به نام خدا

گزارش تمرین اول

درس: مدل‌های زبانی بزرگ

نام و نام خانوادگی: الهه بدلی

آبان ۱۴۰۲

۱. نوت بوک اول: Full Fine Tuning

نوت بوک اول را صرفا نیاز بود اجرا کنیم. به نظرم رویکرد بسیار دقیق و خوبی بود که یک نوت بوک صرفا برای آشنایی با کد قرار داده بودید. برای من بسیار مفید بود.

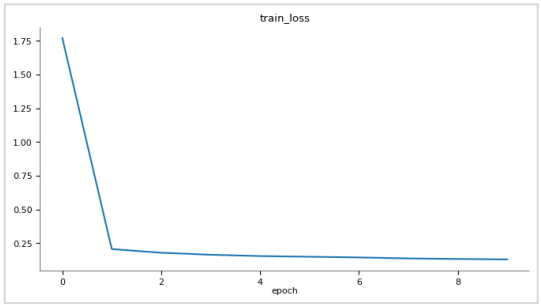
در این تمرین از مجموعه داده IMDB استفاده شده است که مجموعه داده نظرات و احساسات مثبت یا منفی نسبت به آن ها را دارد.

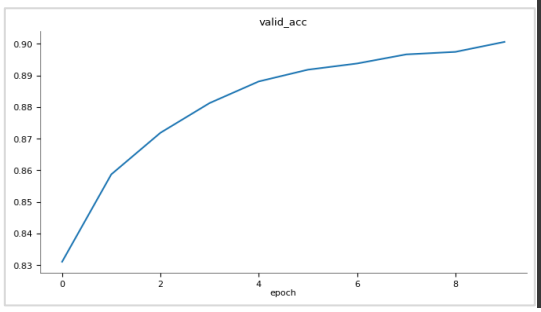
توزیع داده‌ها به این صورت است:

Train	test
25000	25000

در این نوت بوک از مدل T5small استفاده شده است.

وضعیت traning به صورت زیر است. روند train loss و valid accuracy در دو شکل زیر آمده است.





epoch	train_loss	valid_acc
0	1.769300	0.83108
1	0.205696	0.85872
2	0.178165	0.87188
3	0.163573	0.88128
4	0.153670	0.88812
5	0.148679	0.89184
6	0.143487	0.89380
7	0.136211	0.89668
8	0.132390	0.89748
9	0.128704	0.90064

نتایج روش full fine tuning به صورت زیر است:

#Trainable parameters	accuracy	#parameters	method
all	90.1	60506624	Full fine tunning

