人工智能导论 Lab-2 实验报告

褚砺耘 2023012471
致理-信计 31

2025 年 4 月 25 日

1 实验环境

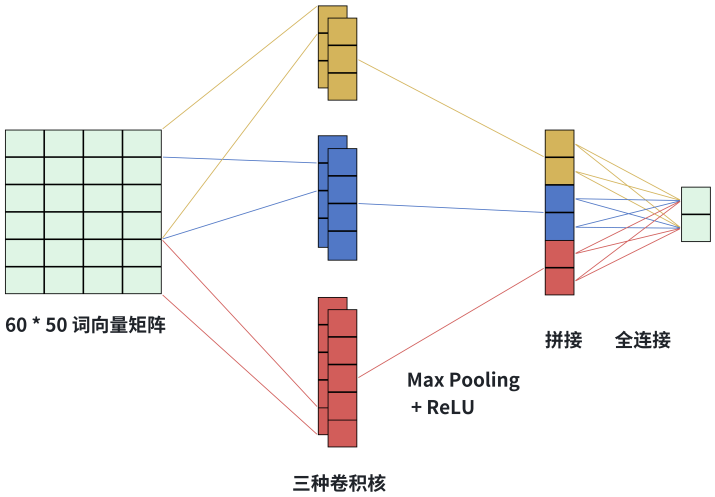
- 操作系统：macOS Sonoma 14.6.1
- Python 版本：Python 3.12.4
- CPU：Apple M3

2 CNN 模型

2.1 数据预处理

对于给定的数据集 train.txt, test.txt, validation.txt 首先加载数据，得到原始句子 + 标签的数据格式，然后对原始句子进行分词，并构建词表 `word2id:{"<PAD>":0, "<UNK>":1, ...}`，然后根据词表将句子转换为 ID 序列。我们设置 max_len = 60, 将所有序列设置为该长度 (超出截断, 不足补 0)。然后根据 Word2Vec 将每个词转换为长度为 50 的向量，并剔除无法识别的词语 (OOV words), 构建成 max_len ×50 的词向量矩阵。同时我们同样需要把代表情感正向与负向表示为二维张量：$[1,0]$ 与 $[0,1]$.

2.2 模型结构图



2.3 流程分析

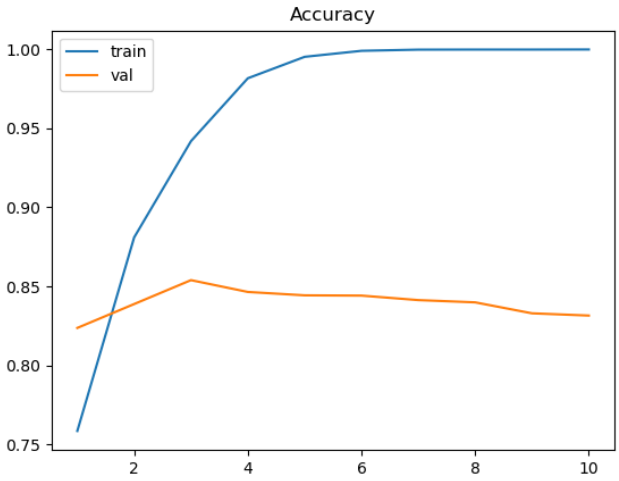
1. 输入句子：
   - 原始句子被转换成 max_len 长度的序列
   - 经过词嵌入层变成 shape 为 [batch_size, max_len, 50] 的张量
2. 卷积层：
   - 使用 3 个卷积核尺寸: 3, 4, 5
   - 每种卷积核使用 num_filiters = 100 个
   - 卷积核在“句子维度”上滑动，卷积的是局部的 n-gram
3. 激活 + 最大池化：
   - 每个 filter 经过 ReLU 激活后进行最大池化，池化后的结果是 1 个数
   - 每个 kerner_size 会输出 100 维的特征向量
4. 拼接：
   - 将三个卷积核输出的结果拼接起来，得到 300 维向量。
5. 全联接层
   - 送入具有两个神经元的全连接层，输出 [batch_size, 2] 的 logits 向量。使用 Softmax 激活函数。
6. 损失函数：使用交叉熵计算 loss。

