实验二 文本情感分类 实验报告

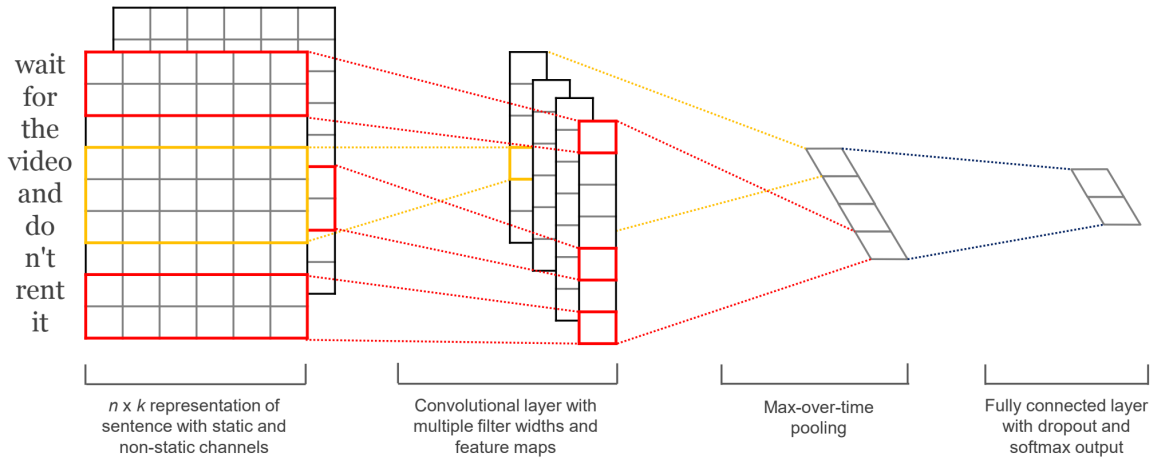
展子航 2022010745

实验报告内容要求

1. 模型的结构与流程分析
2. 实验结果、准确率、F-score
3. 比较不同参数的效果和原因分析
4. 比较三个模型的差异
5. 问题思考
    1. 试验训练什么时候停止最合适？简要陈述你的实现方式，并且分析固定迭代次数和调整验证集等方法的优缺点？
    2. 实验参数的初始化是怎么做的？不同的方法适合哪些地方？
    3. 如何避免陷入过拟合？
    4. 三个神经网络的优缺点？
6. 心得体会