۲. نوت بوک دوم: Soft prompt

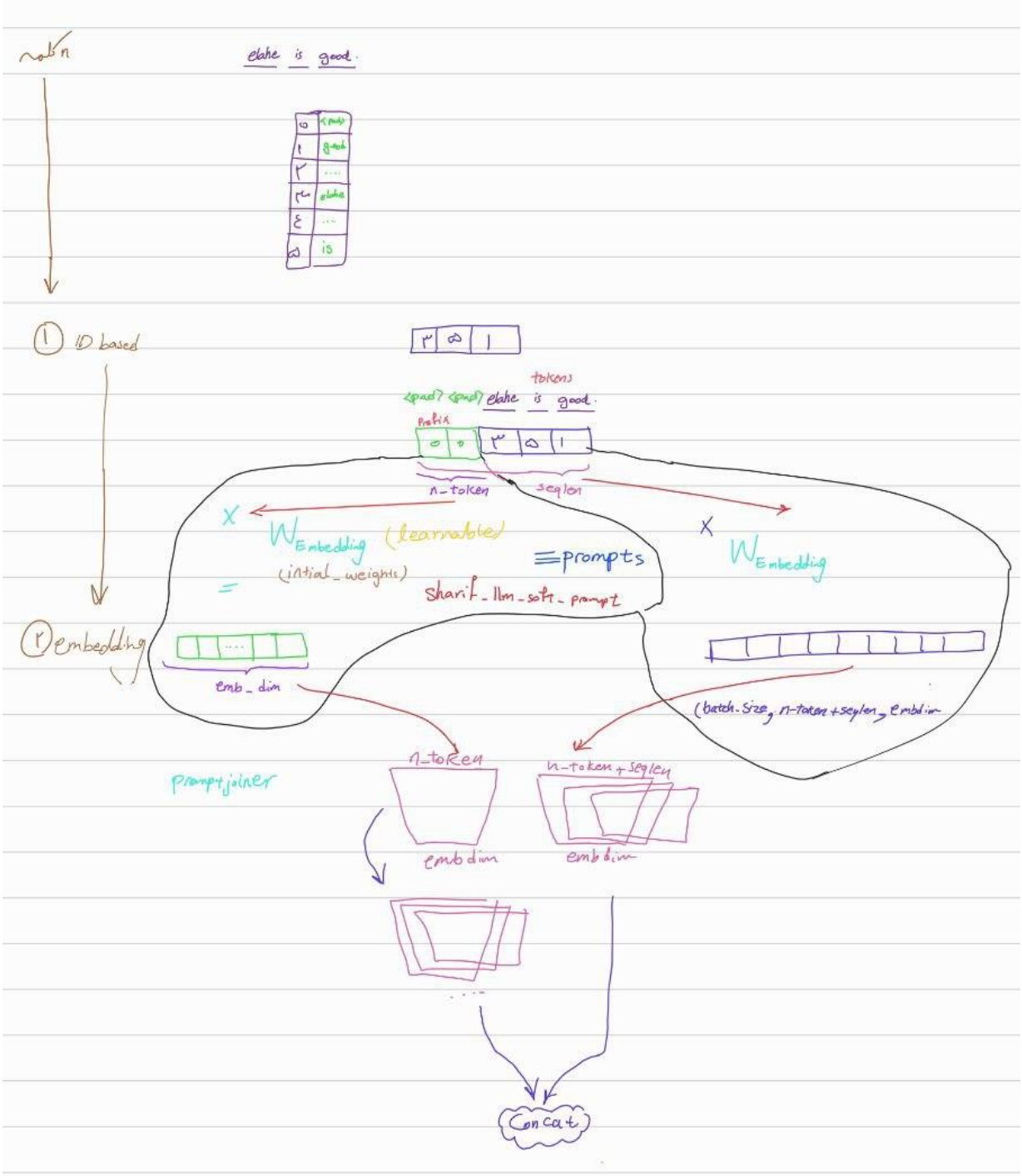
در این نوت‌بوک روش soft prompt tunning استفاده شده است. تعداد ۱۰ prefix با مقدار pad به ابتدای ورودی اضافه شده است. که این کار با خط کد زیر انجام شده است.

```python
def preprocess_input(text):
    text = text.lower()
    text = text.replace('<br />', ' ')
    return text

def preprend_padding_token(text):
    n_soft_prompt_tokens = N_SOFT_PROMPT_TOKENS
    pad_token = tokenizer.pad_token

    ######### Your code begins #########
    prefix = n_soft_prompt_tokens * pad_token
    ######### Your code ends ###########
    return prefix + text

def map_function(row):
    processed_input = [
        preprend_padding_token(preprocess_input(text))
        for text in row['text']
    ]
    input_info = tokenizer(processed_input, truncation=True, max_length=256)
    output_info = tokenizer(id2label(row['label']))
    return {
        **input_info,
        'labels': output_info.input_ids
    }

dataset = dataset.map(map_function, batched=True)
dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'])
```

بنابراین دیتاست به صورت شکل زیر است. می‌بینیم که ۱۰ توکن ابتدایی ۰ هستند که همان نشان دهنده prefix های اضافه شده هستند.

```
{'input_ids': tensor([    0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
           48,   814,    47,  1077,  3555,    57,  8581,   986,    31,     7,
        18236,     6, 28054,    11,     3,    23, 11872,    25,    12,   217,
           24,   814,  1446,     5,     8,   814,    65,   192,  1101,  2479,
           11,   273,    33,     6,  5637,     8,  9555,  6922,  6499,     8,
         4423,     6,    73, 21348,   120,   207,     6,  1202,     5,  3943,
           45,    24,     6,   125, 17613,   140,   167,    19,     8,  9590,
         6093,    13,   108,   195,  6096,     5,    90,    29,     9,     3,
           29,    63,   348,    65,    12,    36,   167, 16241, 15676,    16,
            8,   296,     5,   255,  6775,    78, 13721,    11,    28,    66,
            8,   206,    26,   485,    16,    48,   814,     6,   233,   155,
           31,     7,    73,   144,  6471,  3268,     5,     3, 14622,    12,
         8581,   986,    31,     7,   814,     6,  8445,   485,    65,   118,
         5821,    28, 13721,   485,     5,   406,   352,   396,   623,    30,
           48,  1426,     6,     3,    23,   133,   497,    24,  6963,    45,
            8,  1750,    16,  1523,     7,   344,     8, 20609,    11,     8,
            3,     7,  1123,    26,  1273,  2710,     5,     3,     9,  1974,
           13,   165,    97,     6,    11,   286,     5,   204, 11476,     5,
            1]),
 'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]),
 'labels': tensor([2841,    1])}
```