2.4 实验结果

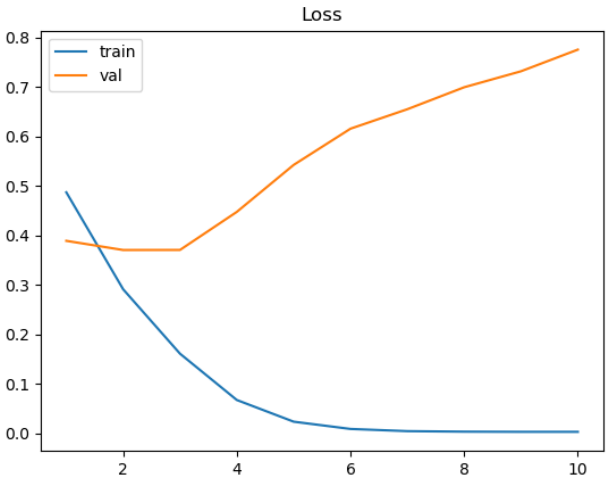
超参数如下：

超参数	数值
epoch	10
batch_size	128
learning_rate	2e-3
kernel_sizes	100
max_len	60

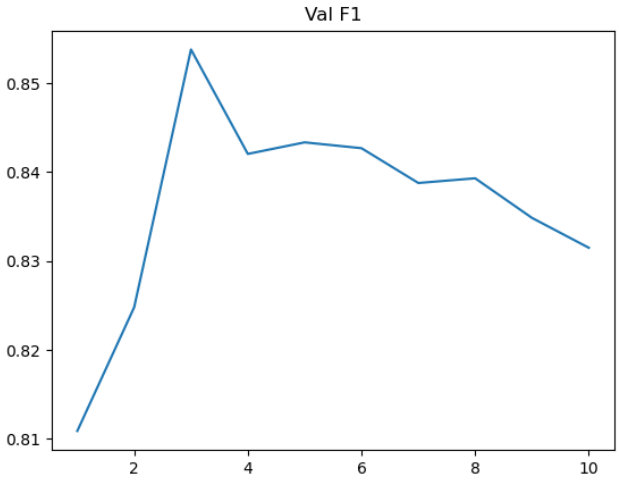
train、val accuracy:



train、val loss:



val F1:



在 test set 上的表现为：

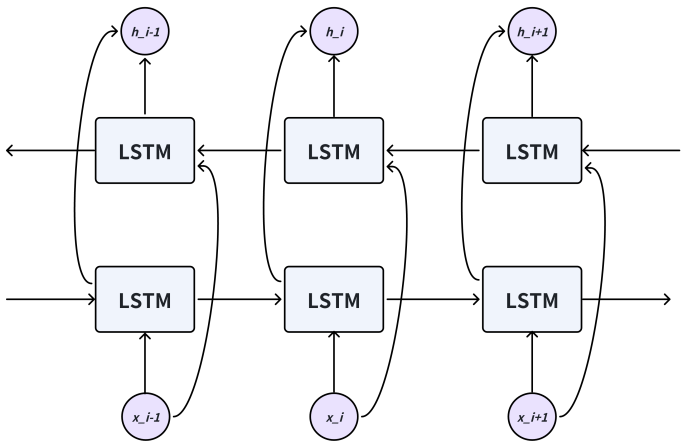
指标	数值
Loss	0.6261
Accuracy	0.8645
F1	0.8619