模型结构与流程分析

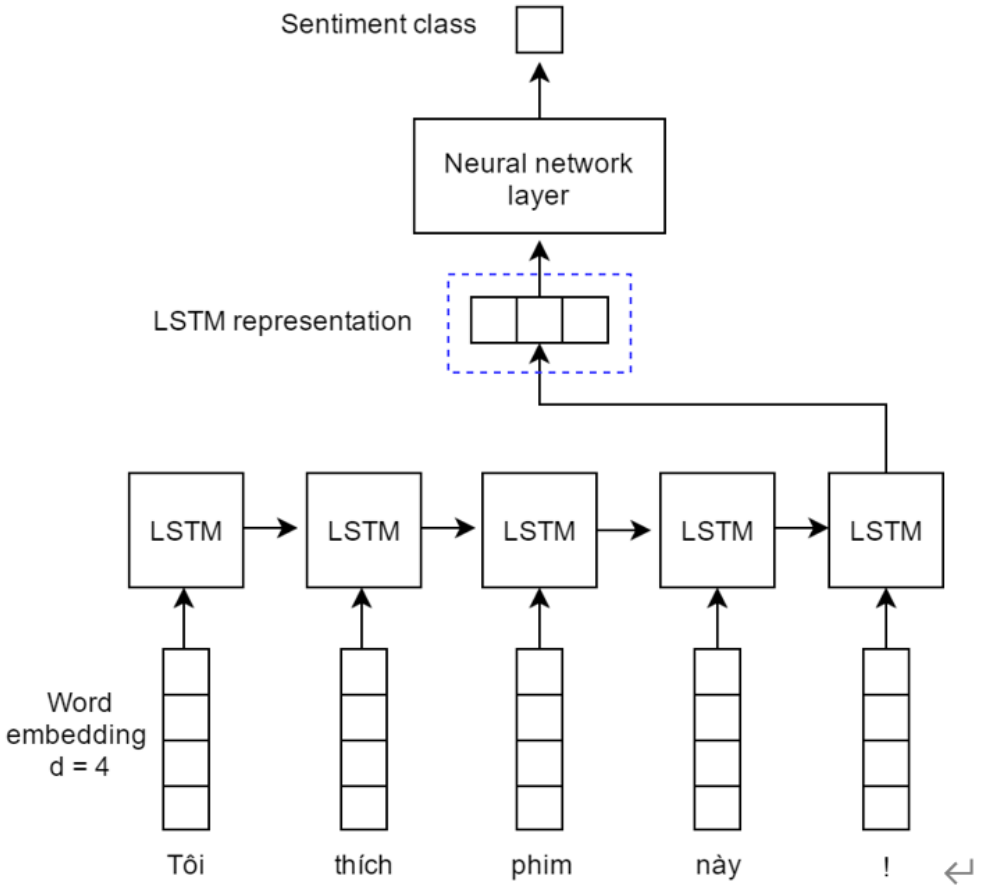
Text-CNN



- 嵌入层：单词-向量映射集。
- 前向传播：
    - 嵌入转换：将预处理过的分词 id 输入句转换为张量组，再增加一个维度将其从三维张量（batch_size, sequence_length, embedding_dim）转换为四维张量（batch_size, 1, sequence_length, embedding_dim）。
    - 卷积层：使用大小为：2，3，5，7 的各 20 个核对转换后的输入执行卷积计算。
    - 池化层：通过 `relu` 函数进行激活后进行池化。
    - Dropout层：隐藏部分训练数据。

