیتاست خواسته شده را دانلود کردم. در این سوال از ۱۰٪ دیتاست train و ۴۰٪ دیتاست validation استفاده شده است و با این تعداد دیتاست validation حدود یک ساعت برای ارزیابی زمان مصرف می شود. من برای ارزیابی، دیتاست validation mismatched را دانلود کردم، چون این دیتاست از منبعی متفاوت از منبع آموزش است و نتیجه ی حاصل از آن عملکرد مدل را منصفانه بیان خواهد کرد.

پاسخ بخش اول- ICL

در اینجا چند تابع را تعریف کردم که در ادامه به آنها نیاز خواهیم داشت.

تابع evaluate را برای نمایش دقت و گزارش کلاس بندی با توجه به دو آرگومان y_true که برچسب های درست و y_pred که برچسب های پیش بینی شده هستند، پیاده سازی کردم. قسمت generate prompt را در قسمت های prompting توضیح خواهم داد. متد predict کار اصلی برای ارزیابی مدل بر روی داده ی validation انجام می دهد. به این صورت که به ازای هر

داده ی validation، مقدار premise و hypothesis آن را می گیرد و prompt مورد نظر را با توجه به تابعی که در آرگومان ورودی دریافت کرده، می سازد. آن را tokenize کرده و به مدل مورد نظر که آن را هم در آرگومان ورودی دریافت کرده، می دهد. پارامتر های مدل شامل temperature و max_new_tokens را نیز در آرگومان ورودی دریافت می کند و به مدل می دهد. در نهایت خروجی مدل را decode می کند و برای پیدا کردن برچسب پیش بینی شده، آن را بر اساس = جدا می کند (با توجه به پرامپت هایی که ساخته شد، توکن = را برای جداسازی انتخاب کردم) و i امین آن را انتخاب می کند (i به صورت پیش فرض برابر 1- است) یعنی جواب تولید شده از مدل را می گیرد و اولین عددی که مدل تولید کرده را به عنوان برچسب پیش بینی شده، ذخیره می کند. این کار را با کتابخانه regex انجام می دهد. اگر هم عددی تولید نکرد، مقدار 1- یعنی تعیین نشده را ذخیره می کند. در نهایت تابع برچسب های صحیح و پیش بینی شده را بر میگرداند.

مدل llama3-8b را با تنظیمات Quantization دانلود کردم. چون مدل حدود ۸ میلیارد پارامتر دارد و باید با Quantization مقدار حافظه مورد نیاز آن را کاهش داد که بتوان با وجود محدودیت های حافظه، از آن در تسک موردنیاز استفاده نمود.

در ادامه برای هر روش prompting خواسته شده، توضیحات مربوطه ارائه می گردد.

Zero shot prompting:

برای آموزش به این روش، از پرامپت با ساختار زیر استفاده کردم:

```
Classify the following sentence pairs into entailment (0), neutral (1),
contradiction (2).
{premise} SEP {hypothesis}
Label=
```

که تابع generate_prompt آن را با توجه به premise و hypothesis دریافتی، تولید می کند.

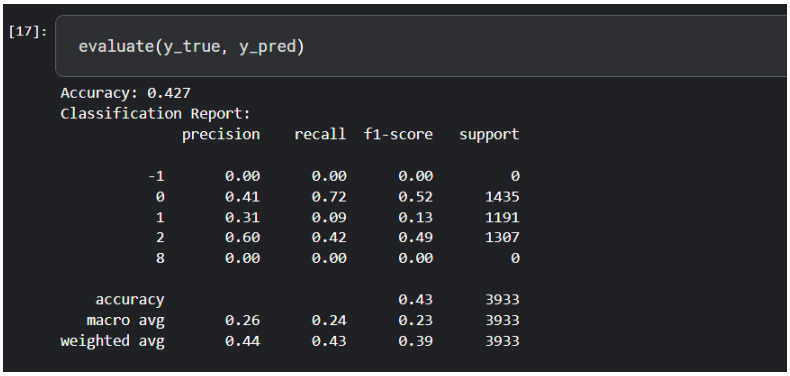
با این پرامپت، مدل ترغیب به تولید توکن و پیش بینی برچسب می شد. برای مدل، این فراپارامتر ها برای تولید توکن استفاده شد:

```
max_new_tokens=16
temprature=0.1
do_sample=True
pad_token_id=tokenizer.eos_token_id
```

max_new_tokens حداکثر تعداد توکنی که مدل باید تولید کند را مشخص می کند. چون وظیفه ما مربوط به کلاس بندی است و مدل فقط باید برچسب پیش بینی کند، آن را ۱۶ گذاشتم. خروجی مدل هم بررسی کردم، معمولا یک عدد به همراه توضیح یا تکرار دوباره جمله داده شده را خروجی می دهد.

