中国科学技术大学

University of Science and Technology of China

深度学习实验 3 报告

姓名：夏金杰

学号：SA25011291

2025 年 12 月 22 日

目录

1 任务表述

- 开发一个 RNN 语言模型，并使用训练好的词向量实现 RNN 模型用于文本分类。
- 使用 Yelp2013 数据集，将 test.json 用作测试集，并从 yelp_academic_dataset_review.json 中手动划分训练集和验证集。
- 将生成的训练集输入模型，进行损失计算与梯度传播，从而训练模型。建议使用框架提供的优化器来完成参数更新。记录训练集和验证集的损失，并绘制图表进行可视化。
- 在验证集上测试模型性能，以 **Top 1 准确率 (ACC)** 作为评估指标。调整 dropout、归一化、学习率衰减、残差连接和网络深度，重新训练、测试，并分析其对性能的影响。
- 选择在验证集上表现最佳的一组超参数，重新训练模型，并在测试集上进行测试（这应是实验中唯一一次在测试集上的测试）。记录测试结果（ACC）。

2 数据预处理

初始数据集样本为一段文本，标签为文本的评分，1-5 分。数据预处理分为 4 个步骤。

2.1 从训练集划分验证集

实验采用随机划分，按照 9:1 的比例从原始训练集中划分出验证集。

2.2 生成词表

实验定义 vocab 类实现词汇表的生成和原始文本到词表 ID 序列的映射，词汇表需要引入 2 个特殊词元：<PAD> 和 <UNK>，代表填充词元和未知词元，对应的 ID 分别为 0 和 1。关键代码和解释如下：

```
    class Vocab:
    def __init__(self, tokens, max_size, min_freq): # 根据tokens列表构建词表
        self.pad_token = '<PAD>' # padding token, id=0
        self.unk_token = '<UNK>' # unknown token, id=1
        all_tokens = []
        for text in tokens:
            all_tokens.extend(text.lower().split())
        token_counts = Counter(all_tokens) # 统计词频, key: token, value: count
        # 过滤低频词并按频率排序
        valid_tokens = [t for t, c in token_counts.items() if c >= min_freq]
        valid_tokens = sorted(valid_tokens, key=lambda x: token_counts[x], reverse=True)
            [:max_size]

        self.token2id = {self.pad_token: 0, self.unk_token: 1}
        for idx, token in enumerate(valid_tokens):
            self.token2id[token] = idx + 2
```

2.3 自定义数据集

由于模型需要 ID 序列作为单个样本的输入数据，所实验需要自定义 Dataset 子类实现原始文本到 ID 序列的映射；同时我们将标签改为 0-4，方便直接用输出的下标作为标签。关键代码和解释如下：

```
        def __getitem__(self, idx): # 定义模型输入样本和标签
        text, label = self.data[idx]
        input_ids = self.vocab.convert_text_to_ids(text, self.max_len)
        # 假设label是 1-5 星，转为 0-4
        label = int(label) - 1
        return torch.tensor(input_ids), torch.tensor(label)
```

2.4 生成数据批量

直接使用 DataLoader 生成数据批量，关键代码如下：

```
        dataset_full = YelpDataset(raw_train_data, vocab, base_config.max_len)
        train_dataset, val_dataset = random_split(dataset_full, [train_size, val_size])
        test_dataset = YelpDataset(raw_test_data, vocab, base_config.max_len)
        # 生成训练集，验证集dataset对象
        train_sampler = DistributedSampler(train_dataset, num_replicas=world_size, rank=
            rank, shuffle=True)
        val_sampler = DistributedSampler(val_dataset, num_replicas=world_size, rank=rank,
             shuffle=False)
        # 使用分布式训练，需要采样器，防止同样数据在多卡重复计算

        train_loader = DataLoader(train_dataset, batch_size=config.batch_size, sampler=
            train_sampler, num_workers=4, pin_memory=True)
        val_loader = DataLoader(val_dataset, batch_size=config.batch_size, sampler=
            val_sampler, num_workers=4, pin_memory=True)
```

