

به نام خدا



دانشگاه صنعتی امیر کبیر  
(پلی تکنیک تهران)

پروژه هفتم درس یادگیری عمیق

آشنایی با شبکه ترانسفورمر و مدل‌های بنیادی در یادگیری عمیق

استاد درس: دکتر صفابخش

نگارش: زهرا اخلاقی

شماره دانشجویی: ۴۰۱۱۳۱۰۶۴

زمستان ۱۴۰۲

## فهرست مطالب

2.....	(الف)
4.....	(ب)
6.....	(ج)

## الف)

تابع زیر `cls_token_embedding` را برمی‌گرداند که بردار ۷۵۶ بعدی توکن `CLS` را ارائه می‌کند و نشان‌دهنده ورودی است.

```
class TextEncoder(nn.Module):
    def __init__(self, model_name=CFG.text_encoder_model, pretrained=CFG.pretrained,
trainable=CFG.trainable):
        super().__init__()
        if pretrained:
            self.model = DistilBertModel.from_pretrained(model_name)
        else:
            self.model = DistilBertModel(config=DistilBertConfig())

        for p in self.model.parameters():
            p.requires_grad = trainable

    # we are using the CLS token hidden representation as the sentence's embedding
    self.target_token_idx = 0

    def forward(self, input_ids, attention_mask):
        output = self.model(input_ids=input_ids, attention_mask=attention_mask)
        last_hidden_state = output.last_hidden_state # This is a tensor of size (BATCH_SIZE,
num_words, 768)
        ### TO DO: Just one Line ####
        ### You shoud return the embeding of the CLS token. CLS token index is
self.target_token_idx
        cls_token_embedding = last_hidden_state[:, self.target_token_idx, :]

        return cls_token_embedding
```

با توجه به اینکه ابعاد تصویر به دست آمده ۲۰۴۸ و ابعاد متن برابر با ۷۶۸ می‌باشد و با یکدیگر قابل مقایسه نیستند، با استفاده از کد زیر با استفاده از یک MLP به فضای ۱۲۸ بعدی نگاشت می‌شوند که هر دو ابعاد برابر داشته باشند و با یکدیگر قابل مقایسه باشند.

ساختار MLP به صورت زیر می‌باشد:

یک لایه خطی برای نگاشت ابعاد ورودی به بعد مد نظر.

یک لایه `dropout` برای منظم سازی.

یک لایه LayerNorm لایه برای پایداری.

یک فعال سازی ReLU برای غیر خطی بودن.

یک لایه خطی نهایی برای اطمینان از اینکه خروجی بعد مورد نظر را دارد.

```
class ProjectionHead(nn.Module):
    def __init__(self,
                 embedding_dim, # the size of the input vector (2048 for images and 768 for texts)
                 projection_dim=CFG.projection_dim, # the size of the output vector (Use 128)
                 dropout=CFG.dropout
                ):
        """
        Whole Class
        It should be an MLP that project embedding_dim to projection_dim. Dont forget to use
        activation functions, layer norms and dropouts!
        super().__init__()

        self.projection = nn.Sequential(
            nn.Linear(embedding_dim, projection_dim),
            nn.Dropout(dropout),
            nn.LayerNorm(projection_dim),
            nn.ReLU(),
            nn.Linear(projection_dim, projection_dim) # Final projection to desired dimension
        )

    def forward(self, x):
        projected_x = self.projection(x)
        return projected_x
```

هدف از کد زیر تبدیل ویژگی‌های تصویر و متن به فضای تعبیه شده مشترک با 128 بعد می‌باشد:

```
### TO DO ###
### Project both features into 128-dimentional space using above defined ProjectionHeads
image_embeddings = self.image_projection(image_features)
text_embeddings = self.text_projection(text_features)
```

: امتیازات شباهت بین تعبیه‌های تصویر و متن را محاسبه می‌کند. label

: به ترتیب امتیازات شباهت را در تعبیه‌های تصویر و متن محاسبه می‌کند. loss\_j و loss\_i

: ماتریس هدف نرمال‌سازی شده با softmax را برای یادگیری متضاد ایجاد می‌کند. targets

: ضرر آنتروپی متقابل بین label و targets را محاسبه می‌کند. cross\_entropy

: ضرر را در تمام عناصر دسته میانگین می‌کند. loss.mean

هدف از کد زیر محاسبه تابع ضرر که آموزش مدل را هدایت می‌کند و در یادگیری هم‌ترازی بین تصاویر و متن کمک می‌کند.

```
### TO DO ###
### Calculate loss and return it. You can set the temperture parameter to 1. (See figure 2 of
the homework)

label = torch.matmul(text_embeddings, image_embeddings.T) / self.temperature
loss_i = torch.matmul(image_embeddings, image_embeddings.T)
loss_j = torch.matmul(text_embeddings, text_embeddings.T)
targets = F.softmax((loss_i + loss_j) / 2 * self.temperature, dim=-1)
loss = (cross_entropy(label.T, targets.T, reduction='none') +
+ cross_entropy(label.T, targets.T, reduction='none')) / 2.0
```

(ب)

در کد زیر به ازای همه تصاویر ورودی با مدلی که از قبل آموزش دیده است، به بعد مورد نظر نگاشت می‌شود و در ذخیره می‌شوند و در نهایت به tensor valid\_image\_embedding تبدیل می‌شود:

```
### TO DO ###
### Loop through validation dataloader and get 128-dim embedding of all images
with torch.no_grad():
    for batch in tqdm(valid_loader):

        image_features = model.image_encoder(batch["image"].to(CFG.device))
        image_embeddings = model.image_projection(image_features)
        valid_image_embeddings.append(image_embeddings)
return model, torch.cat(valid_image_embeddings)
```

