پروژه سوم

Deep Learning

محمدمحسن همایونی

در ابتدا مثل باقی پروژه ها کتاب خانه های مورد نظر را دانلود یا import میکنیم. پس از وصل کردن گوگل کولب، ابتدا فایل GloVe را دانلود میکنیم که به کمک آن بتوانیم word embedding را انجام دهیمو بعد از آن دیتاست را دانلود میکنیم.

"**GloVe (Global Vectors for Word Representation)** یک مدل word embedding است که توسط محققان دانشگاه استنفورد توسعه داده شده است. این مدل به‌طور خاص برای نمایش کلمات در فضای برداری با استفاده از اطلاعات آماری از کل مجموعه داده‌های متنی طراحی شده است."

سپس برای درک شکل کلی داده ها از آنها یک خروجی میگیریم.

Pre Processing

در این مرحله ابتدا مسیر ها را تعریف کرده و طول وکتور و طول دنباله را تعیف میکنیم. همچنین تعداد ایپاک ها و لرنینگ ریت هم در این مرحله تعیف میشود.

سپس با استفاده از GloVe کلمات را embed کرده و ذخیره میکنیم. سپس برای راحت تر شدن توکنایز کردن یکسری کار ها از قبیل حذف سطر و ستون های خالی و... انجام میدهیم.

در مرحله بعدی با استفاده از nltk کلمات را به توکن تبدیل میکنیم تا در ترین کردن مدل استفاده شود.

سپس از هر دیتا ۲۰ توکن اول را برداشته و از آنها استفاده میکنیم. بعد از آن توکن ها را با اعدادی که نمایانگر کلمه در وکب هستند تبدیل میکنیم.

چالش ها:

- در این مرحله پیش پردازش متاسفانه کتاب خانه لازم برای توکن شدن در منبع ذکر نشده بود و بعد از اضافه کردن آن ارورهایی پیش می آمد که قادر به حل آنها نبودم برای همین از کتاب خانه spacy

برای توکنایز کردن استفاده کردم ولی به خاطر سرعت کند آن تایم کولب تمام شد و دوباره به nltk سوییچ کردم و اینبار کار کرد.

Create attention model

در این مرحله طبق منبع داده شده مدل خود را تعریف میکنیم:

```
class AttentionModel(nn.Module):

    def __init__(self, vec_len, seq_len, n_classes):
        super(AttentionModel, self).__init__()
        self.vec_len = vec_len
        self.seq_len = seq_len
        self.attn_weights = torch.cat([torch.tensor([[0.]]),
                                       torch.randn(vec_len, 1) /
                                       torch.sqrt(torch.tensor(vec_len))])
        self.attn_weights.requires_grad = True
        self.attn_weights = nn.Parameter(self.attn_weights)
        self.activation = nn.Tanh()
        self.softmax = nn.Softmax(dim=1)
        self.linear = nn.Linear(vec_len + 1, n_classes)

    def forward(self, input_data):
        hidden = torch.matmul(input_data, self.attn_weights)
        hidden = self.activation(hidden)
        attn = self.softmax(hidden)
        attn = attn.repeat(1, 1, self.vec_len + 1).reshape(attn.shape[0],
                                                           self.seq_len,
                                                           self.vec_len + 1)
        attn_output = input_data * attn
        attn_output = torch.sum(attn_output, axis=1)
        output = self.linear(attn_output)
```

سپس توابعی را برای آزمایش مدل میسازیم و میگذاریم که Train شود.

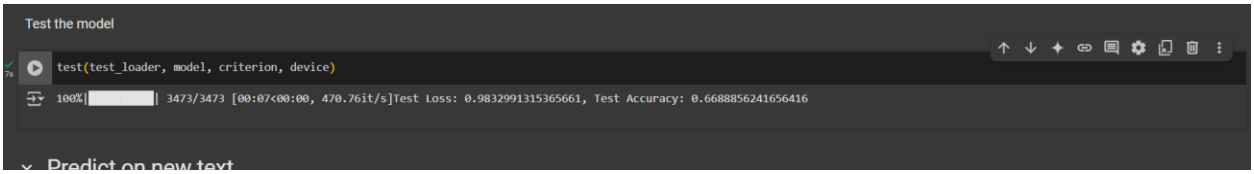
برای سری اول از لرنینگ ریت "۰,۰۰۰۵" و تعداد ایپاک را ۵۰ میگذاریم که نتیجه زیر را میدهد.

