نام و نام خانوادگی	**ادیب رضائی – امیرمحمد خسروی**
شماره دانشجویی	**810198386 - 810198401**
تاریخ ارسال گزارش	**۱۴۰۱.۰۳.۱۴**

به نام خدا
دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق
تمرین پنجم

فهرست

شکل‌ها

جدول‌ها

پاسخ 1. سامانه ی پرسش-پاسخ

۱-۱ مدلسازی مسئله

شرح اجزای کلی مدل BERT و معماری ترنسفورمر، نمایش ورودی و اهداف pretraining:

1. معماری ترنسفورمری:

مدل BERT از معماری ترنسفورمر استفاده می‌کند. ترنسفورمر یک معماری شبکه عصبی با توجه attention-based است که برای پردازش داده‌های توالی مانند جملات و متون استفاده می‌شود. این معماری شامل لایه‌های multi-head attention و لایه‌های feed-forward است.

معماری مدل BERT بر پایه پیاده‌سازی اصلی توضیح داده شده در مقاله Vaswani و همکارانش (2017) و منتشر شده در کتابخانه tensor2tensor است.

اگر تعداد لایه‌ها (بلاک‌های ترنسفورمر) را با L، اندازه hidden size را با H و تعداد self-attention heads را با A نشان می‌دهیم. در این مقاله، به طور کلی نتایج برای دو اندازه مدل گزارش میشود:

BERT (BASE) = (L=12, H=768, A=12, Total Parameters=110M)

BERT(LARGE) = (L=24, H=1024, A=16, Total Parameters=340M)

انتخاب BERTBASE با همان اندازه مدل OpenAI GPT به منظور مقایسه صورت گرفته است.

2. نمایش ورودی:

برای اینکه BERT بتواند با تنوعی از وظایف پس‌انداز مختلف برخورد کند، نمایش ورودی قادر است یک جمله تکی و یا یک جفت جمله (مانند سوال و پاسخ) را به صورت یک توالی توکن نمایش دهد. در طول این کار، "جمله" می‌تواند یک بخش خاص از متن پیوسته باشد، نه لزوماً جمله زبانی واقعی. "توالی" به توالی توکن ورودی به BERT اشاره دارد که می‌تواند یک جمله تکی یا دو جمله کنار هم باشد.

توکن اول هر توالی همیشه یک توکن خاص طبقه‌بندی ([CLS]) است. حالت پنهان نهایی متناظر با این توکن به عنوان نماینده توالی تجمعی برای وظایف طبقه‌بندی استفاده می‌شود. جفت

جملات در یک توالی تکی قرار داده می‌شوند. جملات را به دو روش از یکدیگر متمایز میشوند. ابتدا با یک توکن خاص ([SEP]) جدا میشوند. ثانیاً، به هر توکن یک جاسازی یادگیری‌شده اضافه میشود که نشان می‌دهد آیا آن توکن متعلق به جمله A است یا جمله B.

به طور خلاصه ورودی مدل BERT شامل توالی‌های کلمات است که با استفاده از توکن‌ها نمایش داده می‌شوند. برای نمونه، جمله "من یک انسان هستم" به توالی توکن‌های "[CLS] من یک انسان هستم [SEP]" تبدیل می‌شود. توکن [CLS] برای نشان دادن توالی ورودی و توکن [SEP] برای جداکردن جملات در مواردی که چند جمله وجود دارد، استفاده می‌شوند.

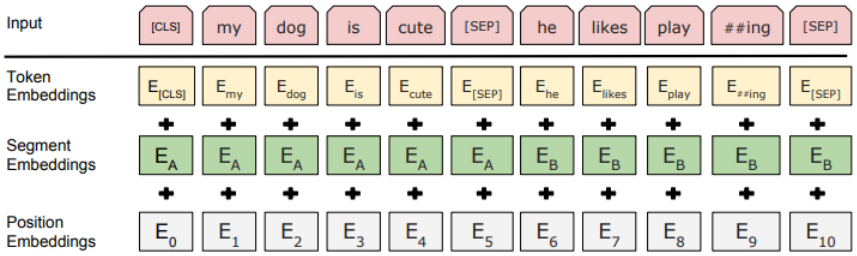


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

3. هدف pretraining:

مدل BERT با استفاده از pretraining بر روی مجموعه‌های بزرگی از متون بی‌برچسب آموزش می‌بیند. هدف pretraining، پیش‌بینی کلمات از متن حذف شده است. در این روش، برخی از کلمات متن اصلی حذف شده و مدل باید سعی کند کلمه‌های حذف شده را با توجه به باقی متن پیش‌بینی کند. این کار باعث یادگیری نمایش معنایی کلمات و ارتباطات بین کلمات مختلف در متون می‌شود.

ساختار کلی مدل پیاده سازی شده:

ورودی مدل شامل دو قسمت است: متن و سوال مربوط به آن.

متن (Text): متنی است که در آن می‌خواهیم پاسخ به سوال را استخراج کنیم. این متن می‌تواند شامل یک جمله تکی یا چند جمله پشت سر هم باشد. برای مثال، می‌تواند یک پاراگراف یا یک مقاله کامل باشد.

سوال (Question): سوالی است که مرتبط با متن است و می‌خواهیم به آن پاسخ را استخراج کنیم. سوال می‌تواند مربوط به محتوای متن باشد و بر اساس آن، مدل باید بتواند بهترین پاسخ ممکن را از متن استخراج کند.

با ترکیب متن و سوال در یک توالی توکن، ورودی مدل تشکیل می‌شود که می‌تواند شامل توکن‌های مربوط به متن، توکن‌های جداکننده (SEP) بین متن و سوال، و توکن‌های نشانگر جمله A و جمله B باشد. این توالی توکن‌ها به عنوان ورودی به مدل داده می‌شود تا اطلاعات مورد نیاز برای استخراج پاسخ را دریافت کند.

بردار [CLS] نهایی، که نشان دهنده وضعیت تمام توکن‌های ورودی است، به عنوان نماینده کلیه توکن‌ها و نماینده تمام اطلاعات موجود در ورودی استفاده می‌شود. این بردار [CLS] به عنوان ورودی به تمام وظایف پردازش پس از آموزش (مانند دسته‌بندی یا دنباله‌بندی) ارسال می‌شود.