temperature تصادفی بودن متن تولید شده را کنترل می کند. دمای پایین تر متن قابل پیش بینی بیشتری تولید می کند، در حالی که دمای بالاتر متن خلاقانه و غیرمنتظره تری تولید می کند. چون وظیفه مربوطه پیش بینی برچسب است، پس به خلاقیت زیادی نیاز ندارد و آن را ۰.۱ گذاشتم.

تابع predict را فراخوانی کردم (توضیحات آن بالاتر گفته شد). حدود یک ساعت و ۷ دقیقه زمان برد تا مدل روی دیتاست ارزیابی شود. نتیجه به صورت زیر حاصل شد:



```
[17]: evaluate(y_true, y_pred)

Accuracy: 0.427
Classification Report:
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.41      0.72      0.52      1435
           1       0.31      0.09      0.13      1191
           2       0.60      0.42      0.49      1307
           8       0.00      0.00      0.00         0

    accuracy                           0.43      3933
   macro avg       0.26      0.24      0.23      3933
weighted avg       0.44      0.43      0.39      3933
```

همان طور که مشاهده می شود، دقت حدود ۴۳ درصد و f1-score درحالت macro avg برابر ۲۳ درصد شد.

One shot prompting:

برای این حالت نیاز به انتخاب یک نمونه از دیتاست آموزش داریم. استفاده از یک نمونه ساده و ترجیحا کوتاه، موجب می شود مدل بهتر وظیفه را درک کند و در نتیجه خروجی بهتری تولید کند. به همین خاطر من به طور دستی داده های ابتدایی دیتاست آموزش را مورد بررسی قرار دادم و چند تا کاندید انتخاب کردم و در نهایت داده ۴۴ را انتخاب کردم که هم ساده و کوتاه بود، هم موجب شده بود مدل بهتر خروجی دهد. ساختار پرامپت در این جا به صورت زیر تعیین شد:

```
Classify the following sentence pairs into entailment (0), neutral (1),
contradiction (2):
{premise0} SEP {hypothesis0}
Label = {label0}
Now classify the following pairs:
{premise} SEP {hypothesis}
Label =
```

که توسط تابع generate_prompt_one_shot تولید می شود. premise0 و hypothesis0 و label0 برای نمونه آموزشی و premise و hypothesis از داده validation هستند.

فراپارامتر ها همان فراپارامتر های مدل در حالت zero-shot را تنظیم کردم (با همان دلایل) که بتوان نتایج را در شرایطی مساوی مورد بررسی و مقایسه قرار داد.

همانند قبل تابع predict را فراخوانی کردم. اینجا i=2 قرار دادم. چون دو تا توکن = در پرامپت ورودی داریم و از split دوم به بعد خروجی مدل است که اولین عدد خروجی آن را به عنوان پیش بینی اش محسوب می کنیم. حدود یک ساعت و ۴۲ دقیقه زمان برد تا مدل روی دیتاست ارزیابی شود. نتیجه به صورت زیر حاصل شد:

```
evaluate(y_true, y_pred)
```

```
Accuracy: 0.457
Classification Report:
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.68      0.03      0.05      1435
           1       0.35      0.88      0.50      1191
           2       0.78      0.55      0.64      1307

    accuracy                           0.46      3933
   macro avg       0.45      0.36      0.30      3933
weighted avg       0.62      0.46      0.39      3933
```

همان طور که مشاهده می شود، دقت حدود ۴۶ درصد و f1-score درحالت macro avg برابر ۳۰ درصد شد که نسبت به حالت zero-shot مدل کمی بهتر عمل کرده است. علت این اتفاق این است که مدل با دیدن یک نمونه، به نوعی آموزش میبیند که باعث می شود خروجی مورد انتظار را تولید کند و در نتیجه دقت آن بهبود یابد.

