期末大作业实验报告

简介

本次作业利用Pytorch搭建了基于CLIP模型原理的神经网络，借助对比学习的方法实现了电影海报内容与简介内容的对齐匹配。

数据处理与数据集搭建

数据读取

本部分采用助教提供的代码，将海报对应的电影名称与简介从所有电影中分离出来。

训练集和测试集的划分

本部分采用助教提供的代码，按照`train.txt`中的要求获取训练集，以全数据作为测试集。注意此时海报尚未被读入内存，数据集中是海报的路径名。

搭建数据集，对数据进行预处理

修改`MovieLensDataset`类

为了结合`Dataset`和`Dataloader`实现数据的**按需读取与预处理**，在`MovieLensDataset`中定义函数`process_poster`来对海报进行读取与大小调整（rescale），使数据符合ViT的输入要求。对电影简介不做过多处理，只进行小写转换，使其符合BERT的输入要求（`bert-base-uncased`只能处理小写字符数据）。

```
class MovieLensDataset(Dataset):
  def __init__(self, data_root, posters, intros, transform, **kwargs):
      self.data_root = data_root
      self.posters = posters
      self.intros = intros
      self.transform = transform

  def __len__(self):
      return len(self.posters)

  def __getitem__(self, index):

      poster_path = os.path.join(self.data_root, self.posters[index])
      poster = self.process_poster(poster_path)
      # 将intro转换为小写，使其能被BERT处理
      intro = self.intros[index].lower()

      return poster, intro

  def process_poster(self, path, **kwargs):

      poster = cv2.imread(path)
      poster = cv2.cvtColor(poster, cv2.COLOR_BGR2RGB)
```

```
        # 对海报进行预处理，使之符合ViT的输入要求
        poster = Image.fromarray(poster)  # 转换为 PIL.Image
        poster_processed = self.transform(poster)

        return poster_processed

transform = transforms.Compose([
  transforms.Resize((224, 224)),
  transforms.ToTensor(),
])

data_root = './poster'
dataset_train =MovieLensDataset(
  data_root=data_root,
  posters=posters_train,
  intros=intros_train,
  transform = transform
)
dataset_test = MovieLensDataset(
  data_root=data_root,
  posters=posters_test,
  intros=intros_test,
  transform = transform
)
```

定义数据加载器

完成数据集的定义后，使用`Dataloader`加载数据，`Dataloader`调用`MovieLensDataset`中的`__get_item__`方法获取数据并完成数据的预处理。

模型设计与训练

问题建模与分析

问题要求实现图文匹配，可以建模为**最小化**海报与对应简介之间的距离，而**最大化**海报与其他简介之间的距离。因此可以通过搭建`poster_encoder`和`intro_encoder`分别提取海报与简介的特征，之后计算所有海报与所有简介之间的距离即可得到距离矩阵。

模型设计

基于CLIP模型的原理

根据上述分析，我们可以考虑通过神经网络进行特征提取。与分类/回归问题不同，该问题中我们并没有明确的标签来判断模型预测结果的准确性，实际上我们希望让模型学习到海报与简介之间的匹配关系，匹配/不匹配可以通过对比的方式来定义，即输入一组海报和一组简介，使模型输出该组海报与该组简介两两之间的匹配度，定义合适的损失函数来训练模型，实现**最大化**海报与对应简介的匹配度，同时**最小化**与其他简介的匹配度。这里我们基于CLIP模型提供的思路设计了模型，如下所示：

```
class MyModel(nn.Module):
    def __init__(self, **kwargs):
        super(MyModel, self).__init__()
```

```
        self.t = nn.Parameter(torch.tensor(1.0))


        # 初始化图片encoder：使用预训练的ViT
        self.poster_feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-
base-patch16-224', do_rescale = False)
        self.poster_encoder = ViTModel.from_pretrained('google/vit-base-patch16-224')
        # 调整图片编码维度
        self.poster_fc = nn.Linear(self.poster_encoder.config.hidden_size, 128)


        # 初始化文本encoder
        self.bert_model = BertModel.from_pretrained('bert-base-uncased')
        self.bert_tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
        # 调整文本编码的维度
        self.text_fc = nn.Linear(768, 128)


        self.poster_feature_extractor
        for param in self.poster_encoder.parameters():
            param.requires_grad = False

        for param in self.bert_model.parameters():
            param.requires_grad = False


    def forward(self, poster, intro):
        poster_input= self.poster_feature_extractor(images=poster,
return_tensors="pt").to(device)
        poster_output = self.poster_encoder(**poster_input)
        poster_feature = poster_output.last_hidden_state[:, 0, :]
        poster_feature = self.poster_fc(poster_feature)

        intro_inputs = self.bert_tokenizer(intro, return_tensors="pt", padding=True,
truncation=True, max_length=128).to(device)
        intro_outputs = self.bert_model(**intro_inputs)
        intro_feature = intro_outputs.last_hidden_state[:, 0, :]
        intro_feature = self.text_fc(intro_feature)

        norm_poster_feature = functions.normalize(poster_feature, p=2, dim=1)
        norm_intro_feature = functions.normalize(intro_feature, p=2, dim=1)
        logits = (norm_poster_feature @ norm_intro_feature.T) * torch.exp(self.t)

        return logits
```