خروجی مدل در مسئله پرسش و پاسخ شامل بردار [CLS] نهایی باشد که نماینده اطلاعات ورودی است و از آن برای استخراج پاسخ استفاده می‌شود.

در نهایت نیز بهترین پاسخی که مدل از متن خارج میکند را خواهیم داشت.

ساختار مدل، توابع خطای مورد استفاده و مدل قرار است چه چیزی را آموزش ببیند:

1. ساختار مدل:

- کلاسی به صورت یک زیرکلاس از nn.Module است که مدل BERT را شامل می‌شود.
- مدل از یک لایه Transformer بر پایه BERT استفاده می‌کند.
- بعد از لایه Transformer، یک لایه Dropout با نرخ Dropout برابر با 0.2 اعمال می‌شود.
- در نهایت، یک لایه Fully Connected با ورودی‌های به اندازه hidden_size مدل Transformer و خروجی با اندازه 2 (برای استخراج شروع و پایان پاسخ) قرار دارد.

2. توابع خطا مورد استفاده:

- تابع خطای cross_entropy

3. آموزش مدل:

- مدل با استفاده از تابع AdamW به عنوان بهینه‌ساز و StepLR به عنوان برنامه‌ریزی نرخ یادگیری، آموزش داده می‌شود.

- تابع خطا باید تعریف شده باشد و در criterion قرار گیرد.

- برای آموزش مدل، پارامترها را به صورت زیر تعیین میکنیم:

epocks = 1

weight_decay=0.1

learning_rate = 5e-5

gamma = 0.1

train_batch_size = 16

test_val_batch_size = 4

max_length = 512

step_size = 1

4. ورودی و خروجی مدل:

- ورودی مدل شامل شناسه‌های توکن‌ها، token_type_ids شناسه‌های نوع جمله و attention_mask است که به وسیله توکنایزر BERT توسط tokenizer تولید می‌شود.

- خروجی مدل شامل دو بردار است که شامل استخراج شروع و پایان پاسخ از متن استفاده می‌شوند.

۱-۲ پیش پردازش داده ها

پیش پردازش‌های مورد نیاز:

1. تبدیل به دیتافریم: ابتدا داده ها را از فایل های json میخوانیم و آنها را در dataframe ذخیره میکنیم. سپس dataframe های به دست آمده را با استفاده از توابع کتابخانه datasets به آبجکت های Dataset تبدیل میکنیم و در یک DatasetDict ذخیره میکنیم.

2. توکن‌بندی (Tokenization): سپس داده‌های متنی را به توکن‌های جداگانه تقسیم میکنیم. این عملیات توسط توکنایزر BERT انجام می‌شود. هر جمله را به توکن‌های جداگانه تقسیم کرده و به صورت مجموعه‌ای از اعداد (شاخص توکن) نمایش می‌دهد.

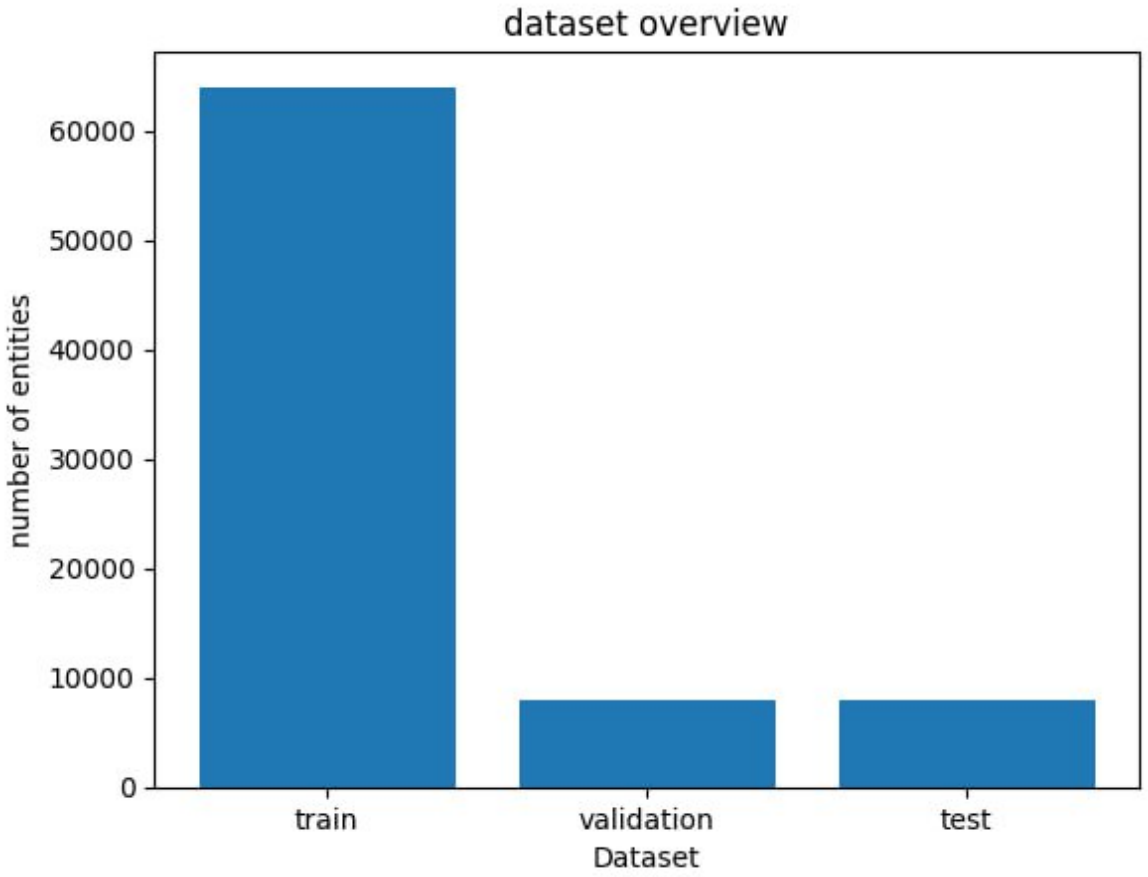
3. افزودن توکن‌های ویژه: برای جداسازی جملات مختلف در مدل BERT، باید توکن‌های ویژه مانند [CLS] (نشان دهنده شروع جمله) و [SEP] (نشان دهنده اتمام جمله) به توکن‌ها اضافه میشوند. همچنین، برای تمایز بین دو جمله در صورت وجود، توکن‌های اختصاص داده شده به هر جمله نیز به توکن‌ها اضافه می‌شوند.