3 模型架构实现

本实验采用双向 LSTM 作为骨干，最后使用 anntention 机制缓解 RNN 对于长文本的遗忘现象。

3.1 Attention 模块实现

Attention 使用 MLP 得到最终输出的每个向量的重要性权重，加权求和，关键代码如下：

```
        def forward(self, hidden_states):# 前向计算注意力权重并生成上下文向量
            # hidden_states: [batch, seq_len, hidden_dim]

            # energy: [batch, seq_len, hidden_dim]
            energy = torch.tanh(self.attn(hidden_states))

            # attention_scores: [batch, seq_len, 1]
            attention_scores = self.v(energy)

            # weights: [batch, seq_len, 1]
            weights = F.softmax(attention_scores, dim=1)
```

```
12
13            # context_vector: [batch, hidden_dim]
14            # sum(weights * hidden_states) across seq_len
15            context_vector = torch.sum(weights * hidden_states, dim=1)
16
17            return context_vector, weights
```

3.2 BiLstm+Attention 实现

实验实现标准的双向 BiLstm，在最后调用 attention 输出聚合了所有文本信息的向量用输入分类器，关键代码和解释如下：

```
1     def forward(self, x):
2         # x: [batch, seq_len]
3         out = self.embedding(x) # [batch, seq_len, embed_dim]
4         out = self.dropout(out)
5
6         for i, lstm in enumerate(self.lstm_layers):
7             prev_out = out
8             lstm_out, _ = lstm(out) # [batch, seq_len, hidden*2]
9
10            # 这里的残差连接应用在LSTM层之间
11            if self.use_residual:
12                # 调整prev_out维度以匹配lstm_out
13                res_in = self.residual_projections[i](prev_out)
14                out = lstm_out + res_in
15            else:
16                out = lstm_out
17
18            if self.use_layer_norm:
19                out = self.layer_norms[i](out)
20
21            out = self.dropout(out)
22
23        # Attention
24        context_vector, _ = self.attention(out) # [batch, hidden*2]
25
26        logits = self.fc(context_vector)
27        return logits
```

4 训练细节

实验实现了两种训练模式，单卡训练和多卡并行，单卡训练运行 train.py 文件，多卡并行运行 train_distributed.py 文件，通过 world_size 参数确定使用的 GPU 数量。训练使用交叉熵损失，adam 优化函数，训练时不变的超参数如下表所示：

表 1: 超参数配置表 (Hyperparameters Configuration)

参数 (Parameter)	值 (Value)	描述 (Description)
max_vocab_size	50000	最大词汇表大小
max_len	256	最大句子长度
min_freq	5	词汇表词元最小频次
batch_size	64	批次大小
epochs	15	训练轮数
learning_rate	0.001	学习率
device	cuda	使用设备 ('cuda' 或'cpu')
seed	42	随机种子
embed_dim	256	词嵌入维度
hidden_dim	256	LSTM 隐藏层维度

5 实验

所做的实验研究了不同 dropout 概率，是否使用层归一化，是否使用学习率衰减，是否使用残差连接，网络深度对训练集损失和精度以及验证集损失和精度的影响，下图给出一张所有实验的对比图。可以看到，在本次实验的所有超参数配置中，不使用残差连接，其余超参数基于基线配置明显取得最佳性能。下面详细分析所做对比实验。

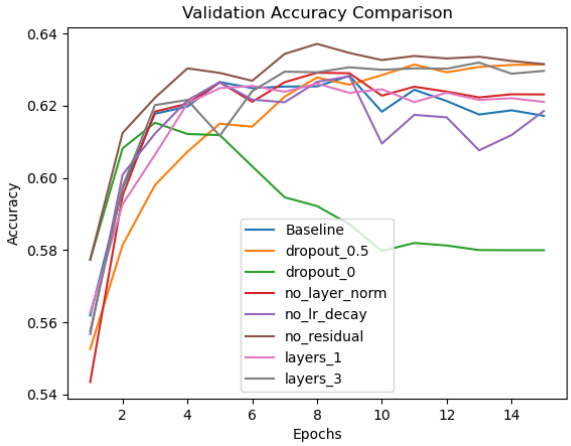


图 1: 所有实验验证精度对比

5.1 dropout 概率研究

实验采取 dropout0，0.3，0.5 的比率。当其值为 0 时，能够训练集的损失下降速度明显加快，最终精度甚至逼近 100%，但是验证损失却在后期大幅上涨，验证精度也在快速下降，过拟合严重。而后两者都能有效缓解模型过拟合，由于 dropout 为 0.3 时的验证精度更高，实验最终选取 0.3 的 dropout。细节如图 2 所示。