损失函数定义

```
def contrastive_loss(logits, n):
    labels = torch.arange(n).to(logits.device)
    loss_poster = functions.cross_entropy(logits, labels)
    loss_intro = functions.cross_entropy(logits.T, labels)
    loss = (loss_poster + loss_intro)/2
    return loss
```

损失函数采用对比损失函数，将模型输出的`logits`矩阵（实际上是海报特征与简介特征经过可训练参数`t`缩放后的余弦相似度矩阵）看作样本的相似度分数。`labels`代表了理想情况下相似度矩阵的对角线，通过计算`logits`与标签的交叉熵损失，确保每个海报的`logits`与其对应的正确标签（对应的简介）尽可能匹配；而计算`logits`的转置与标签的交叉熵损失则可以确保简介的`logits`与其对应的正确标签（对应的海报）尽可

能匹配。

距离函数定义

```
full_poster_features = []
full_intro_features = []
model.eval()
for batch, (posters, intros) in enumerate(test_dataloader):
    posters = posters.to(device)
    poster_inputs = model.poster_feature_extractor(images=posters,
return_tensors="pt").to(device)
    poster_outputs = model.poster_encoder(**poster_inputs)
    poster_features = poster_outputs.last_hidden_state[:, 0, :]
    poster_features = model.poster_fc(poster_features)

    intro_inputs = model.bert_tokenizer(intros, return_tensors="pt", padding=True,
truncation=True, max_length=128).to(device)
    intro_outputs = model.bert_model(**intro_inputs)
    intro_features = intro_outputs.last_hidden_state[:, 0, :]
    intro_features = model.text_fc(intro_features)

    full_poster_features.append(poster_features)
    full_intro_features.append(intro_features)

full_poster_features = torch.cat(full_poster_features)
full_intro_features = torch.cat(full_intro_features)
D_test = torch.mm(full_poster_features, full_intro_features.T)
D_test = D_test / (torch.norm(full_poster_features, dim=1, keepdim=True) *
torch.norm(full_intro_features, dim=1))
D_test = -D_test # 相似度 -> 距离
```

根据模型的训练结果，一个自然的想法是在测试时将**所有海报与标签的组合**输入模型，输出二者的相似度，再转换为距离（如取负等）填入距离矩阵。但由于数据量大（2938*2938），实际上难以在有限时间内完成计算。实验过程中还尝试了缓存中间量（如已读入内存的海报图像和已经在训练时计算出的特征），但测试时间仍然很长。因此不能直接通过模型输出来计算距离。观察模型的前向传播过程可以发现，实际上我们只需要使用模型的两个`encoder`分别提取所有海报和简介的特征向量，最后计算余弦相似度并转换即可得到距离矩阵。虽然推测这种方法效果可能不如直接使用模型计算得到的相似度，但仍是可行的且大大缩短了测试时间。