پاسخ بخش دوم- آموزش مدل با استفاده از روش QLoRA

رویکرد QLora(Quantized Low-Rank Adaptes) روشی است که برای تنظیم دقیق مدل‌های زبان بزرگ با استفاده از تکنیک‌های Quantization و انطباق با رتبه پایین طراحی شده است. تنظیم دقیق مدل‌های زبان بزرگ (LLM) مانند مدل‌های مبتنی بر معماری GPT می‌تواند از نظر محاسباتی گران و حافظه فشرده باشد. QLoRA این چالش ها را با موارد زیر برطرف می کند:

- کاهش حافظه موردنیاز از طریق کوانتیزاسیون.

- استفاده از آداپتورهای با رتبه پایین برای فاین تیون کارآمد مدل بدون نیاز به به روز رسانی تمام پارامترها.

Quantization تکنیکی است که دقت پارامترهای مدل را کاهش می‌دهد و در نتیجه مصرف حافظه و بار محاسباتی را کاهش می‌دهد. QLoRA معمولاً از یک Quantization ۴ بیتی استفاده می کند که به طور قابل توجهی مدل را فشرده می کند در حالی که بیشتر عملکرد آن را حفظ می کند.

Quantization ۴ بیتی: حافظه مورد نیاز برای ذخیره سازی هر پارامتر را به ۴ بیت کاهش می دهد که یک چهارم فضای مورد نیاز برای دقت ۱۶ بیتی است.

LoRA ماتریس های کوچک و قابل آموزش (آداپتورها) را معرفی می کند که به پارامترهای مدل اصلی اضافه می شوند. این آداپتورها تقریبی با رتبه پایین هستند، به این معنی که پارامترهای آنها به طور قابل توجهی کمتر از مدل اصلی است.

تجزیه رتبه پایین: LoRA از دو ماتریس با رتبه پایین، A و B برای تقریب به روز رسانی وزن W در مدل اصلی استفاده می کند. به‌روزرسانی توسط W+ A.B ارائه می‌شود، که در آن A و B رتبه‌های بسیار پایین‌تری نسبت به W دارند.

QLora ترکیبی از این دو روش یعنی Quantization و Lora را استفاده می کند. یعنی مدل زبانی بزرگ ابتدا با دقت ۴ بیت کوانتیزه می شود و حافظه مورد نیاز برای ذخیره و پردازش مدل را کاهش می دهد. سپس آداپتورهای رتبه پایین به مدل کوانتیزه شده معرفی می شوند. این آداپتورها تنها اجزایی هستند که در طول فرآیند فاین تیون آموزش دیده اند. در طول تنظیم دقیق، پارامترهای کوانتیزه اصلی ثابت نگه داشته می شوند و فقط آداپتورهای رتبه پایین به روز می شوند. این به شدت تعداد پارامترهایی را که باید آموزش داده شوند کاهش می‌دهد و منجر به تنظیم دقیق‌تر و کارآمدتر می‌شود.

آموزش مدل با این روش:

برای Quantization مدل را با این تنظیمات دانلود و آماده می کنیم:

```
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=False,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=compute_dtype,
)

model = AutoModelForCausalLM.from_pretrained(
    model_name,
    device_map=device,
    torch_dtype=compute_dtype,
    quantization_config=bnb_config,
)
```

یعنی llama3 را به عنوان یک causal language model که توانایی پیش بینی کلمه بعدی را دارد و از قبل آموزش دیده، با تنظیمات Quantization ۴ بیتی دانلود و ذخیره می کنیم. آن را برای kbit training آماده می کنیم و تنظیمات Lora را با استفاده از کتابخانه peft به آن اعمال می کنیم. تنظیماتی که استفاده شد:

```
peft_config = LoraConfig(
    lora_alpha= 16,
    lora_dropout= 0.1,
    r= 64,
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj","gate_proj",
"up_proj"]
)
```

lora_alpha: نرخ یادگیری برای ماتریس های به روز رسانی LoRA.

lora_dropout: احتمال انصراف برای ماتریس های به روز رسانی LoRA.

r: رتبه ماتریس های به روز رسانی LoRA.

bias: نوع bias برای استفاده. مقادیر ممکن none, additive, learned هستند.

task_type: نوع وظیفه ای که مدل برای آن آموزش می بیند. که در اینجا CAUSAL_LM است.