3 RNN-LSTM

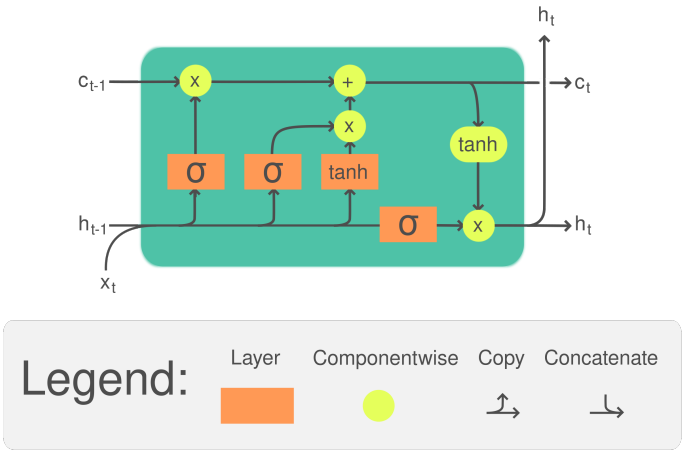
3.1 数据预处理

同上 CNN 模型的数据预处理过程。

3.2 模型结构图



如果是单向 lstm, 则只有向前传播的模块。



3.3 流程分析

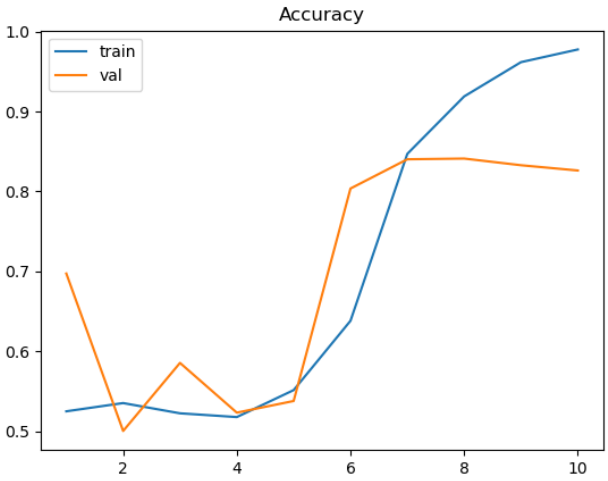
1. 输入句子：
   - 原始句子被转换成 max_len 长度的序列
   - 经过词嵌入层变成 shape 为 [batch_size, max_len, 50] 的张量
2. 输入 LSTM 模块：
   - 分别采用单向和双向的 LSTM 模型, num_layers = 1,hidden_size = 128
3. 输入全连接层：
   - 对于单向 LSTM, 直接对正向结果使用 128×2 的全连接层
   - 对于双向 LSTM, 将正向结果和反向结果拼接后使用 256×2 的全连接层
4. Softmax 激活、计算交叉熵损失、反向传播、更新。

3.4 实验结果

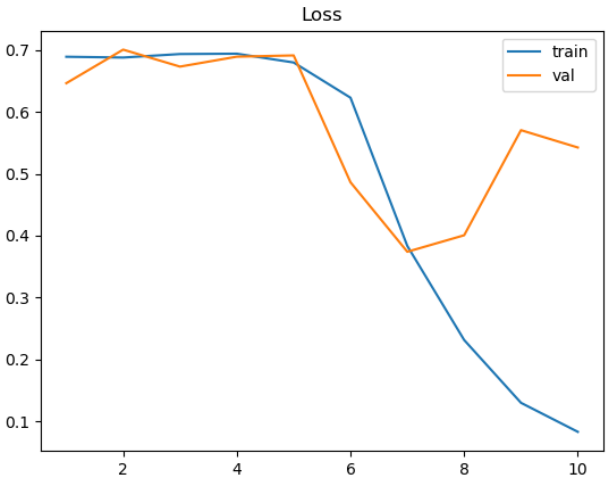
超参数如下：

超参数	数值
epoch	10
batch_size	128
learning_rate	2e-3
hidden_size	128
num_layers	1
max_len	60

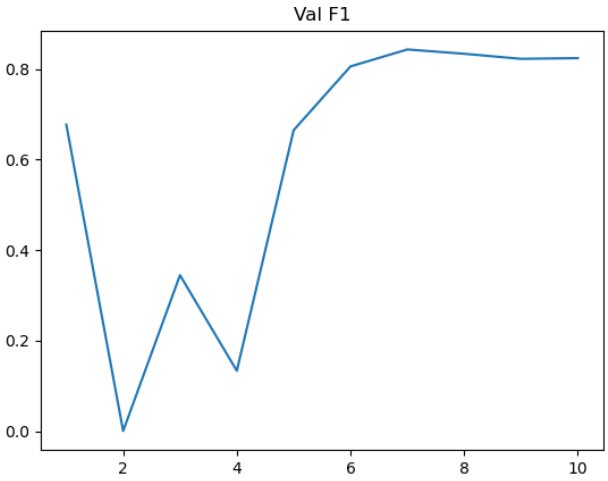
我们绘出单向 LSTM 的结果图像。train、val accuracy:



train、val loss:



val F1:



在 test set 上我们对比了单向和双向 LSTM 的结果：在 test set 上的表现为：

指标	单向 LSTM	双向 LSTM
Loss	0.7289	0.4461
Accuracy	0.8347	0.8049
F1	0.8338	0.7966

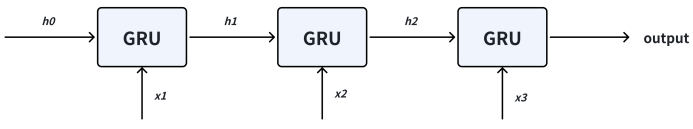
- 双向 LSTM 的 Loss 明显更低，说明它拟合得更充分
- 单向 LSTM 的 Accuracy 和 F1 更高，说明它在 test set 上泛化得更好

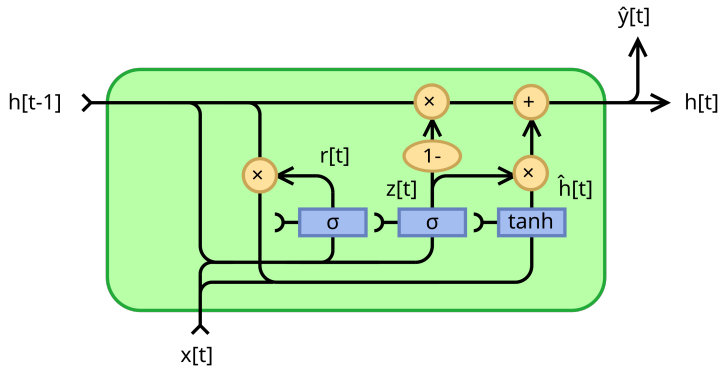
4 RNN-GRU

4.1 数据预处理

同上 CNN 模型的数据预处理过程。

4.2 模型结构图





4.3 流程分析

基本结构同 LSTM, 但是 GRU 只有 2 个门:

- 重置门: 控制当前收入和之前记忆的结合程度
- 更新门：控制保留多少过去信息，约等于遗忘门 + 输入门的组合

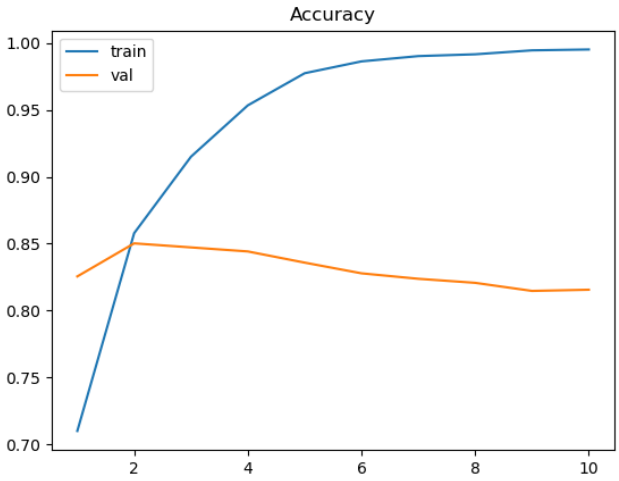
输出状态只有 h_t，没有记忆状态 c_t

4.4 实验结果

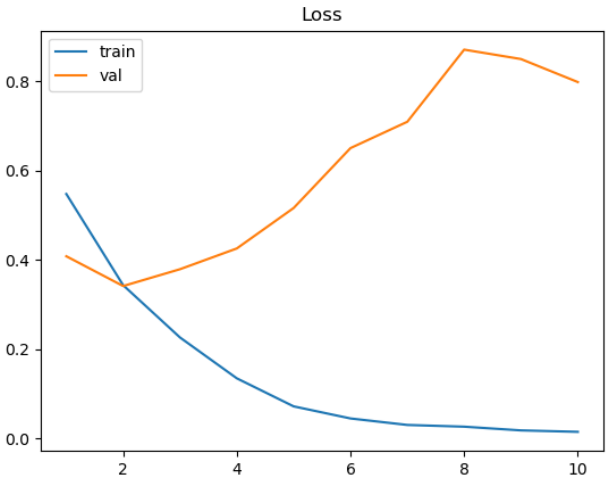
超参数如下：

超参数	数值
epoch	10
batch_size	128
learning_rate	2e-3
hidden_size	128
num_layers	1
max_len	60