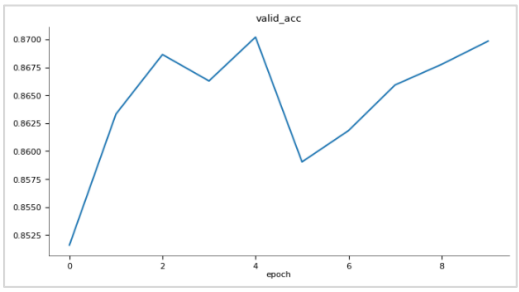
در ادامه تحلیل خودم از نحوه اضافه کردن prefixها را در ادامه در شکل آورده شده است.

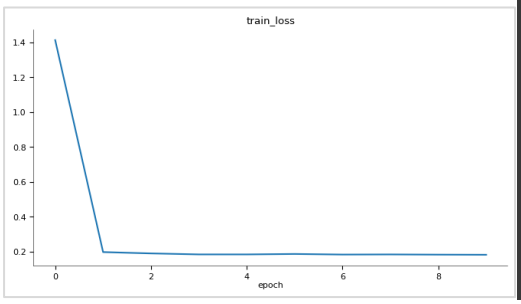


در بخش بعد پارامترهایی که **freeze** نشده اند و قرار است **train** شوند در ادامه آمده است.

```
Non freezed weights:
encoder.embed_tokens.sharif_llm_soft_prompts.prompt_emb
```

روند **training** به صورت زیر است:





epoch	train_loss	valid_acc
0	1.411817	0.85160
1	0.196808	0.86332
2	0.189176	0.86864
3	0.183523	0.86628
4	0.183484	0.87020
5	0.185974	0.85904
6	0.182724	0.86184
7	0.183395	0.86592
8	0.182233	0.86776
9	0.181513	0.86984

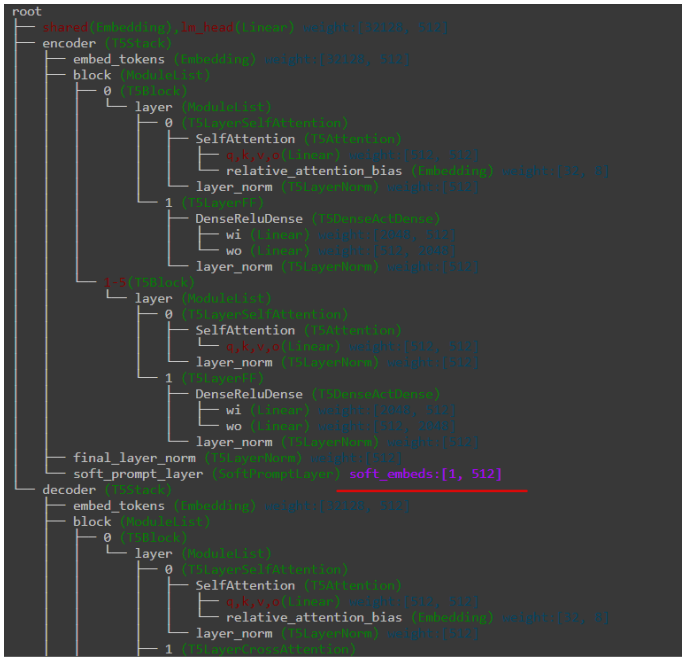
پارامترها نتایج این مدل به شرح زیر است:

#Trainable parameters	accuracy	#parameters	method
5120	87.0	60511744	Soft prompt – main

بخش کتابخانه ای:

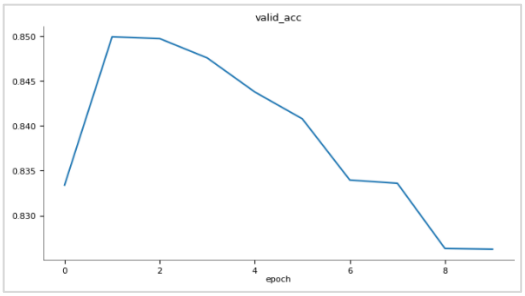
در این بخش از کتابخانه open delta استفاده شده است. مدل بر اساس n_soft_token ۱ و ۱۰ آموزش داده شده است.