بعد از اعمال این تنظیمات، تعداد پارامتر های قابل آموزش را چاپ کردم. در اینجا حدود ۱.۵٪ پارامتر های مدل آموزش میبینند. حدود ۱۳۰ میلیون پارامتر:

```
trainable params: 130,023,424 || all params: 8,160,284,672 || trainable%: 1.5934
```

بعد از آن نیاز به آماده سازی داده ها بود، به طوری که بتوان با آن ها مدل را آموزش داد. برای همین من تابع generate_prompt_for_training را روی کل دیتاست آموزش اعمال کردم. این تابع خیلی شبیه generate_prompt برای حالت zero-shot است، با این تفاوت که انتهای آن label درست را اضافه می کند تا مدل در طول آموزش بتواند با استفاده از کلمات قبلی، لیبل صحیح را پیش بینی کند و بر اساس لیبل درست، loss خود را بهبود ببخشد.

در نهایت یک دیتاست با ستون text ساختم که پرامپت برگردانده شده از تابع generate_prompt_for_training را در آن ستون ذخیره کرده است. فرمت پرامپت به شکل زیر است:

```
Classify the following sentence pairs into entailment (0), neutral (1),
contradiction (2).
{example['premise']} SEP {example['hypothesis']}
Label = {example['label']}
```

در نهایت مدل را آموزش دادم. برای آموزش آن از SFTTrainer استفاده کردم با آرگومان های زیر:

```
training_args = TrainingArguments(
    output_dir=output_dir,
    save_strategy="epoch",
    num_train_epochs=EPOCHS,
    per_device_train_batch_size=1,
    gradient_accumulation_steps=4,
    logging_steps= 30,
    max_steps=1000,
    warmup_steps=4,
    warmup_ratio=0.03,
    learning_rate=2e-4,
    weight_decay= 0.001,
    fp16= False,
    bf16= False,
    max_grad_norm= 0.3,
    group_by_length= False,
    optim="paged_adamw_8bit",
```

```
    lr_scheduler_type= "constant",
)
```

output_dir: فهرستی که لاگ های آموزشی و چک پوینت ها در آن ذخیره می شوند.

save_strategy: یعنی هر step ذخیره شود یا هر epoch که من به علت کمبود منابع آن را epoch قرار دادم.

num_train_epochs: تعداد ایپاک برای آموزش مدل که برابر ۱ گذاشتم.

per_device_train_batch_size: تعداد نمونه ها در هر دسته در هر دستگاه که آن را ۱ گذاشتم چون حافظه محدود بود.

gradient_accumulation_steps: تعداد دسته هایی برای جمع آوری گرادیان قبل از به روز رسانی پارامترهای مدل که آن را ۴ گذاشتم.

logging_steps: تعداد مراحلی که پس از آن باید معیارهای آموزشی ثبت شود که آن را ۳۰ گذاشتم.

max_steps: حداکثر تعداد مراحل برای آموزش مدل که من آن را ۱۰۰۰ گذاشتم که مدل حدود یک ساعت آموزش ببیند.

warmup_ratio: نسبت مراحل آموزش برای گرم کردن میزان یادگیری استفاده می شود که آن را ۰.۰۳ گذاشتم.

Learn_rate: نرخ یادگیری برای بهینه ساز که آن را 2e-4 گذاشتم.

weight_decay: پارامتر کاهش وزن برای بهینه ساز که آن را ۰.۰۰۱ گذاشتم.

fp16: آیا از دقت ممیز شناور ۱۶ بیتی استفاده شود یا خیر که آن را False گذاشتم.

bf16: آیا از دقت BFloat16 استفاده شود یا خیر که آن را False گذاشتم.

max_grad_norm: حداکثر norm گرادیان که آن را ۰.۳ گذاشتم.

group_by_length: آیا می توان نمونه های آموزشی را بر اساس طول گروه بندی کرد که آن را False گذاشتم.

optim: بهینه‌سازی برای آموزش مدل که از paged_adamw_8bit استفاده کردم.

lr_scheduler_type: نوع زمانبندی نرخ یادگیری برای استفاده که آن را constant گذاشتم.