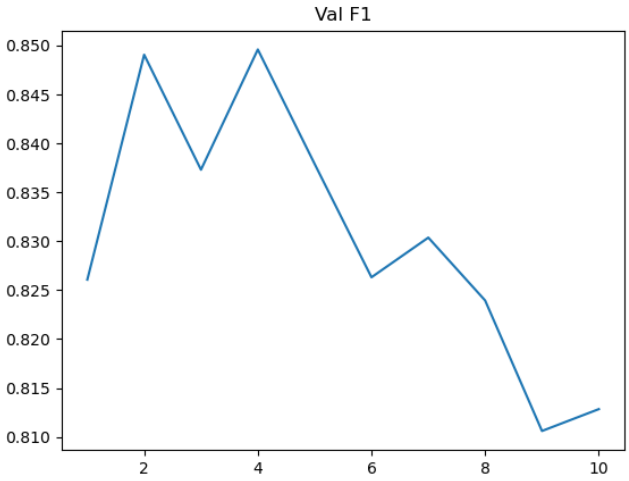
train、val accuracy:



train、val loss:



val F1:



在 test set 上的表现为：

指标	数值
Loss	0.7148
Accuracy	0.8347
F1	0.8365

5 BERT

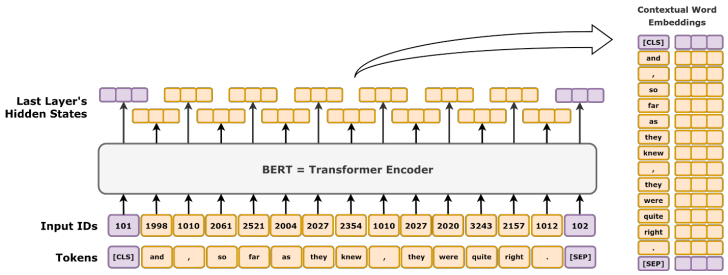
5.1 数据预处理

把数据集变成 HuggingFace BERT 能接受的 input_ids , attention_mask , labels 三张张量：

- input_ids: 句子每个 token 的词 id 组成的 tensor。长度过长截断, 否则填充为 0
- attention_mask: 有效位置标注 1, 填充位置标注 0
- labels: 句子的标签

然后直接返回 PyTorch DataLoader, 可直接用于训练和验证阶段。

5.2 模型结构图



5.3 流程分析

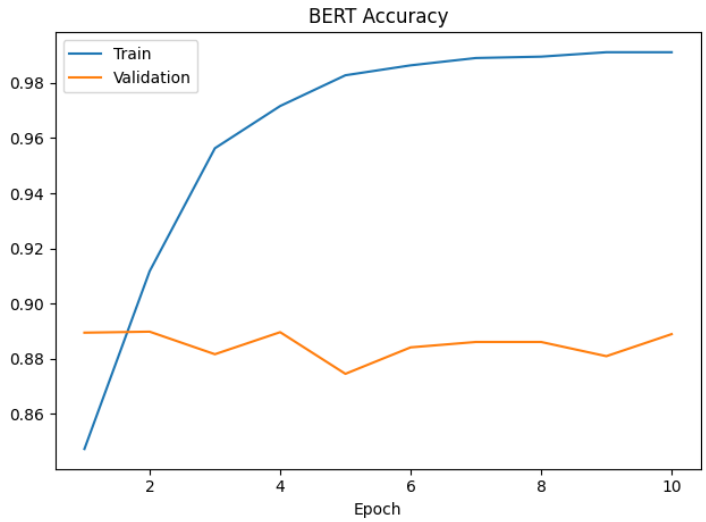
- 从 HuggingFace 模型库加载预训练的 bert 模型
- 将预处理的数据传入模型
- 取 [CLS] 表示向量，代表整个句子的聚合向量，代表句子级语义
- cls 送入分类器，经过 Dropout + Linear
- Softmax 后输出