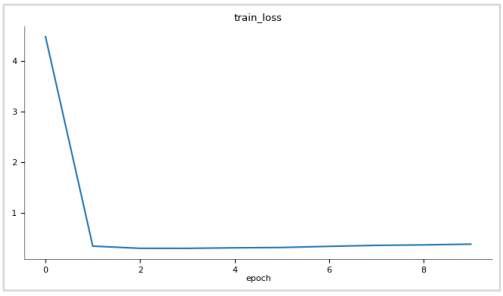
N_soft_token = 1:



روند آموزش به شرح زیر است:

epoch	train_loss	valid_acc
0	4.473649	0.83340
1	0.339410	0.84996
2	0.295714	0.84976
3	0.295343	0.84760
4	0.305809	0.84380
5	0.312423	0.84080
6	0.334522	0.83396
7	0.354233	0.83360
8	0.363573	0.82632
9	0.378416	0.82624





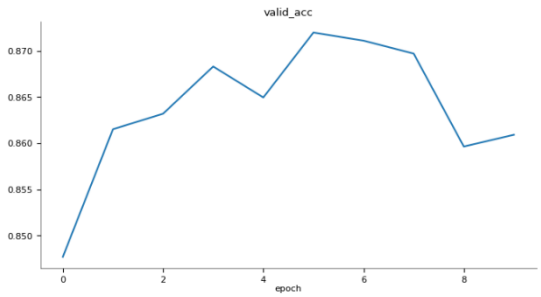
تعداد پارامترها و دقت مدل در ادامه آورده شده است. نکته قابل ذکر این که به دلیل داشتن ۱ ورودی ، تعداد پارامترهای embedding برابر با ۵۱۲ است.

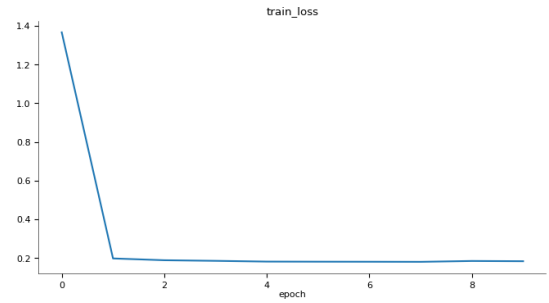
#Trainable parameters	accuracy	#parameters	method
512	84.9	60507136	Soft prompt n_soft_prompt_token = 1

N_soft_tokn = 10:

روند آموزش این مدل به شرح زیر است:

epoch	train_loss	valid_acc
0	1.366570	0.84768
1	0.197127	0.86152
2	0.187921	0.86320
3	0.184733	0.86832
4	0.180878	0.86496
5	0.180370	0.87200
6	0.180083	0.87112
7	0.179626	0.86972
8	0.184012	0.85964
9	0.182812	0.86092





تعداد پارامترها و دقت این مدل به شرح زیر است. نکته اینکه به دلیل افزودن ۱۰ prefix تعداد پارامتر ها برابر است با:

512 * 10 = 5120

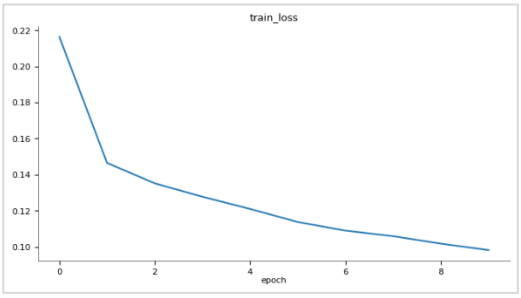
#Trainable parameters	accuracy	#parameters	method
5120	87.2	60511744	Soft prompt n_soft_prompt_token = 10

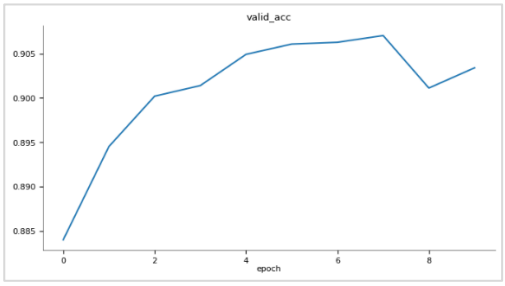
۳. نوت بوک سوم: Adapter

مدل adapter مدلی است که در آن از دو لایه Linear با یک لایه غیرخطی ReLU در آن استفاده شده است.

```python
class AdapterLayer(nn.Module):
    def __init__(
        self,
        emb_dim: int,
        bottleneck_size: int
    ):

        super().__init__()

        ######### Your code begins #########
        self.sharif_llm_adapter = nn.Sequential(
            nn.Linear(emb_dim, bottleneck_size),
            nn.ReLU(),
            nn.Linear(bottleneck_size, emb_dim)
        )
        ######### Your code ends ###########

    def forward(self, x: torch.Tensor):
        # do not remember to have residual connect
        ######### Your code begins #########
        output = self.sharif_llm_adapter(x) + x
        ######### Your code ends ###########

        return output
```