4. پدینگ (Padding): برای تطابق طول تمام دنباله‌ها، بخشی از داده‌ها نیاز به پدینگ دارند. در این مرحله، توکن‌های پد به توکن‌هایی که طول کمتری دارند، اضافه می‌شوند تا طول آن‌ها به طول بیشترین دنباله برسد. طول همه توکن ها به یک مقدار که تعیین شده است میرسد.

4. ایجاد ماسک توجه (Attention Mask): در BERT، برای محاسبه توجه، یک ماسک توجه استفاده می‌شود که نشان می‌دهد کدام توکن‌ها باید در محاسبات توجه در نظر گرفته شوند و کدام‌ها باید نادیده گرفته شوند. در این مرحله، یک ماسک توجه ایجاد می‌شود که نشان می‌دهد کدام توکن‌ها معتبر هستند.

اطلاعات آماری دیتاست و پلات های مربوط به Exploration داده ها:

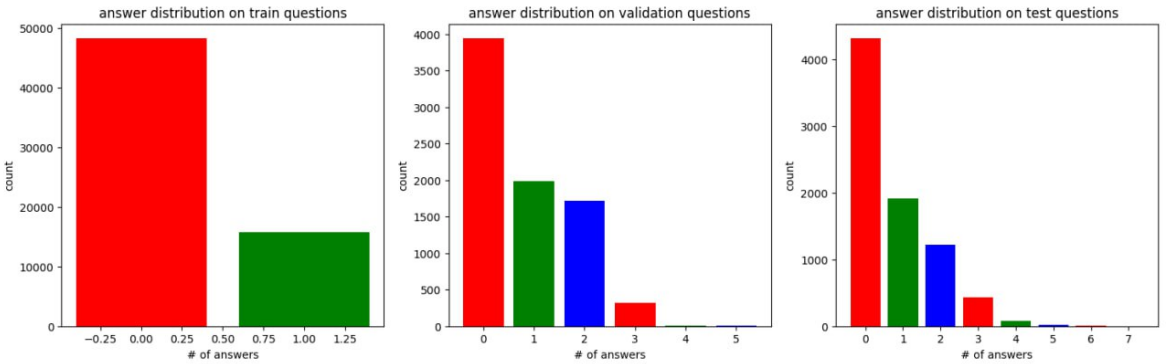
نحوه تقسیم داده ها در دیتاست:



شکل ۱. Dataset Overview

```
[17] ds['train']

Dataset({
    features: ['id', 'title', 'context', 'question', 'answers'],
    num_rows: 63994
})

[36] ds['validation']

Dataset({
    features: ['id', 'title', 'context', 'question', 'answers'],
    num_rows: 7976
})

[37] ds['test']

Dataset({
    features: ['id', 'title', 'context', 'question', 'answers'],
    num_rows: 8002
})
```

شکل 2. فیچر های موجود در داده ها و تعداد آن ها



شکل 3. نمودار تعداد داده ها بر حسب تعداد جواب ها

۳-۱ پیاده سازی مدل

مدل را با دوبار با استفاده از parsBERT و ALBERT و با شبکه طراحی شده در مراحل قبل پیاده سازی میکنیم و هر دو را ترین کرده و نتایج را گزارش میکنیم.

۴-۱ . ارزیابی و پس پردازش (Postprocessing)

در طول اجرای تسک توسط مدل، استثناهایی به دلیل بارگیری داده‌ها، پیش‌پردازش و پس‌پردازش طولانی متون زمینه و محدودیت ورودی مدلهای ترنسفورمری به وجود می‌آیند. نیاز است که این استثناها را مدیریت شوند تا مدل بتواند با آنها سازگاری یابد و نتایج مناسبی را تولید کند.

نتایج:

مدل parsBERT:

برای مدل parsBERT نتایج بعد از آموزش مدل و evaluate کردن آن با داده های تست به صورت زیر خواهد بود:

```
Epoch [1] Batch [3700] - Loss = 0.9790
Epoch [1] Batch [3800] - Loss = 0.9744
Epoch [1] Batch [3900] - Loss = 0.9724
Downloading builder script: 6.46k/? [00:00<00:00, 336kB/s]
Downloading extra modules: 11.3k/? [00:00<00:00, 551kB/s]
Train Loss: 0.9689
Validation Loss: 0.8452 - Exact Match: 69.709 - F1_score: 82.936
```