表 2: 基线配置 (Baseline Configuration)

参数 (Parameter)	值 (Value)	描述 (Description)
n_layers	2	LSTM 层数
dropout	0.3	Dropout 概率
use_layer_norm	true	是否使用 Layer Normalization
use_residual	true	是否使用残余连接
lr_decay	true	是否使用学习率衰减

5.2 层归一化研究

在本实验中，训练时是否使用层归一化未见明显影响，但使用层归一化在模型能加快模型在开始的几个 epich 的拟合，达到更高的验证精度。细节如图 3 所示。

5.3 学习率衰减策略

如果不使用学习率衰减，模型在训练后期过拟合验证，会损失模型的泛化能力。

5.4 残差连接

由于本实验最多使用 3 层 BiLstm，所以梯度消失的问题不严重，使用残差连接反而导致模型最后的验证精度下降，所以本实验决定不采用残差连接策略。

5.5 网络深度

观察图 1 可以明显看出 2 层验证精度最佳，3 层次之，1 层最差。

6 最佳超参数测试

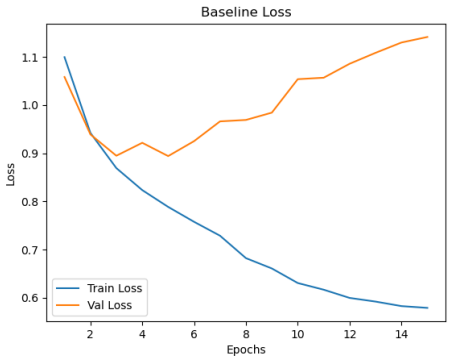
最终采取的超参数配置如表 3 所示：

表 3: 基线配置 (Baseline Configuration)

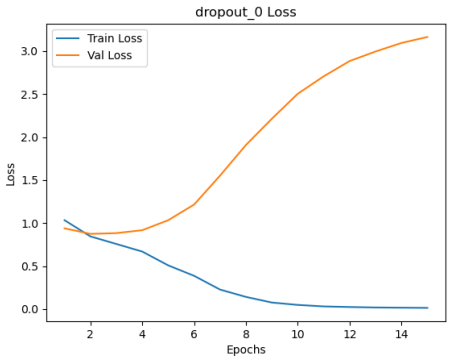
参数 (Parameter)	值 (Value)	描述 (Description)
n_layers	2	LSTM 层数
dropout	0.3	Dropout 概率
use_layer_norm	true	是否使用 Layer Normalization
use_residual	false	是否使用残余连接
lr_decay	true	是否使用学习率衰减