Test the model

```
test(test_loader, model, criterion, device)
```

```
100%|██████████| 3473/3473 [00:07<00:00, 495.37it/s]Test Loss: 0.9828092007928096, Test Accuracy: 0.6694434992016335
```

نتیجه آنچنان دلچسب نیست برای همین ابتدا لرنینگ ریت را به "۰/۰۰۰۰۵" تغییر میدهم و تعداد ایپاک ها را به ۱۰۰ میرسانم.



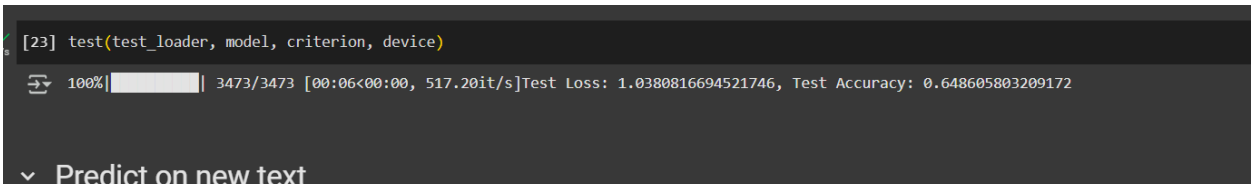
نتیجه بد تر شد.

سپس برای بهتر کردن نتیجه بجای single-head attention model یک مدل multi head به صورت زیر ساختم ولی نتوانستم اندازه ورودی آن را برای درست کار کردنش تنظیم کنم.

```python
def train_multihead(train_loader, valid_loader, model, criterion, optimizer,
                    device, num_epochs, model_path):
    """
    Function to train the Multi-Head Attention model.

    Args:
        train_loader: Data loader for train dataset.
        valid_loader: Data loader for validation dataset.
        model: Multi-Head Attention model object.
        criterion: Loss function.
        optimizer: Optimizer.
        device: CUDA or CPU.
        num_epochs: Number of epochs.
        model_path: Path to save the model.
    """
    best_loss = 1e8
    for i in range(num_epochs):
        print(f"Epoch {i+1} of {num_epochs}")
        valid_loss, train_loss = [], []
        model.train()
        # Train loop
        for batch_labels, batch_data in tqdm(train_loader):
            # Move data to GPU if available
            batch_labels = batch_labels.to(device)
            batch_data = batch_data.to(device)
            # Forward pass
            batch_output = model(batch_data)
            batch_output = torch.squeeze(batch_output)
            # Calculate loss
            loss = criterion(batch_output, batch_labels)
            train_loss.append(loss.item())
            optimizer.zero_grad()
            # Backward pass
            loss.backward()
            # Gradient update step
            optimizer.step()
        model.eval()
        # Validation loop
        for batch_labels, batch_data in tqdm(valid_loader):
            # Move data to GPU if available
            batch_labels = batch_labels.to(device)
            batch_data = batch_data.to(device)
            # Forward pass
            batch_output = model(batch_data)
            batch_output = torch.squeeze(batch_output)
            # Calculate loss
            loss = criterion(batch_output, batch_labels)
```

در آخر گفتم شاید مشکل از اور فیتینگ باشه و برای رفعش از dropout = ۰٫۵ استفاده کردم ولی باز هم نتیجه بد تر شد.



پس بهترین نتیجه زمانی حاصل شد که از مدل ارائه شده در منبع استفاده کردم. برای بهبود میتوان از مدل بهتری برای امبد کردن کلمات استفاده کرد ولی به علت پایان وقت پروژ] و طولانی بودن زمان توکایز کردن از آن صرف نظر کردم.