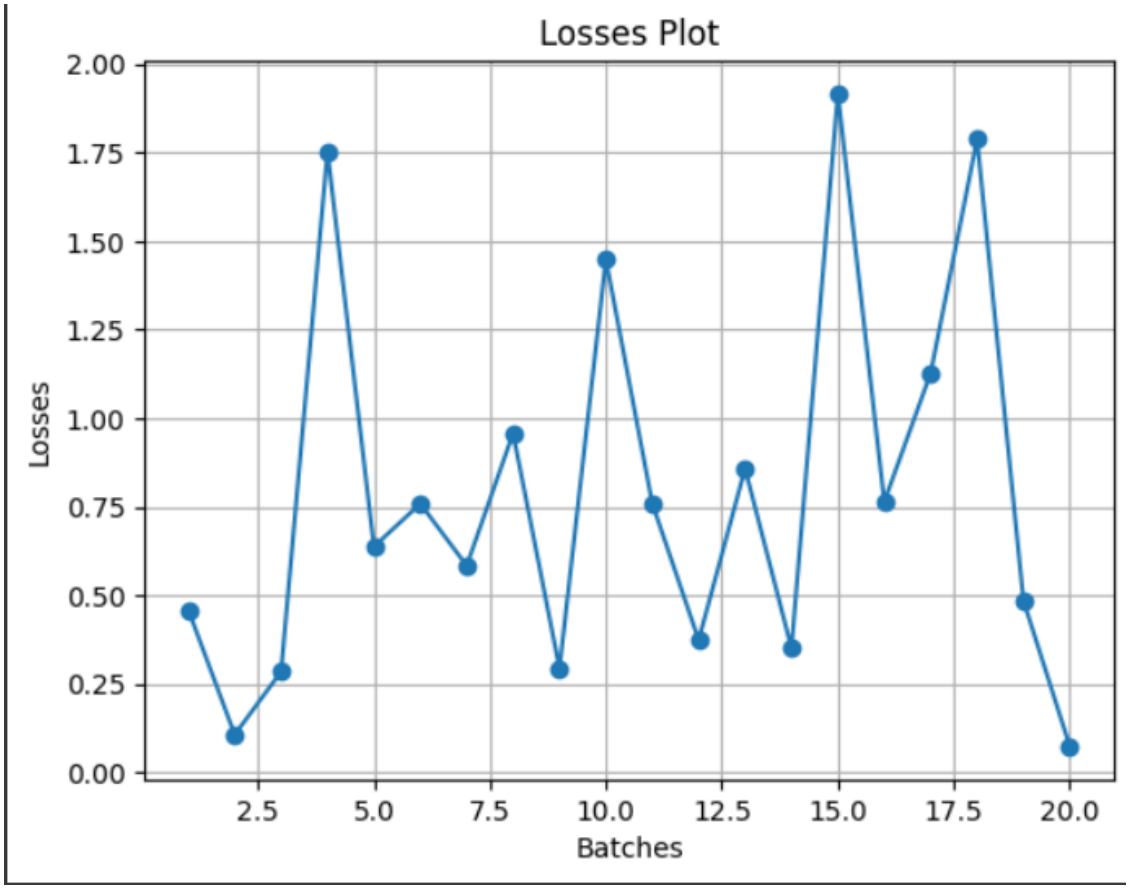
شکل 4. نتایج بعد از train و evaluate

در شکل زیر نتایج شامل Exact Match ،validation loss ،train loss و F1 score بهتر نمایش داده شده اند:

```
Train loss =  0.9689259925708175
Validation loss =  0.8452220881738912
Exact Match score =  69.70912738214643
F1 score =  82.93638169533077
```

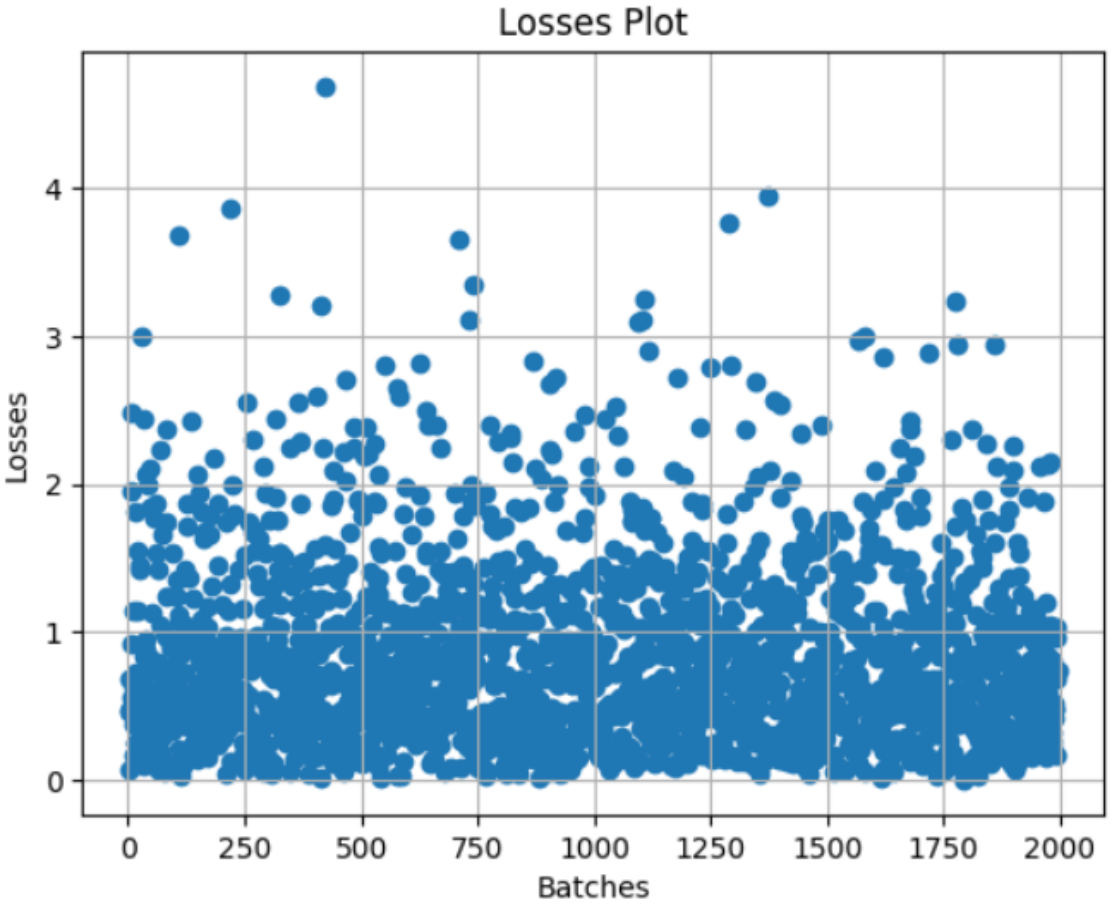
شکل 5. نتایج parsBERT

نمودار لاس مربوط به batchهای مختلف. چون یک epoch داشتیم به طور کل نمودار لاس واقعی را نمیتوانیم رسم کنیم و از این نمودار رسم شده نتیجه خاصی نمیتوان گرفت:



شکل 6. نمودار خطای هر بچ

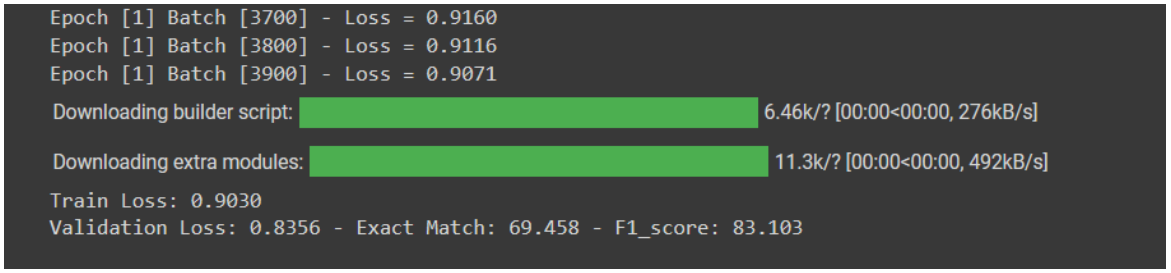
اگر بخواهیم برای همه loss ها scatter plot رسم کنیم به صورت زیر خواهد شد. برای شکل بالا از هر 100 لاس یک لاس را در نظر گرفتیم و رسم کردیم:



شکل 7. نمودار همه lossها

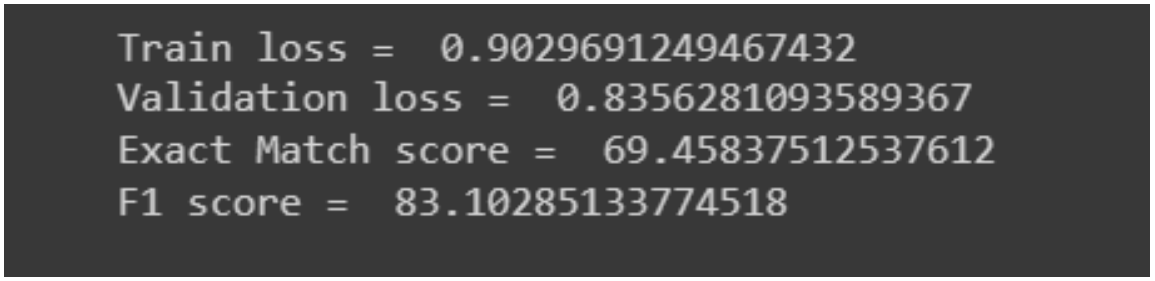
مدل ALBERT:

برای مدل ALBERT نتایج بعد از آموزش مدل و evaluate کردن آن با داده های تست به صورت زیر خواهد بود:



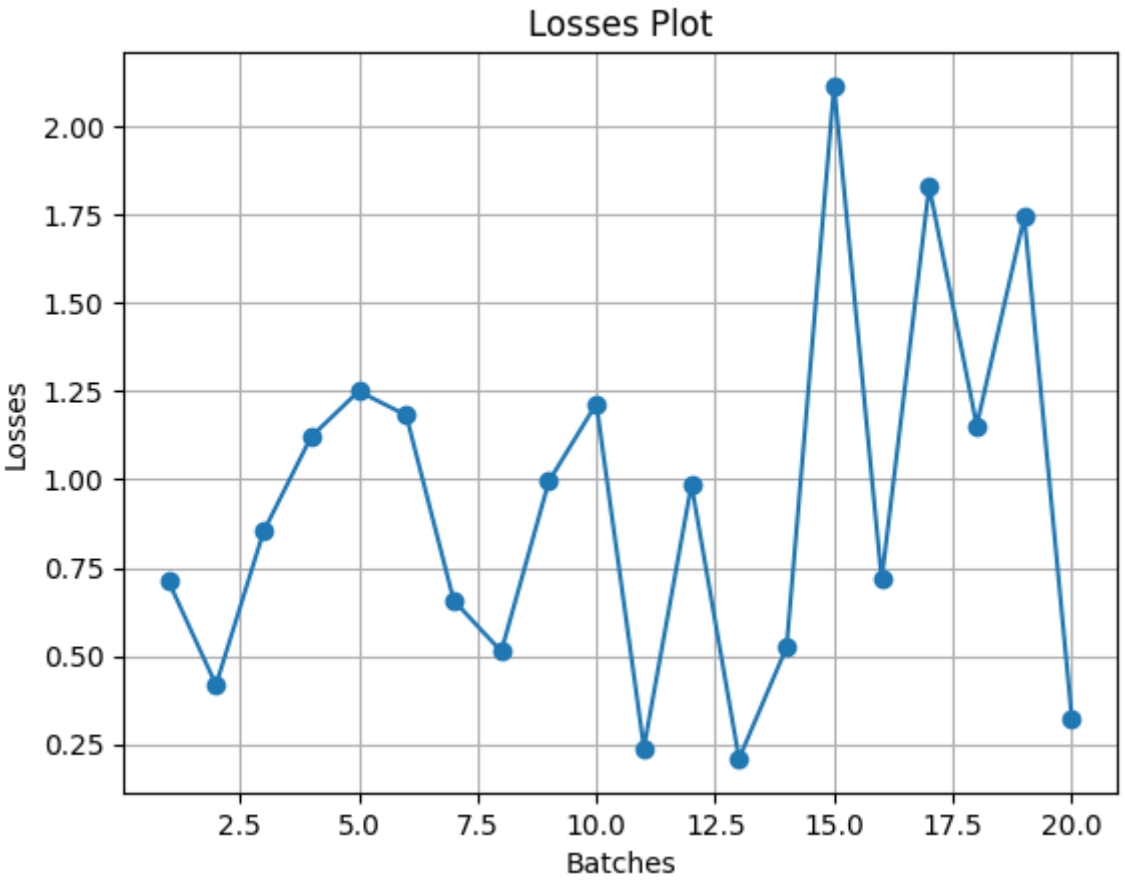
شکل 8. نتایج بعد از train و evaluate

در شکل زیر نتایج شامل Exact Match ،validation loss ،train loss و F1 score بهتر نمایش داده شده اند:



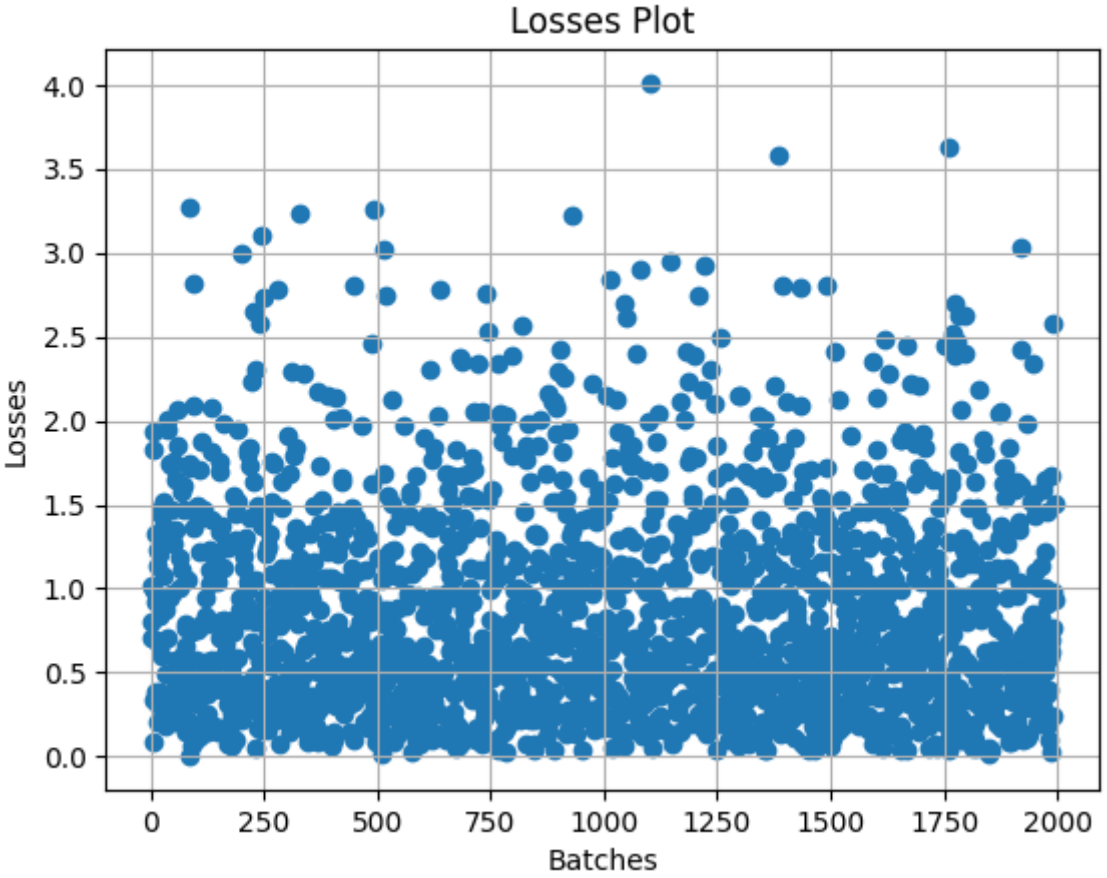
شکل 9. نتایج ALBERT

نمودار لاس مربوط به batchهای مختلف. چون یک epoch داشتیم به طور کل نمودار لاس واقعی را نمیتوانیم رسم کنیم و از این نمودار رسم شده نتیجه خاصی نمیتوان گرفت:



شکل 10. نمودار خطای هر بچ

اگر بخواهیم برای همه loss ها scatter plot رسم کنیم به صورت زیر خواهد شد. برای شکل بالا از هر 100 لاس یک لاس را در نظر گرفتیم و رسم کردیم:

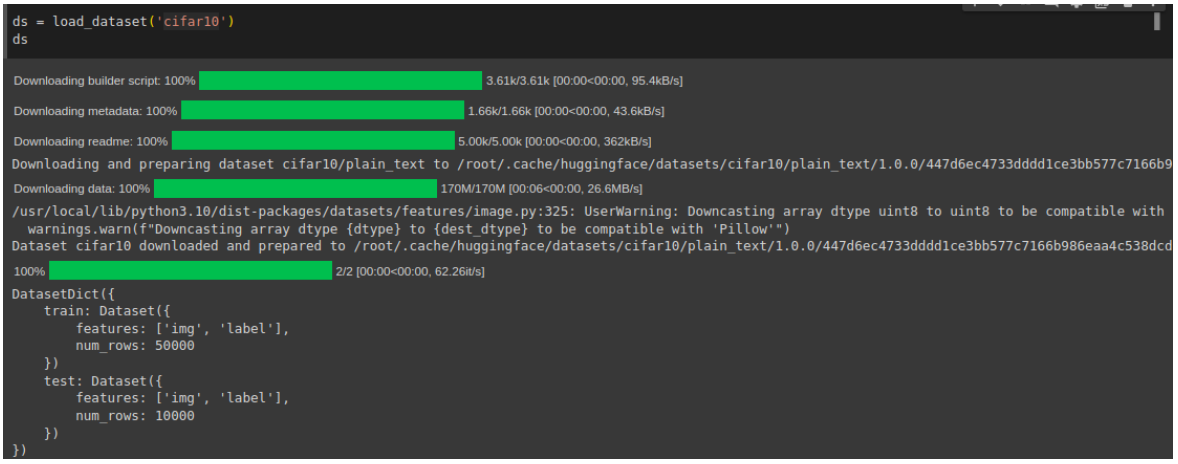


شکل 11. نمودار همه lossها

همانطور که نتایج نشان میدهد، مدل ها تقریبا مشابه یکدیگر عمل میکنند و دقت و خطای مشابهی نیز دارند (اختلاف کمتر از یک درصد). زمان آموزش ALBERT مقداری بیشتر از مدل دیگر بود.