5.4 实验结果

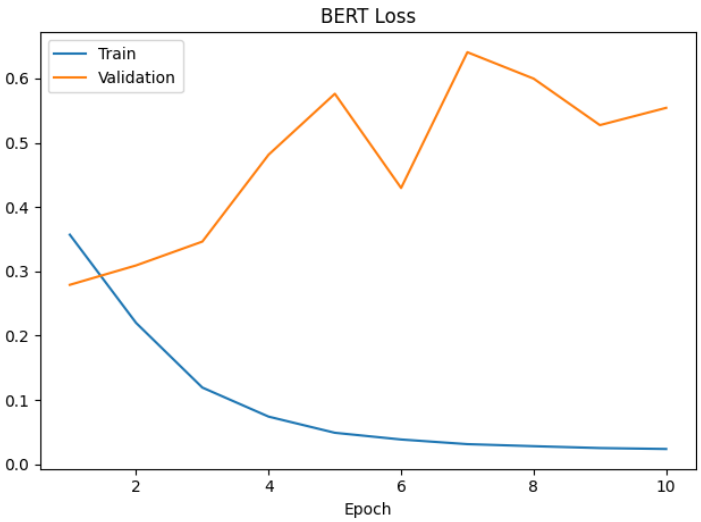
考虑到性能问题，本实验转移到另一台电脑的 RTX4080 上完成。超参数如下：

超参数	数值
epoch	10
batch_size	16
learning_rate	2e-5
max_len	128
dropout	0.3

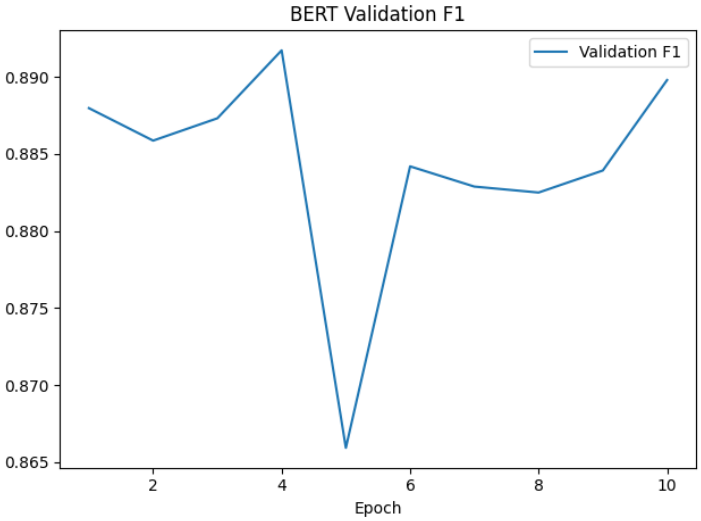
train、val accuracy:



train、val loss:



val F1:



在 test set 上的表现为：

指标	数值
Loss	0.5215
Accuracy	0.8943
F1	0.8966

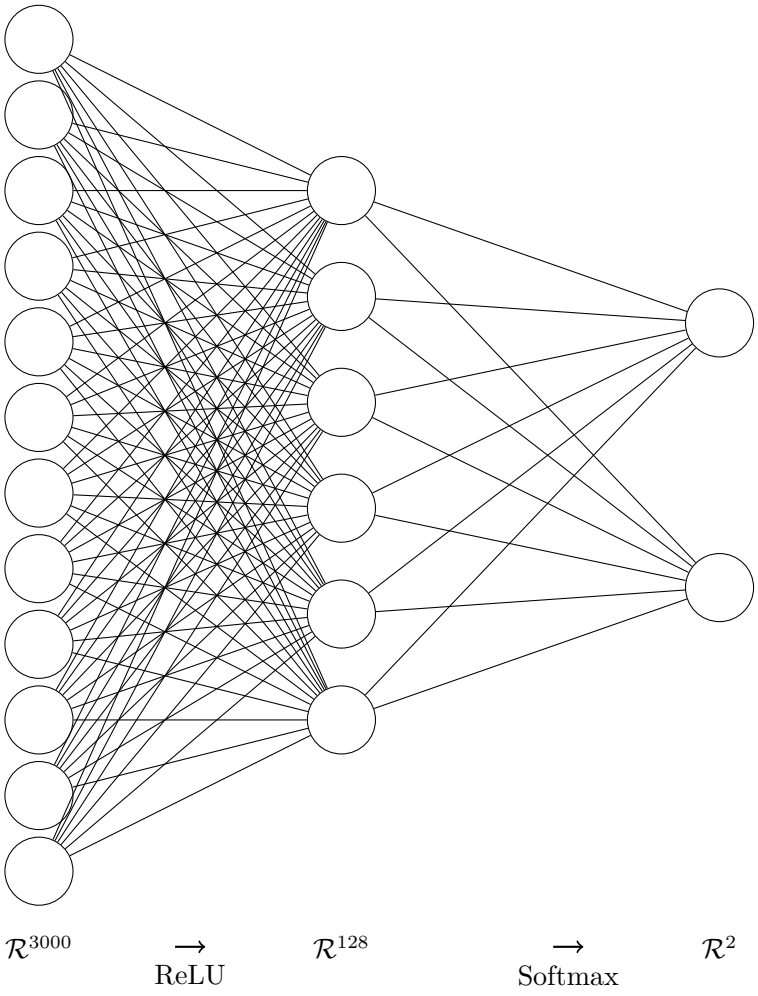
可以看到 Bert 模型的 F1 和 Accuracy 都接近了 0.9, 与前面的模型相比有极大提升。