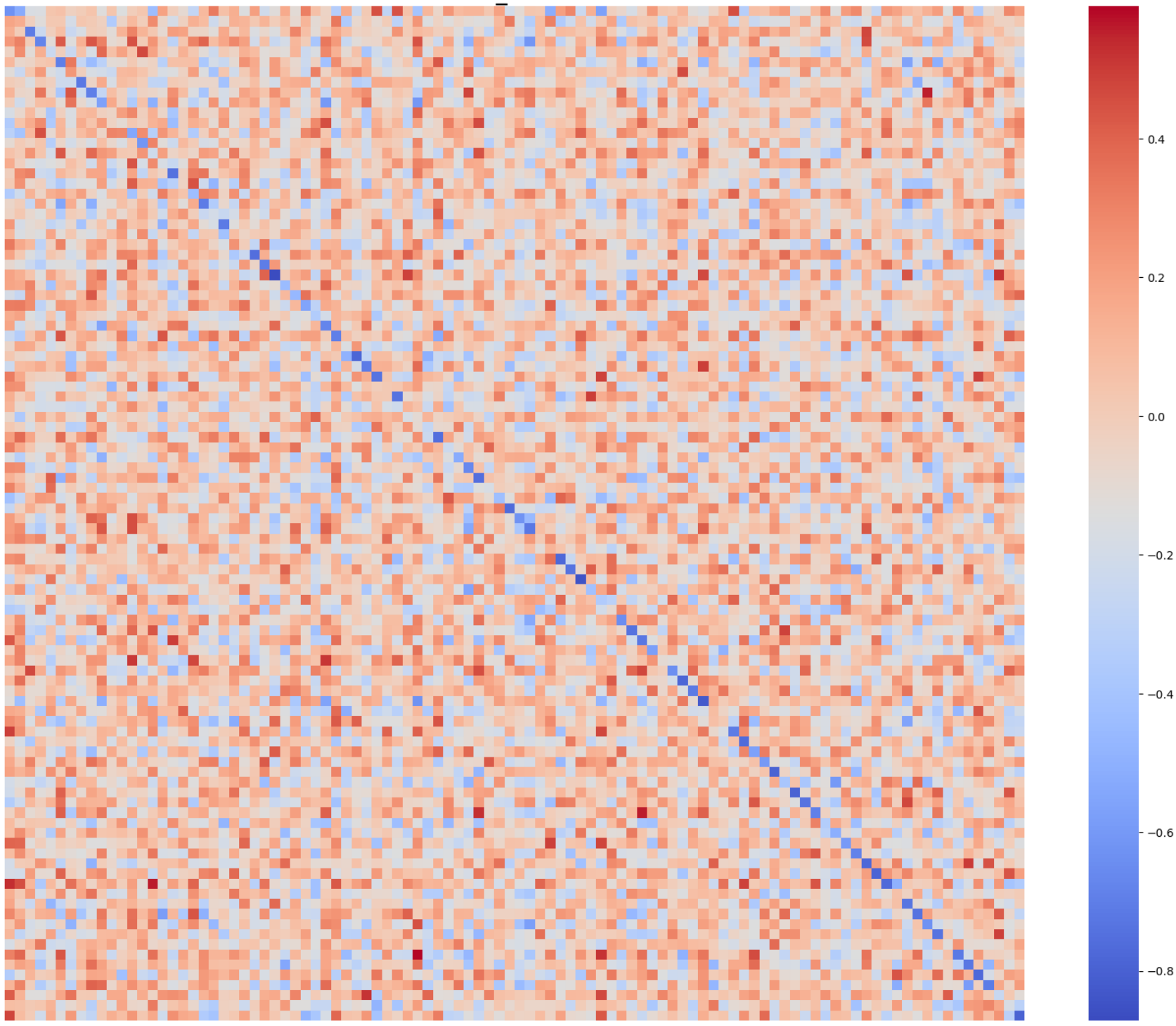
测试结果展示与分析

测试结果

为了得到匹配准确率最高的模型，在较低的学习率下测试了每个epoch训练结束后模型在测试集上的匹配准确率，较高的模型匹配准确率为

```
Accuracy on dim 0: 66.68%
Accuracy on dim 1: 67.43%
```

对距离矩阵`D_test`部分数据的可视化结果如下：



可以观察到距离矩阵对角元总体小于非对角元，可见模型的训练是有效的。 将ViT模型替换为ResNet50模型，总体效果略差于当前模型，两个维度的准确率均在65%左右。

模型的泛化能力

为了观察模型的泛化能力，我们可以统计模型在训练集和测试集（不包括训练集）数据上的匹配准确率，统计结果如下：

```
Train dataset accuracy on dim 0: 100.00%
Test dataset (excluding train dataset) accuracy on dim 0: 11.18%
Train dataset accuracy on dim 1: 99.94%
Test dataset (excluding train dataset) accuracy on dim 1: 18.71%
```

统计结果表明，模型的泛化能力不强，测试集中的非训练集样本大部分匹配都不够准确，原因可能为不同电影的海报和简介风格差异较大，模型在训练集上学习到的“匹配方法”可能难以正确应用在测试集上。同时根据文献，CLIP模型在MNIST数据集上的表现能力同样不佳，因为训练集中数据没有和MNIST数据集类似的数据样本。可见该类多模态对比学习模型可能在一定程度上都存在着泛化能力不足的问题。

潜在的改进方法

解冻预训练模型的部分参数

为了加快训练速度，模型冻结了加载的预训练模型的参数，但事实上ViT模型中有一些参数似乎并没有经过预训练，因此解冻部分参数可能可以增强模型的泛化能力，提高准确度。

增加模型输入每张海报对应的简介数量

当前模型每个batch输入多张海报和这些海报对应的简介，可以考虑将相似题材/风格/类型的电影简介加入当前电影的简介，使模型能够更好地获取海报风格-电影类型-电影简介之间的匹配关系，可能可以提升模型的泛化能力。

从海报中获取更多信息

类似地，也可以考虑提取海报中的文本信息来增强海报的特征，尝试融合文本特征和海报图像特征来提升模型对海报和简介的匹配能力。