روند آموزش این مدل به شرح زیر است:



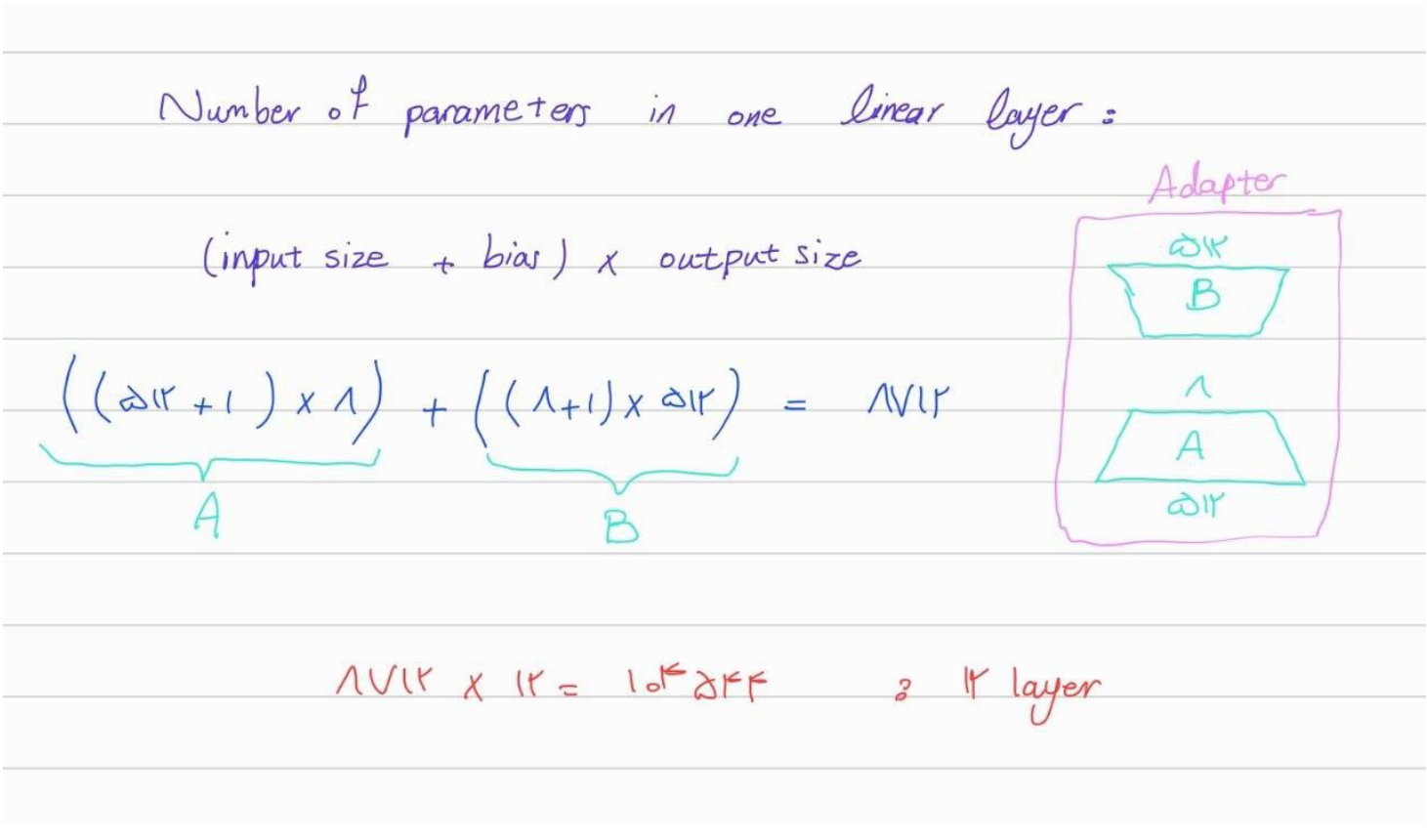


epoch	train_loss	valid_acc
0	0.216338	0.88400
1	0.146477	0.89452
2	0.135167	0.90020
3	0.127743	0.90140
4	0.120969	0.90492
5	0.113744	0.90608
6	0.108942	0.90628
7	0.105874	0.90704
8	0.101744	0.90112
9	0.098182	0.90340

