- 姓名：吕奇澹
- 学号：SA23006171

实验要求

实验要求 编写RNN的语言模型，并基于训练好的词向量，编写RNN模型用于文本分类·请助教准备相关数据集

实验过程

数据集处理

```
class ReviewsDataset(Dataset):
    def __init__(self, texts, stars, seq_length=300):
        self.texts = texts
        self.stars = stars
        self.seq_length = seq_length

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

    def __getitem__(self, idx):
        text = self.texts[idx]
        # 将文本截断或填充到指定长度
        if len(text) > self.seq_length:
            text = text[:self.seq_length]
        else:
            text += [vocab_to_int['<PAD>']] * (self.seq_length - len(text))
        return torch.tensor(text, dtype=torch.long),
torch.tensor(self.stars[idx],

 dtype=torch.long)
```

模型搭建

我们使用了自己搭建的RNN模型进行实验

```
class RNN(nn.Module):
    def __init__(self, vocab_size, n_layers, dropout, norm, residual,
                 embedding_dim=1024, hidden_dim=1024, output_dim=5):
        super(RNN, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.residual = residual
        self.layers = nn.ModuleList()
        for _ in range(n_layers):
            layer = nn.Sequential(
                nn.LSTM(embedding_dim, hidden_dim, batch_first=True),
                nn.LayerNorm(hidden_dim) if norm else nn.Identity(),
                nn.Dropout(0.5) if dropout else nn.Identity(),
                nn.ReLU()
            )
            self.layers.append(layer)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        embedded = self.embedding(x)
        embedded_res = embedded
        output = embedded
        for layer in self.layers:
            output, _ = layer[0](output)  # LSTM层
            output = layer[1:](output)  # 其余层
            if self.residual:
                output = output + embedded_res  # 非原地操作
        output = self.fc(output[:, -1, :])
        return output
```

模型训练

我们定义train_once函数进行训练

```
def trainOnce(vocab_size, n_layers, dropout, norm, residual, lr_decay):
    model = RNN(vocab_size, n_layers, dropout, norm, residual).to(device)


    remaining_texts, test_texts, remaining_stars, test_stars = train_test_split(
        texts_ints, stars, test_size=1000, random_state=42
    )


    train_texts, val_texts, train_stars, val_stars = train_test_split(
        remaining_texts, remaining_stars, test_size=0.2, random_state=42
    )


    train_dataset = ReviewsDataset(train_texts, train_stars)
    val_dataset = ReviewsDataset(val_texts, val_stars)
    test_dataset = ReviewsDataset(test_texts, test_stars)

    batch_size = 896
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    if lr_decay:
        scheduler = LambdaLR(optimizer, lr_lambda=lambda epoch: 0.95**epoch)

    train_loss_list = []
    valid_loss_list = []
    for epoch in range(10):
        start = time.time()
        model.train()
        train_loss = 0.0
        for x, y in train_loader:
            optimizer.zero_grad()
            x = x.to(device)
            y = y.to(device, dtype=torch.long)
            output = model(x)
            loss = criterion(output, y)
            loss.backward()
            optimizer.step()
            train_loss += loss.item() * len(x)
        train_loss /= len(train_dataset)
        train_loss_list.append(train_loss)

        model.eval()
        valid_loss = 0.0
        with torch.no_grad():
            for x, y in val_loader:
                x = x.to(device)
                y = y.to(device, dtype=torch.long)
                output = model(x)
                loss = criterion(output, y)
                valid_loss += loss.item() * len(x)
            valid_loss /= len(val_dataset)
            valid_loss_list.append(valid_loss)
        end = time.time()
        print(
            f"epoch {epoch}, train loss {train_loss}, valid loss {valid_loss},
time {end-start}"
        )
        if lr_decay:
            scheduler.step()

    acc = calTop1Acc(model, val_loader)
    return train_loss_list, valid_loss_list, acc, model, test_loader
```

超参数搜索及结果

```
def searchHyperParameter(vocab_size):
    LayerList = [5, 4, 3]
    dropoutList = [True, False]
    NormList = [True, False]
    residualList = [True, False]
    lrDecayList = [True, False]
    # LayerList = [4]
    # dropoutList = [False]
    # NormList = [False]
    # residualList = [False]
    # lrDecayList = [False]
    bestAcc = 0
    bestParam = None
    for n_layer in LayerList:
        for dropout in dropoutList:
            for Norm in NormList:
                for residual in residualList:
                    for lrDecay in lrDecayList:
                        param = (vocab_size, n_layer, dropout, Norm, residual,
lrDecay)
                        print(param)
                        _, _, acc, _ = trainOnce(*param)
                        print(f"ACC {acc}\n")
                        if acc > bestAcc:
                            bestAcc = acc
                            bestParam = param
    print("\nbest Param:", bestParam)
    print("best Acc:", bestAcc)
    return bestParam
```