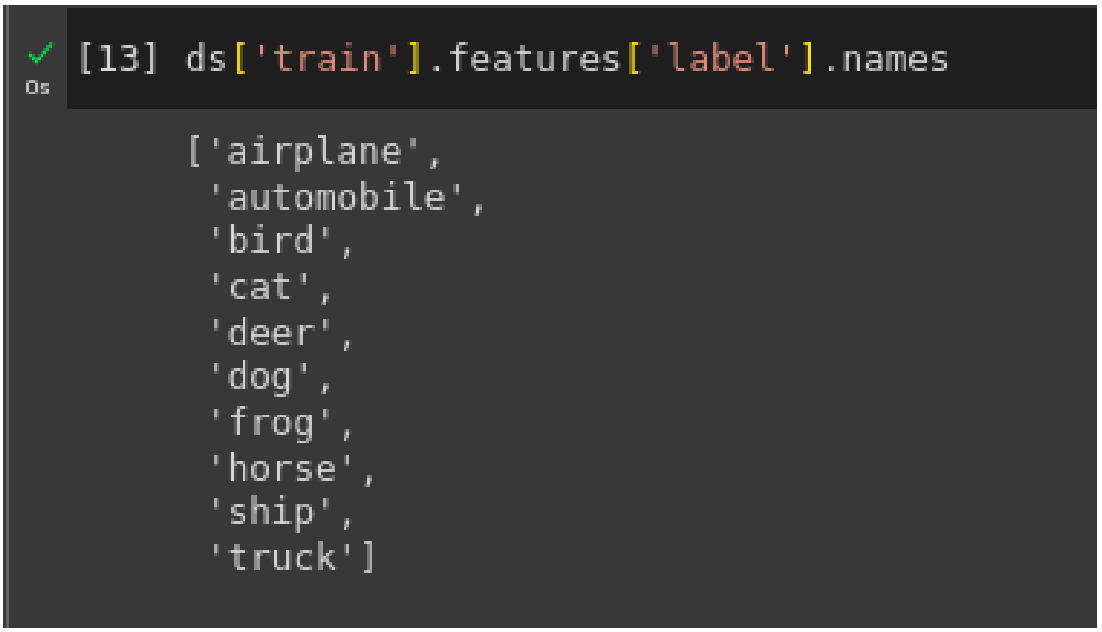
پاسخ ۲ - استفاده از vision transformer برای طبقه بندی تصاویر

در بخش اول ابتدا دیتاست را توسط قطعه کد زیر لود میکنیم.



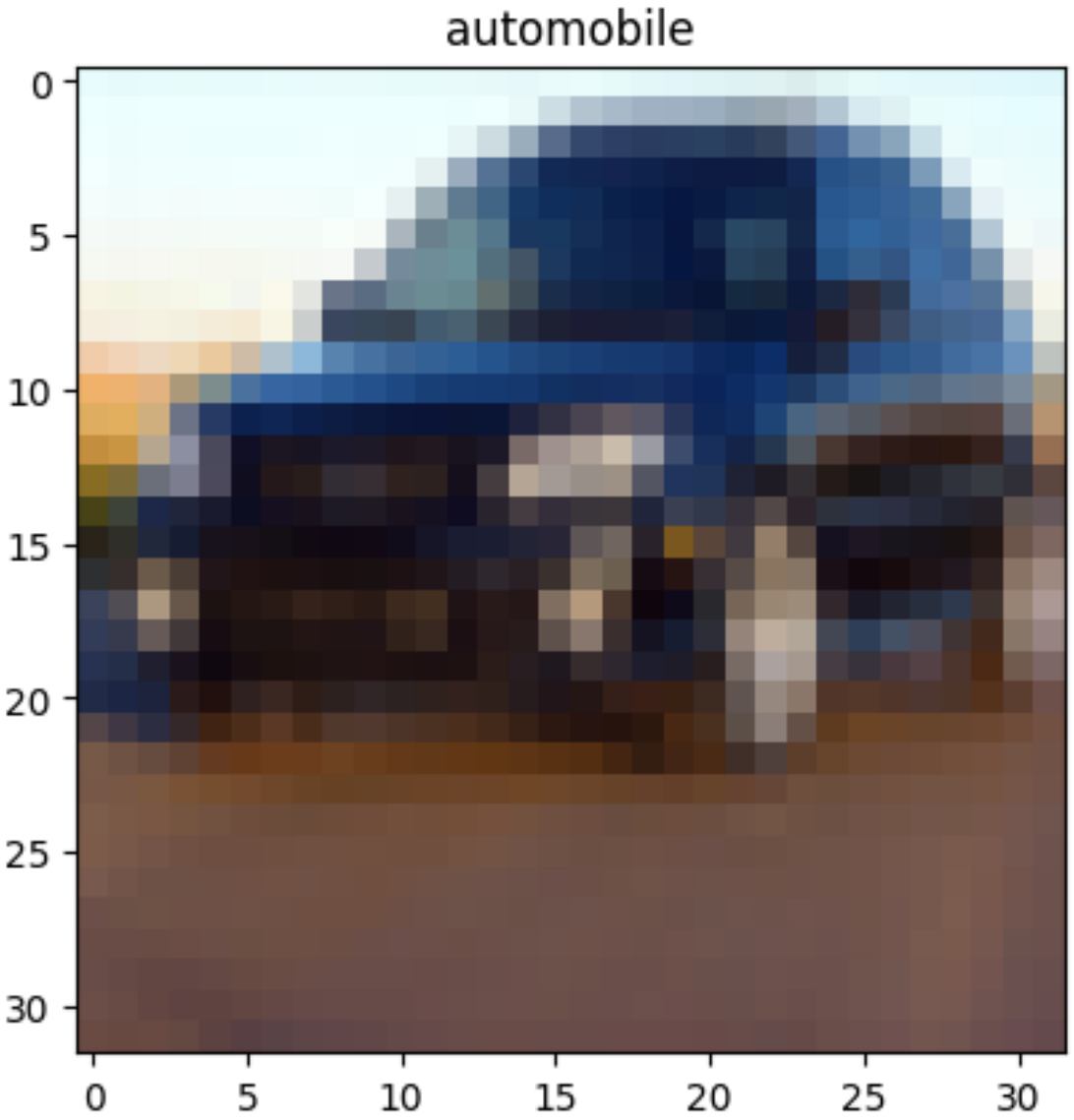
شکل ۱. لود کردن دیتاست توسط load_dataset

کلاس های دیتاست را چاپ میکنیم تا دید کلی نسبت به دیتاست داشته باشیم:



شکل ۲. کلاس های دیتاست CIFAR-10

یک عکس رندوم از دیتاست به همراه لیبل مرتبط آن نشان میدهیم



شکل ۳. تصویر یک اتوموبیل

قطعه کد زیر تابعی به نام `transform` را تعریف می‌کند که یک دسته نمونه را به عنوان ورودی می‌گیرد و تغییراتی روی آن اعمال می‌کند.