6 Baseline: MLP

6.1 数据预处理

同上过程。

6.2 模型结构图



6.3 流程分析

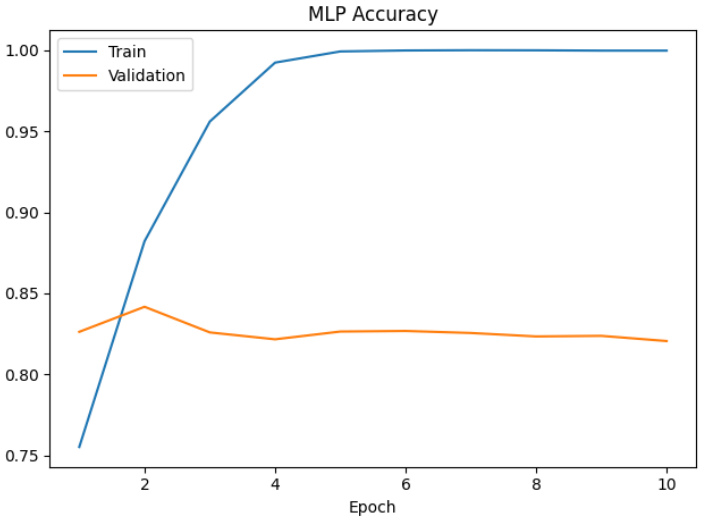
- 使用 max_len × 50 规模的输入，映射到 hidden_size 规模的隐藏层
- ReLU + Dropout
- 将隐藏层特征映射为 2 个神经元
- Softmax, 输出结果