تعداد پارامترها و دقت این مدل به شرح زیر است:

#Trainable parameters	accuracy	#parameters	method
104544	90.7	60611168	Adapter – bottleneck = 8

محاسبه تعداد پارامترهای مدل به صورتی دستی:



۴. نوت بوک چهارم: Adapter Hub

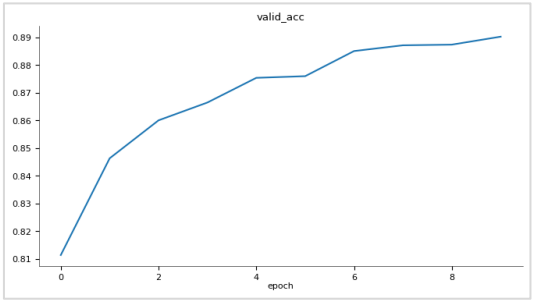
این نوت بوک در دو حالت زیر اجرا شده است:

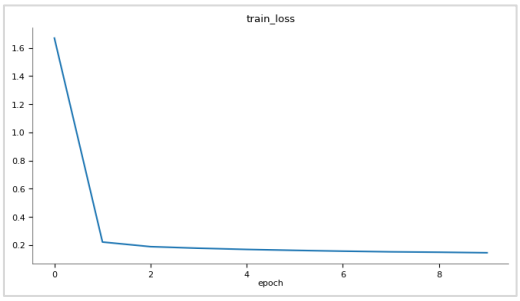
Bottleneck = 8 :

```python
from transformers import PfeifferConfig

BOTTLENECK_SIZE = 8

model = T5ForConditionalGeneration.from_pretrained(BASE_MODEL_NAME)

######### Your code begins #########
####################################
####################################
reduction_factor = model.config.d_model / BOTTLENECK_SIZE

adapter_config = PfeifferConfig(reduction_factor = reduction_factor)
model.add_adapter("text-classification", adapter_config)
model.train_adapter("text-classification", adapter_config)
```

نتایج اجرا به شرح زیر است:

epoch	train_loss	valid_acc
0	0.789543	0.86116
1	0.168655	0.88136
2	0.152748	0.88988
3	0.145260	0.89436
4	0.140173	0.89480
5	0.136079	0.89828
6	0.132952	0.89584
7	0.130564	0.90020
8	0.127528	0.89940
9	0.125409	0.90336





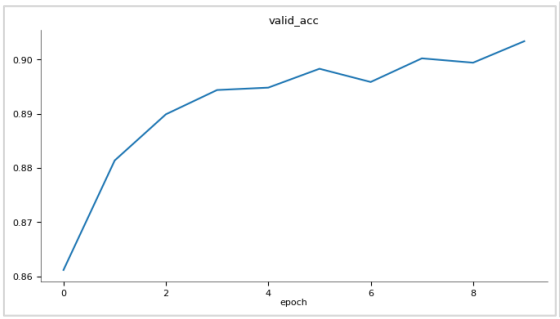
تعداد پارامترها و دقت مدل در جدول زیر آمده است.

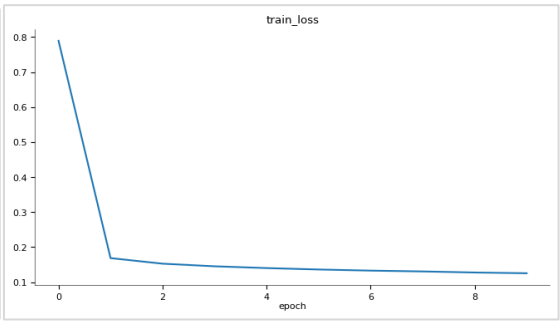
#Trainable parameters	accuracy	#parameters	method
104544	90.3	60611168	Adapter – bottleneck = 8

Bottleneck = 1:

```
from transformers import PfeifferConfig

BOTTLENECK_SIZE = 1

model = T5ForConditionalGeneration.from_pretrained(BASE_MODEL_NAME)

######### Your code begins #########
####################################
####################################
reduction_factor = model.config.d_model / BOTTLENECK_SIZE

adapter_config = PfeifferConfig(reduction_factor = reduction_factor)
model.add_adapter("text-classification", adapter_config)
model.train_adapter("text-classification", adapter_config)
```

نتایج اجرا به شرح زیر است:





epoch	train_loss	valid_acc
0	1.669941	0.81132
1	0.221337	0.84628
2	0.188494	0.86000
3	0.177335	0.86644
4	0.168908	0.87536
5	0.162210	0.87596
6	0.156521	0.88504
7	0.151749	0.88712
8	0.149025	0.88736
9	0.145179	0.89024