使用该超参数在最终的测试集取得的精度为 **79.7%**。

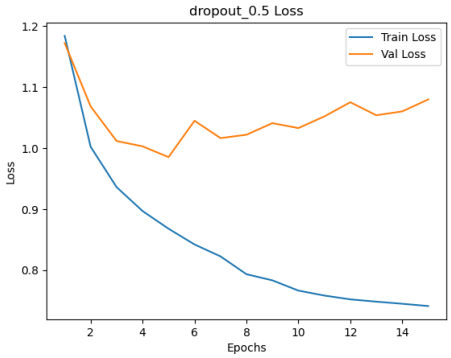
7 实验结果



(a) dropout0.3

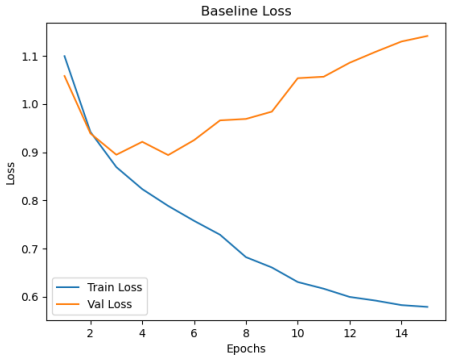


(b) dropout0

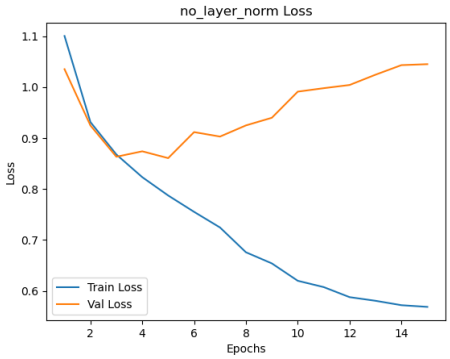


(c) dropout0.5

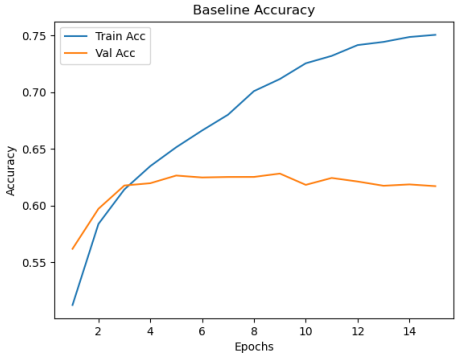
图 2: dropout 损失对比



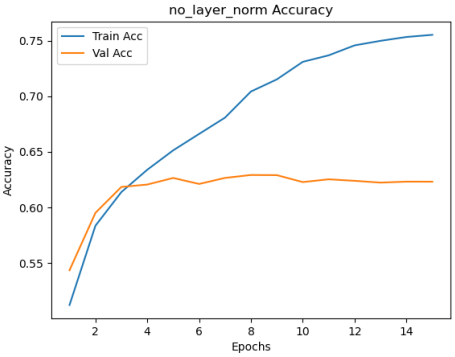
(a) 使用归一化损失



(b) 无归一化损失

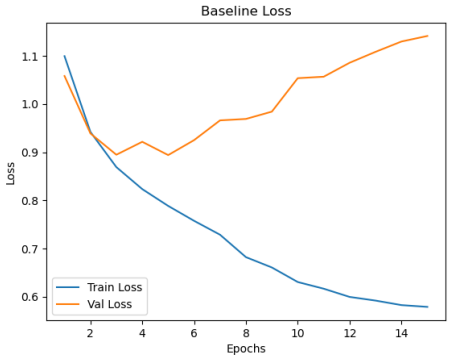


(c) 使用归一化精度

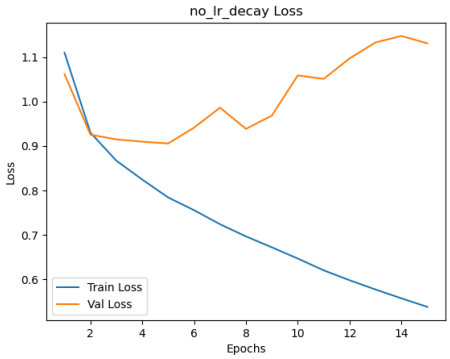


(d) 无归一化精度

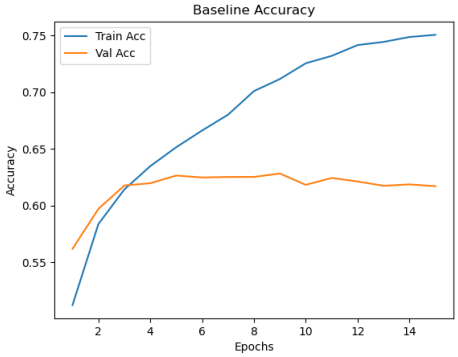
图 3: 归一化对比实验