همچنین به trainer آبجکت tokenizer, model، dataset_text_field و data_collator را نیز دادم که trainer مدل را با این دیتاست train می کند (در dataset_text_field مشخص می کنیم که کدوم ستون دیتاست را باید پردازش کند) و در نهایت مدل را با استفاده از آن train کردم. حدود یک ساعت و ۳۵ دقیقه آموزش آن زمان برد. بعد از آن با استفاده از متد merge_and_unload وزن های Lora را با وزن های اولیه مدل merge کردم. سپس با استفاده از تابع predict که توضیحات آن بالاتر گفته شد، آن را مورد ارزیابی قرار دادم. حدود ۴۸ دقیقه ارزیابی آن طول کشید. در نهایت این نتایج حاصل شد:

```
[22]: evaluate(y_true, y_pred)

Accuracy: 0.479
Classification Report:
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.73      0.31      0.43      1435
           1       0.39      0.17      0.24      1191
           2       0.46      0.95      0.62      1307

    accuracy                           0.48      3933
   macro avg       0.39      0.36      0.32      3933
weighted avg       0.54      0.48      0.43      3933
```

همان طور که مشاهده می کنید دقت حدود ۴۸ درصد و f1-score درحالت macro avg برابر ۳۲ درصد شد که این از هر دو حالت prompting در قسمت قبل بهتر شده است. چون در قسمت قبل مدل آموزش ندیده بود ولی در اینجا با فاین تیون کردن آن روی بخشی از دیتاست آموزش، مدل کمی بهتر عمل کرده است. احتمالا اگر بیشتر آموزش می دید، نتیجه از این بهتر می شد ولی متاسفانه به خاطر محدودیت زمان، این امر ممکن نشد.

Step	Training Loss
30	2.142200
60	1.678400
90	1.665900
120	1.678800
150	1.649700
180	1.653400
210	1.563000
240	1.655700
270	1.575700
300	1.592400
330	1.634300
360	1.565200
390	1.602300
420	1.614600
450	1.650100
480	1.564000
510	1.608000

شکل ۱ بخشی از loss مدل در زمان آموزش

پاسخ بخش سوم- آموزش مدل مدل با استفاده از روش QLoRA (روش دوم)

در این بخش ابتدا یک کلاس برای مدل خواسته شده تعریف می کنیم. برای این کار یک کلاس به اسم MyConfig که از کلاس PretrainedConfig ارث می برد، برای config این کلاس، تعریف می کنم. کلاس CustomLLamaModel را برای مدل خواسته شده پیاده سازی کردم. به این صورت که config, model, num_labels را می گیرد و این ویژگی ها را ایجاد می کند:

config, num_labels, model, input_dim, dropout, classifier

که model همان llama3 است و آن را هنگام تعریف کلاس از طریق آرگومان هایش دریافت می کند و classifier همان لایه خطی خواسته شده است. آن را در بالای مدل قرار می دهم. یعنی آخرین لایه آن است. متد forward این کلاس به این صورت عمل می کند که input_ids, attention_mask ورودی را به مدل می دهد. خروجی آن را یک hidden_state است که آخرین آن را می گیرد و dropout را روی آن اعمال می کند و نتیجه را به classifierx می دهد. خروجی آن logits است که با استفاده از آن، loss مدل را به روش Cross entopy حساب می کند. در نهایت loss و logits را بر می گرداند.

مدل llama3 را همانند قسمت قبل با تنظیمات Quantization ۴ بیتی دانلود و ذخیره می کنم. سپس یک آبجکت جدید custom_model از کلاس CustomLLamaModel تعریف کردم و config که آبجکت MyConfig است به همراه مدل llama3 و تعداد لیبل ۳ را به آن دادم. از این جا به بعد از این آبجکت custom_model به عنوان مدل استفاده می کنم. همانند قسمت قبل آن را برای kbit training آماده کردم و تنظیمات Lora را با استفاده از کتابخانه peft به آن اعمال کردم. همان تنظیمات قسمت قبل را روی آن اعمال کردم به جز task_type که آن را SEQ_CLS گذاشتم.

تعداد پارامتر های قابل آموزش هم تقریبا شبیه قسمت قبل شد. ۱.۵٪ کل پرامتر های مدل قابل آموزش هستند یعنی حدود ۱۳۰ میلیون پارامتر:

```
trainable params: 130,408,195 || all params: 8,161,054,214 || trainable%: 1.5979
```

در مرحله بعد، لازم بود داده ها را پیش پردازش کنم. در این جا داده ها را tokenize کردم و ستون های اضافه را حذف کردم. و ستون label را به labels تغییر نام دادم. تابعی که برای tokenize استفاده کردم:

```
def tokenize_function(examples):
    outputs = tokenizer(examples["premise"], examples["hypothesis"],
truncation=True, max_length=None)
    return outputs
```

این کار را برای هر دو دیتاست آموزش و ارزیابی انجام دادم. در نهایت DataLoader هر دوی آن ها را ساختم:

```
data_collator=transformers.DataCollatorWithPadding(tokenizer, padding=True,
max_length=120)
train_dataloader = DataLoader(
    tokenized_datasets, shuffle=True, batch_size=1, collate_fn=data_collator
)
eval_dataloader = DataLoader(
    val_tokenized_datasets, batch_size=2, collate_fn=data_collator
)
```

برای اندازه گیری دقت مدل از متد load_metric از کتابخانه datasets استفاده شد.