اولین کاری که انجام می‌دهد، تبدیل تصاویر PIL (Pillow) در دسته نمونه به مقادیر پیکسل است. این کار با استفاده از تابع `feature_extractor` انجام می‌شود. `feature_extractor` احتمالاً یک مدل است که بر روی تصاویر عملیاتی مانند استخراج ویژگی‌ها را انجام می‌دهد و

مقادیر پیکسل تصاویر را برمی‌گرداند. ورودی `x` ها به صورت لیستی از تصاویر است که از دسته نمونه گرفته شده است.

در خط بعد، برچسب‌ها (labels) نمونه‌ها را به ورودی اضافه می‌کند. فرضاً `example_batch['label']` لیستی از برچسب‌ها است که با تصاویر مربوطه در دسته نمونه متناظر است. ورودی‌ها به صورت یک دیکشنری تنسورها (PyTorch tensors) برگردانده می‌شود که شامل مقادیر پیکسل تصاویر و برچسب‌ها می‌شود.

```
def transform(example_batch):
    inputs = feature_extractor([x for x in example_batch['img']], return_tensors='pt')
    inputs['labels'] = example_batch['label']
    return inputs

[24] prepared_ds = ds.with_transform(transform)
```

شکل ۴. تابع transorm توضیح داده شده

در مرحله بعدی از مدل تماما ترنسفورمر deiT استفاده میکنیم.

```
[28] labels = ds['train'].features['label'].names

    model = DeiTForImageClassification.from_pretrained(
        'facebook/deit-small-distilled-patch16-224',
        num_labels=len(labels),
        id2label={str(i): c for i, c in enumerate(labels)},
        label2id={c: str(i) for i, c in enumerate(labels)},
        ignore_mismatched_sizes=True
    )
```

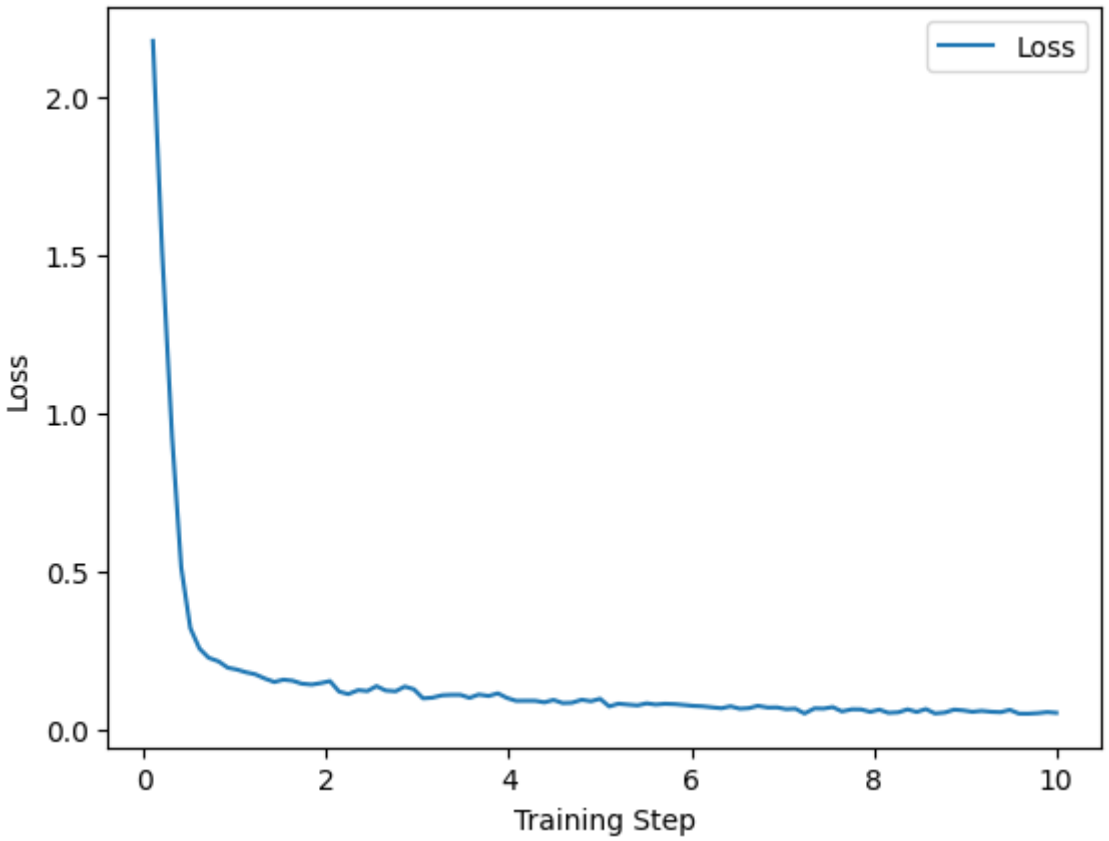
شکل ۵. استفاده از مدل از پیش آماده DeiT برای fine tune کردن

همچنین نیاز داریم تا لایه های مدل را به جز لایه آخر جهت fine-tune کردن مدل freeze کنیم. اینکار را مطابق شکل ۶ انجام میدهیم:

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

for param in model.deit.encoder.layer[11].parameters():
    param.requires_grad = True

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

شکل ۶. فریز کردن همه لایه ها به جز لایه classifier

در آخر مدل را با هایپر پارامتر learning_rate = 0.0001 و batch_size = 512 و epoch = 10 آموزش میدهیم. نتیجه نمودار loss آن مطابق شکل زیر خواهد شد:



شکل ۷. نمودار loss بر حسب training step

مقدار نهایی loss عدد 0.0531 میشود.