تعداد پارامترها و دقت مدل در جدول زیر آمده است. تعداد پارامترها مشابه حالت قبل داریم:

(512+1)*1 + (1+1)*512 = 1537
for 12 layers : 1537 * 12 = 18444

#Trainable parameters	accuracy	#parameters	method
18444	89.02	60525068	Adapter – bottleneck = 1

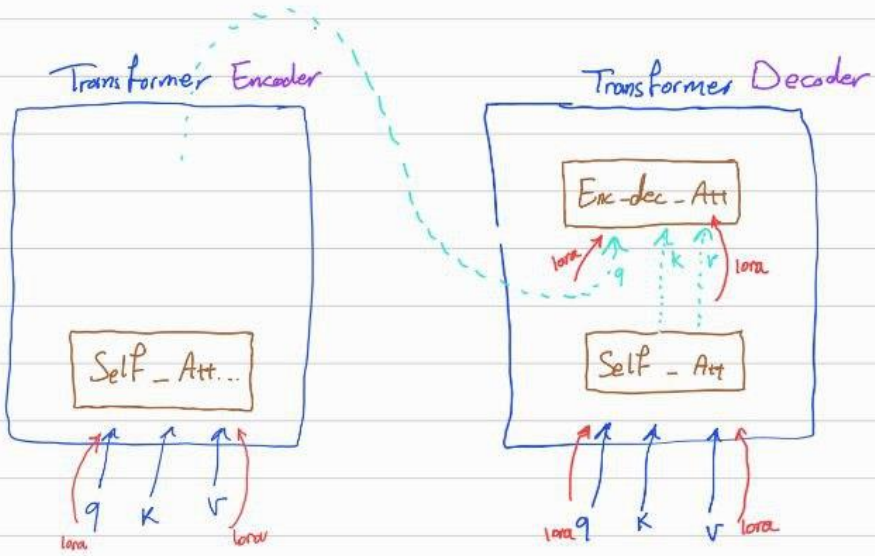
۵. نوت بوک پنجم: LoRA

در روش LoRA پارامتر ها به Q و V افزوده می‌شوند. البته در اسلاید ها گفته شده که به Q و K اضافه شده است. نکته قابل ذکر اینکه LoRA یک شبکه دو لایه با لایه‌های خطی است. که لایه اول با kaiming_uniform و لایه دوم صفر مقدار دهی شده است.

```
        alpha: float = ALPHA
        ):

        super().__init__()
        self.rank = rank
        self.alpha = alpha
        self.scaling = self.alpha / self.rank # scaling factor
        self.in_dim = module.in_features
        self.out_dim = module.out_features
        self.pretrained = module
        ######### Your code begins #########
        # create the A and initialize with Kaiming
        self.sharif_llm_A = nn.Linear(self.in_dim , self.rank , bias = False)
        # initilize sharif_llm_A
        nn.init.kaiming_uniform_(self.sharif_llm_A .weight)
        # create B and initialize with zeros
        self.sharif_llm_B = nn.Linear(self.rank , self.out_dim , bias = False)
        # initilize sharif_llm_B
        nn.init.zeros_(self.sharif_llm_B.weight)
        ######### Your code ends ###########


    def forward(self, x: torch.Tensor):

        pretrained_out = self.pretrained(x) # get the pretrained weights -> x.W

        # do the forward path for low rank matrices and scale the result
        ######### Your code begins #########
        lora_out = self.sharif_llm_A(x) # x@A
        lora_out = self.sharif_llm_B(lora_out) # x@A@B
        lora_out = self.scaling * lora_out # Scale by the scaling factor
        ######### Your code ends ###########

        return pretrained_out + lora_out # x@W + x@A@B*(scaling_factor)
```

تحلیلی که از تعداد پارامترهای این مدل داشتم در ادامه آورده شده است.

: LoRA rank = 8



هر lora : $\underbrace{(۵۱۲ \times ۸)}_{A} + \underbrace{(۸ \times ۵۱۲)}_{B} = ۸۱۹۲$ (بدون بایاس)

تعداد پارامترهای Encoder :

* هر Transformer_Encoder : روی q و روی v، lora دارد.
* از طرفی T5 ۴ لایه Encoder دارد.