我们对LayerList, dropoutList, NormList, residualList, lrDecayList 进行超参数搜索具体范围为:

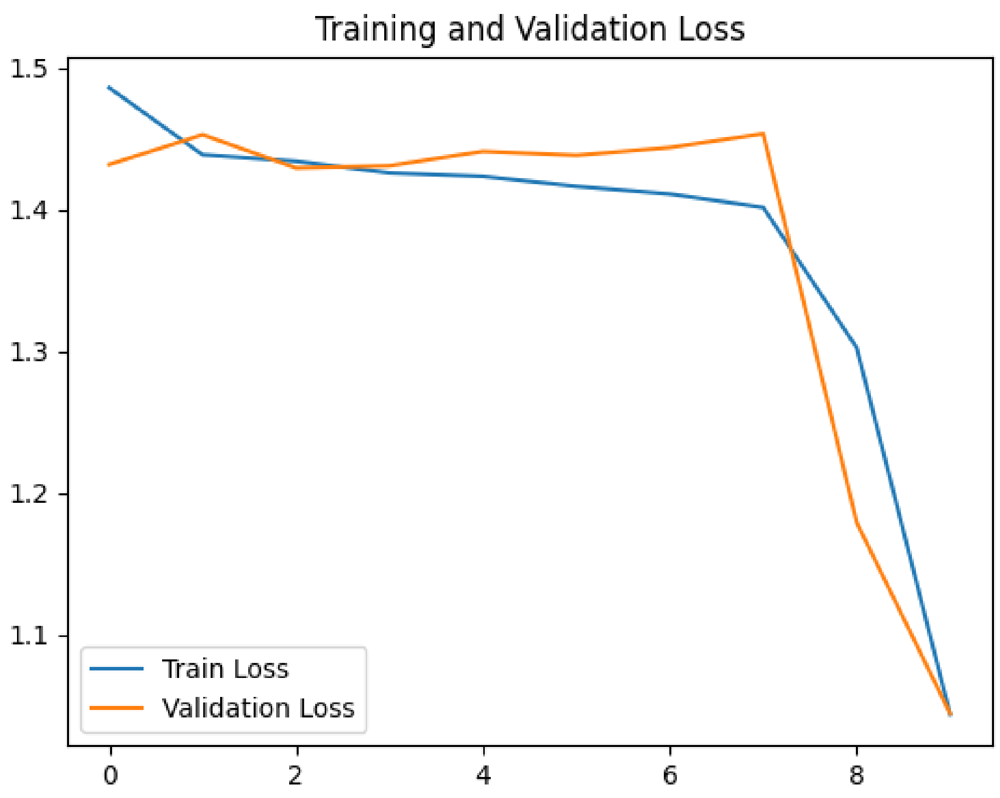
- LayerList = [6,5,4]
- dropoutList = [True, False]
- NormList = [True, False]
- residualList = [True, False]
- lrDecayList = [True, False]

超参数搜索的最终结果为:

- depthList in [6]
- normalizeList in [True]
- dropoutList in [True]
- residualList in [True]
- lrDecayList in [True]

实验结果

最优超参数对应的训练损失可视化图像为:



将最优超参数搜索结果在测试集上进行测试的最终结果为:
Test ACC: 0.556375

实验收获

1. dropout是一个可以很好防止过拟合的办法，没有dropoutout，现有的网络容易过拟合
2. batchnorm和dropout一起用并没有之前认为会导致效果变差的情况，至少这个实验没有出现
3. 越深的网络不一定能带来更好的结果，
4. RNN词表的构建需要思考的地方不少，包括, 等特殊token的补充是一个很重要的部分，并且在构建Reviewdataset的时候需要考虑到不同句子的长度不一样，这会导致句子转化为tokens的长度不一样，构建的时候要考虑截断操作将超过阈值的部分截断不足的padding，或许也可以考虑重载dataloader中的collate_fn。
5. 和长程关系有关的数据RNN难以建模，LSTM效果也不好，可能后续实际还是需要使用transformer架构进行更好的长程关系建模。