از بهینه ساز AdamW با نرخ یادگیری lr=2e-5 استفاده کردم. تعداد ایپاک=۱ و تعداد step ها را برابر ۵۰۰۰ و هر ۱۰۰ step مقدار loss را نمایش خواهم داد. هم چنین از زمان بند نرخ یادگیری linear استفاده کردم. در دو حلقه for یکی برای تعداد ایپاک

و دیگری بر روی train_dataloader کار آموزش را انجام دادم: به ازای هر batch داده آموزش، آن را به device ست شده، بردم و حاصل را به مدل دادم. Loss برگردانده شده را گرفتم و روی آن backward را فراخوانی کردم. در ادامه با optimizer پارامتر ها را به روز کردم و lr_scheduler را با متد step به روز رسانی کردم و optimizer.zero_grad() را فراخوانی کردم. در نهایت مدل با دیدن ۵۰۰۰ batch داده، آموزش میبینند. این آموزش حدود یک ساعت و ۸ دقیقه زمان برد. ارزیابی آن را روی دیتاست ارزیابی انجام دادم. در اینجا بعد از دادن batch داده به مدل، خروجی logits آن را میگیرد و argmax آن را به عنوان لیبل پیش بینی شده برمی گرداند. در نهایت دقت مدل روی داده های ارزیابی برابر ۳۳٪ شد. این نشان می دهد که مدل هنوز خوب آموزش ندیده است و لازم است بیشتر آموزش ببیند. متاسفانه این امر به علت محدودیت زمانی و منابع محقق نشد.

مقایسه و جمع بندی:

در جدول زیر مقایسه بین رویکرد های مختلف بر روی دو مدل با معیار های خواسته شده آورده شده است:

مدل و روش فاین تیون	زمان آموزش	دقت	تعداد پارامتر های آموزش دیده
مدل Roberta-large با آموزش کل پارامترها	یک ساعت و ۱۶ دقیقه	۳۵٪	355,363,844
مدل Roberta-large با روش Lora	۲۸ دقیقه	۸۶٪	2,626,564
مدل Roberta-large با روش P-tuning	۲۰ دقیقه	۳۳٪	1,353,988
مدل llama با روش Qlora	یک ساعت و ۳۵ دقیقه	۴۸٪	130,023,424
مدل llama با روش Qlora (بخش ۲)	یک ساعت و ۸ دقیقه	۳۳٪	130,408,195

در مدل Roberta-large برای این که بتوان نتایج را مقایسه کرد، برای همه ی روش ها تعداد ایپاک برابر ۱ در نظر گرفتم که با این تعداد ایپاک، مدل با روش LORA بهتر فاین تیون شد و از همه دقت بالاتری گرفت. در بین همه ی روش های دیگر نیز بهترین نتیجه را کسب کرد. از نظر تعداد پارامتر های آموزش دیده و زمان آموزش هم در اینجا بعد از روش P-tuning قرار می گیرد. بدترین نتایج را برای Roberta-large با آموزش کل و روش p-tuning و مدل llama با روش Qlora بخش ۲ حاصل شد. مدل llama مدل بسیار بزرگی با ۸ میلیارد پارامتر است. حتی بعد از اعمال Qlora باز هم تعداد پارامتر های قابل آموزش بسیار بالاست (حدود ۱۳۰ میلیون). همین هم باعث می شود نیاز به منبع و زمان بیشتری برای آموزش داشته باشد. در نتیجه تعداد پارامتر های زیاد لزوما به معنای افزایش دقت نمی باشد و باید انتخاب مدل را با توجه به تسک و حجم دیتا انجام داد. به نظر می رسد مدل Roberta-large در این تسک که مرتبط با استنتاج است، موفق تر بوده است. مدل llama برای تسک هایی که نیاز به خلاقیت دارند مثلا چت یا داستان گویی موفق تر است.