در تابع زیر برای جمله ورودی embedding آن با تصاویر مقایسه شده (ضرب داخی) و n شبیه‌ترین تصاویر در خروجی نمایش داده می‌شوند:

```
def find_matches(model, image_embeddings, query, image_filenames, n=9):
    tokenizer = DistilBertTokenizer.from_pretrained(CFG.text_tokenizer)
    encoded_query = tokenizer([query])
    batch = {
        key: torch.tensor(values).to(CFG.device)
        for key, values in encoded_query.items()
    }
    with torch.no_grad():
        text_features = model.text_encoder(
            input_ids=batch["input_ids"], attention_mask=batch["attention_mask"]
        )
        text_embeddings = model.text_projection(text_features)
    ### TO DO ###
    ### Find and plot n nearest image to given query
    # Calculate similarity scores between text embedding and all image embeddings
    similarity_scores = torch.matmul(image_embeddings, text_embeddings.T)
    # Find the indices of the n most similar images
    top_n_indices = torch.topk(similarity_scores.T, n*5)[1].cpu().numpy()
    images = [image_filenames[idx] for idx in top_n_indices[0]]
    images = list(set(images))

    # Plot the n most relevant images
    fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 15))
    for i, ax in enumerate(axes.flat):
        if i < n:
            image_index = top_n_indices[0][i]
            image_filename = image_filenames[image_index]
            image = plt.imread(f"{CFG.image_path}/{images[i]}")
            ax.imshow(image)
            ax.set_title(image_filename)
            ax.axis("off")
    plt.tight_layout()
    plt.show()
```

خروجی برای جمله "This is a dog" به صورت زیر می‌باشد:



(ج)

برای استفاده از مدل CLIP آموزش دیده شده از تکه کد زیر استفاده می شود، وزنهای آن ثابت است و تغییری نمی کند:

```
# Load CLIP pre-trained weights and set up the processor
# Load the model
device = "cuda" if torch.cuda.is_available() else "cpu"
clip_model, clip_preprocess = clip.load('ViT-B/32', device)

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

دیتاست خواسته شده به صورت زیر گرفته می شود و پیش پردازش های مدل clip روی آن اعمال می شود

```
from torchvision.datasets import CIFAR10
import torchvision.transforms as transforms
# Load the dataset
root = os.path.expanduser("~/cache")
train = CIFAR10(root, download=True, train=True, transform=clip_preprocess)
test = CIFAR10(root, download=True, train=False, transform=clip_preprocess)
```

تابع زیر با استفاده از مدل clip تصاویر ورودی را به بعد ۵۱۲ نگاشت می شوند و برداری ۵۱۲ بعدی برای همه تصاویر ساخته می شود و در نهایت خروجی tensor برای تصاویر و کلاس آنها در خروجی است

```
def get_features(dataset,model):
    all_features = []
    all_labels = []

    with torch.no_grad():
        for images, labels in DataLoader(dataset, batch_size=100):
            features = model.encode_image(images.to(device))

            all_features.append(features)
            all_labels.append(labels)

    return torch.cat(all_features).cpu().numpy(), torch.cat(all_labels).cpu().numpy()
```

بعد از به دست آوردن برداری ۵۱۲ بعدی برای هر تصویر، یک لایه تمام متصل اضافه می شود که ورودی آن ۵۱۲ به اندازه تصویر و خروجی ۱۰ به اندازه تعداد کلاس ها در دیتاست می باشد. و روی خروجی تابع softmax اعمال می شود:

```

: class Classifier(nn.Module):
    def __init__(self, input_size, output_size):
        super(Classifier, self).__init__()
        self.fc = nn.Linear(input_size, output_size)

    def forward(self, x):
        # Apply softmax activation
        x = nn.functional.softmax(self.fc(x))
        return x

```

مدل اضافه شده به صورت زیر آموزش داده می‌شود:

```

: num_epochs = 5
for epoch in range(num_epochs):
    for i in range(len(train_features)):
        features, labels = torch.Tensor(train_features[i]).to(device), torch.Tensor(train_labels[i]).to(device)

        # Forward pass
        outputs = classifier(features)

        # Compute loss
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        print(f'Epoch {epoch + 1}/{num_epochs}, Loss: {loss.item()}')

<ipython-input-175-55dda7985d8f>:8: UserWarning: Implicit dimension choice for softmax has been deprecated. Change call to include dim=x as an argument.
  x = nn.functional.softmax(self.fc(x))

Epoch 1/5, Loss: 2.014613389968872
Epoch 2/5, Loss: 2.2185089588165283
Epoch 3/5, Loss: 2.378230333328247
Epoch 4/5, Loss: 2.410783052444458
Epoch 5/5, Loss: 2.3916091918945312

```

در نهایت دقت مدل آموزش دیده شده که داده اولیه آن خروجی مدل clip میباشد به صورت زیر است:

```

# Evaluate on the test set
classifier.eval()
correct = 0
total = 0

with torch.no_grad():
    for i in range(len(test_features)):
        features, labels = torch.Tensor(test_features[i]).to(device), torch.Tensor(test_labels[i]).to(device)
        outputs = classifier(features)

        predicted = torch.argmax(outputs.data)
        value = torch.argmax(labels)
        total += 1
        correct += (predicted == labels).sum().item()

accuracy = correct / total
print(f'Test Accuracy: {accuracy * 100:.2f}%')

<ipython-input-175-55dda7985d8f>:8: UserWarning: Implicit dimension choice for softmax has been deprecated. Change call to include dim=x as an argument.
  x = nn.functional.softmax(self.fc(x))

Test Accuracy: 95.47%

```

Test Accuracy: 95.47%