بنابراین : تعداد پارامترهای Encoder = ۸۱۹۲ × ۲ × ۴ = ۹۸۳۰۴

تعداد پارامترهای Decoder :

* هر Transformer Decoder : روی q و روی v، اول sub layer ← lora دارد.
* روی q و v، دوم sub layer ← lora دارد.
* از طرفی ← مدل T5 ۴ لایه Decoder دارد بنابراین :

تعداد پارامترهای Decoder = ۸۱۹۲ × ۳ × ۴ = ۱۹۴۶۰۸

جمع : ۹۸۳۰۴ + ۱۹۴۶۰۸ = ۲۹۴۹۱۲

#Trainable parameters	accuracy	#parameters	method
294912	88.9	60801536	LoRA – rank = 8

LoRA rank = 1:

در این حالت هر lora ۱۰۲۴ پارامتر دارد. با تحلیل مشابه روند فوق، داریم:

برای encoder:

۱۰۲۴ × ۲ × ۶ = ۱۲۲۸۸

برای decoder:

۱۰۲۴× ۴ × ۶ = ۲۴۵۷۶

جمع کل : ۳۶۸۶۴

#Trainable parameters	accuracy	#parameters	method
36864	88.8	60543488	LoRA – rank = 1

نکته با توجه به دقت ها می توان نتیجه گیری کرد که در LoRA رنک ۱ هم می‌توان جواب خوبی دهد. دلیل آن انتخاب u و v های متناظر با بزرگترین singular value در تجزیه svd این ماتریس است. چون این بردارها بیشترین اطلاعات ماتریس را در خود نگه داری می کنند. تحلیل زیر در اسلایدهای درس وجود دارد:

Why r=1 works well in practice? (cont.)

$$A = V\Sigma U$$

$$\Rightarrow A = \sum_{i=1}^{r} \sigma_i v_i u_i^T$$

$$\Rightarrow Ax = \sum_{i=1}^{r} \sigma_i v_i u_i^T x$$

$$\Rightarrow Ax = \sum_{i=1}^{r} \sigma_i \langle u_i, x \rangle v_i$$

Pick highest σ_i , compare the corresponding u_i's in two A's

25

Why r=1 works well in practice? (cont.)

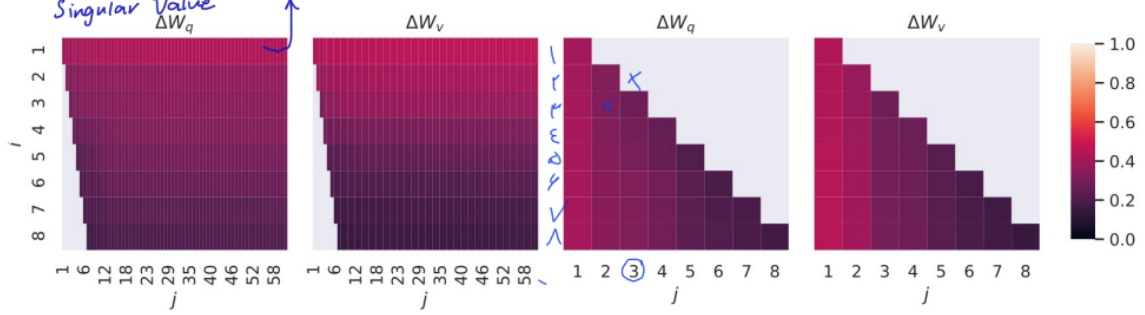


Figure 3: Subspace similarity between column vectors of $A_{r=8}$ and $A_{r=64}$ for both ΔW_q and ΔW_v. The third and the fourth figures zoom in on the lower-left triangle in the first two figures. The top directions in $r = 8$ are included in $r = 64$, and vice versa.

در ادامه مقایسه همه مدل‌ها آورده شده است:

Notebook 5			Notebook 4		Notebook 3	Notebook 2			Notebook 1	
LoRA rank = 8	LoRA rank = 1	Lora code rank = 8	Adapter bottleneck = 1	Adapter Bottleneck = 8	Adapter	My soft prompt 10	My soft prompt 1	Main Soft Prompt	Full Fine Tuning	
88.7	88.8	88.9	89.02	90.3	90.7	87.2	84.9	87.0	90.1	accuracy
60801536	60543488	60801536	60525068	60611168	60611168	60511744	60507136	60511744	60506624	#parameters from notebook
294912	36864	294912	18444	104544	104544	5120	512	5120	all	#trainable parameters (notebook – full finetune)
294912	36864	294912	18444	104544	104544	5120	512	5120	all	#trainable parameters (my calculation)

منابع :

https://learnprompting.org/docs/trainable/soft_prompting

https://www.youtube.com/watch?v=Us5ZFp16PaU