6.4 实验结果

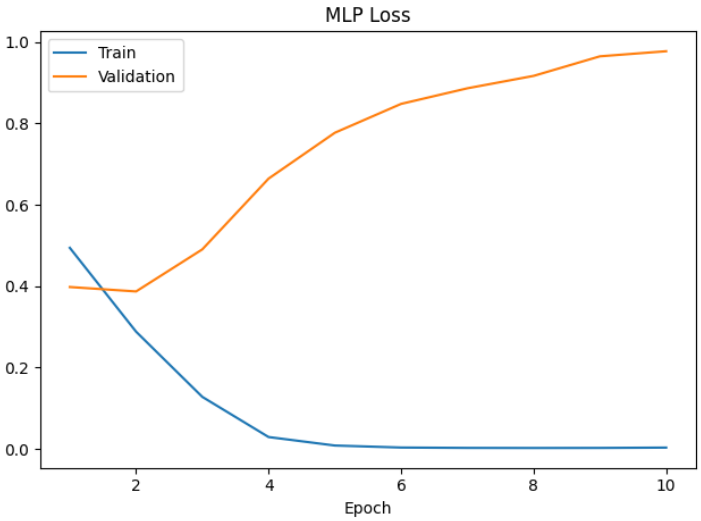
超参数如下：

超参数	数值
epoch	10
batch_size	128
learning_rate	2e-3
hidden_size	128
max_len	60

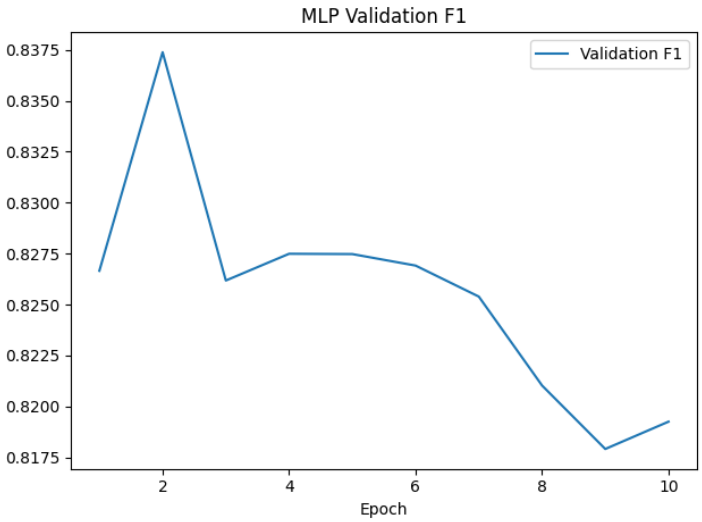
train、val accuracy:



train、val loss:



val F1:



在 test set 上的表现为：

指标	数值
Loss	0.9975
Accuracy	0.8049
F1	0.7931

值得注意的是训练到第 10 轮的时候, train set 的 accuracy 已经几乎完全接近 1 了。

7 参数效果对比

7.1 batch size 实验

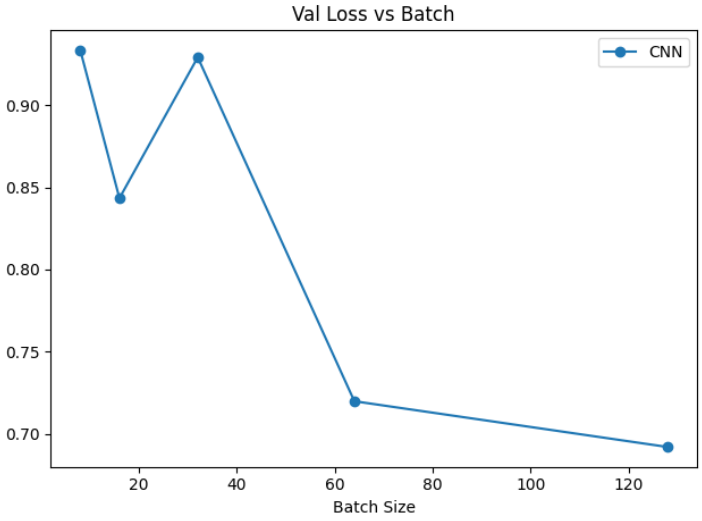


图 1: 不同 batch 的 loss 表现

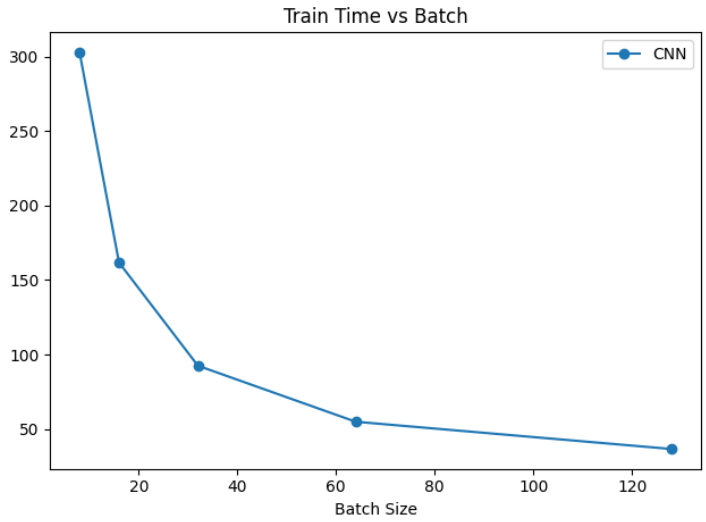


图 2: 不同 batch 的用时表现

当选取的 batch size 越大，意味着反向传播过程的进行次数越少，这也就对应着训练时长的降低。随着 batch_size 增大，模型更新更稳定、更接近“真实梯度方向”，训练过程变得平滑，这通常有助于更好的收敛，因此 valid loss 在一定范围内可能会减小。但这不是绝对的趋势——它受到多因素影响。CNN 本身模型参数量小 → 更依赖稳定训练 TextCNN 这种模型，参数少、结构浅，训练非常敏感。小 batch 容易抖动，训练过程不稳定，valid loss 高。大 batch 给它更稳定的梯度，反而容易学得好。