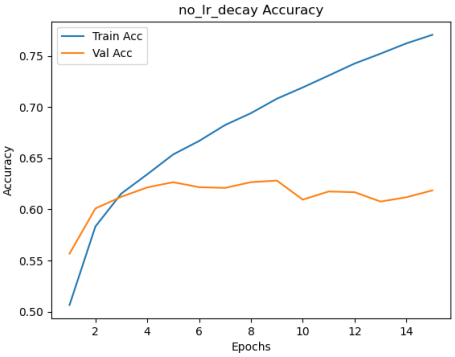
(a) 使用衰减损失



(b) 无衰减损失

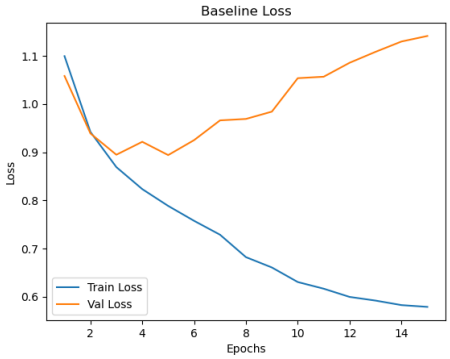


(c) 使用衰减精度

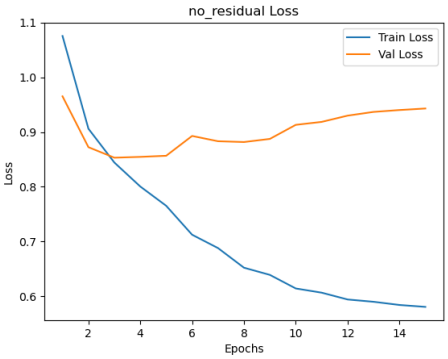


(d) 无衰减精度

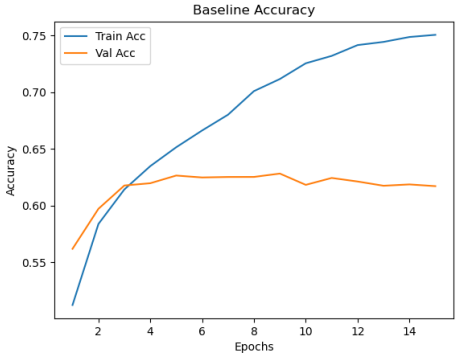
图 4: 衰减对比实验



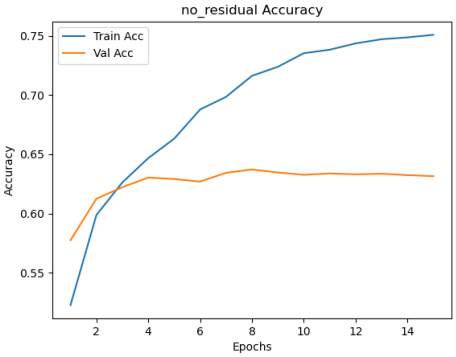
(a) 使用残差损失



(b) 无残差损失

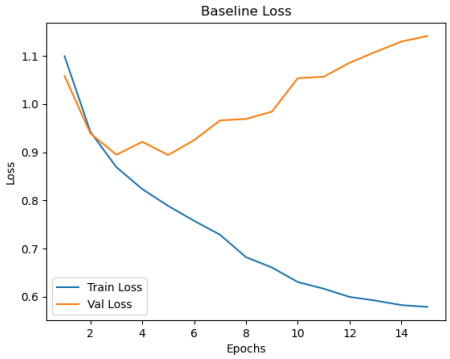


(c) 使用残差精度

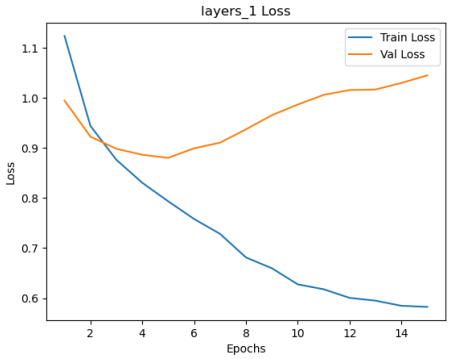


(d) 无残差精度

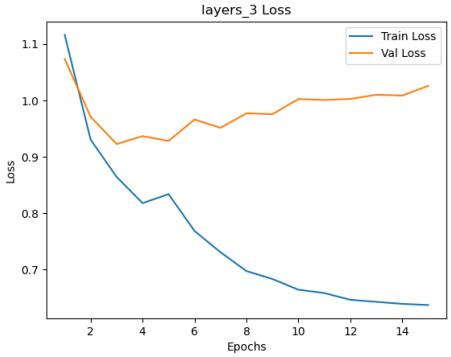
图 5: 残差对比实验



(a) 2 层 loss



(b) 1 层 loss



(c) 3 层 loss

图 6: layer 损失对比