- 线性输出层：拼接池化后结果得到张量，之后使用一层线性和 `softmax` 函数得到表示类别标签预测的向量。

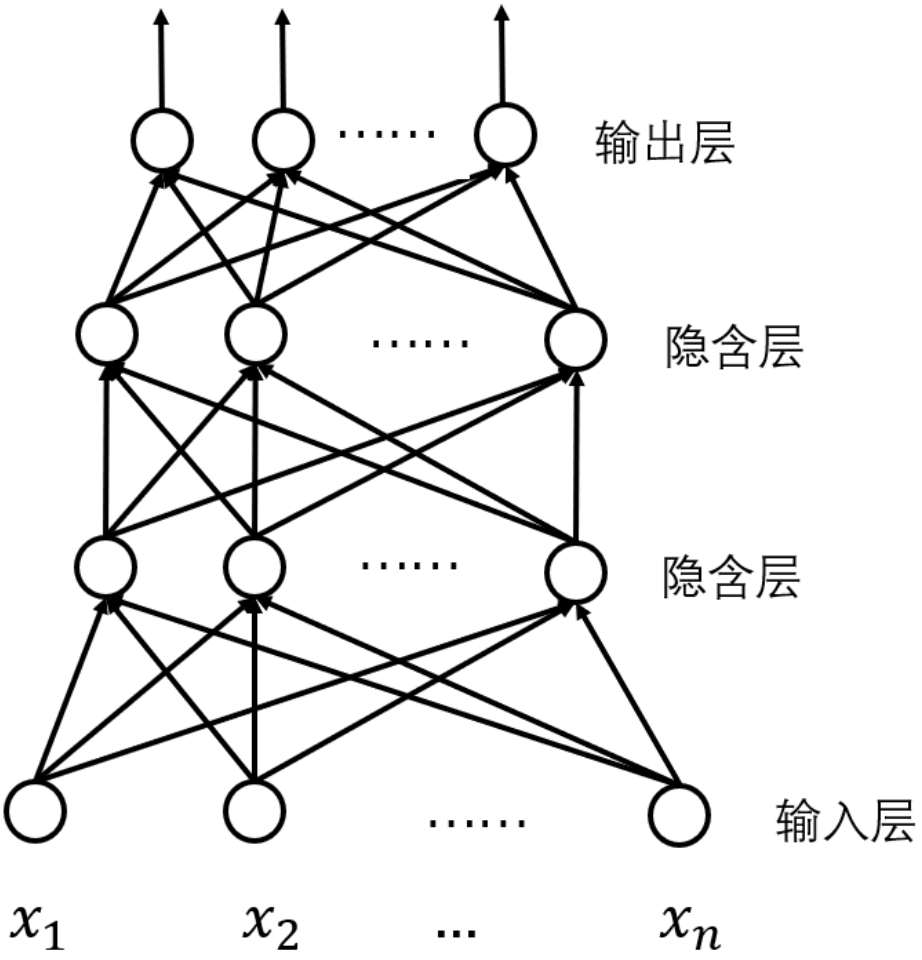
LSTM-RNN



- 嵌入层：同上不变。
- 前向传播：
  - 嵌入转换：将预处理过的分词 id 输入句转换为张量组，并将其由（sequence_length, batch_size, embedding_dim）转置为（batch_size, sequence_length, embedding_dim）以匹配输入。
  - LSTM：将张量组进行LSTM迭代。
- 线性层输出：取出最后一次迭代的隐藏层，进行类似于MLP的线性迭代（句长 -> 128输出 -> 64输出 -> 2输出）得到最终的类别标签预测的向量。

MLP

下图中结构严格意义上出现了两次：

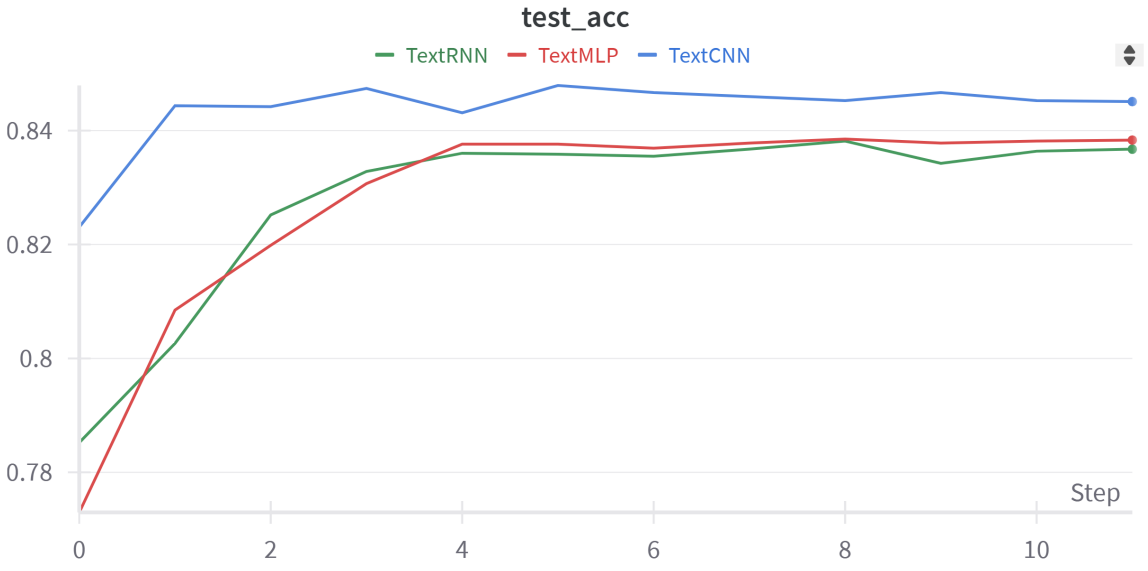


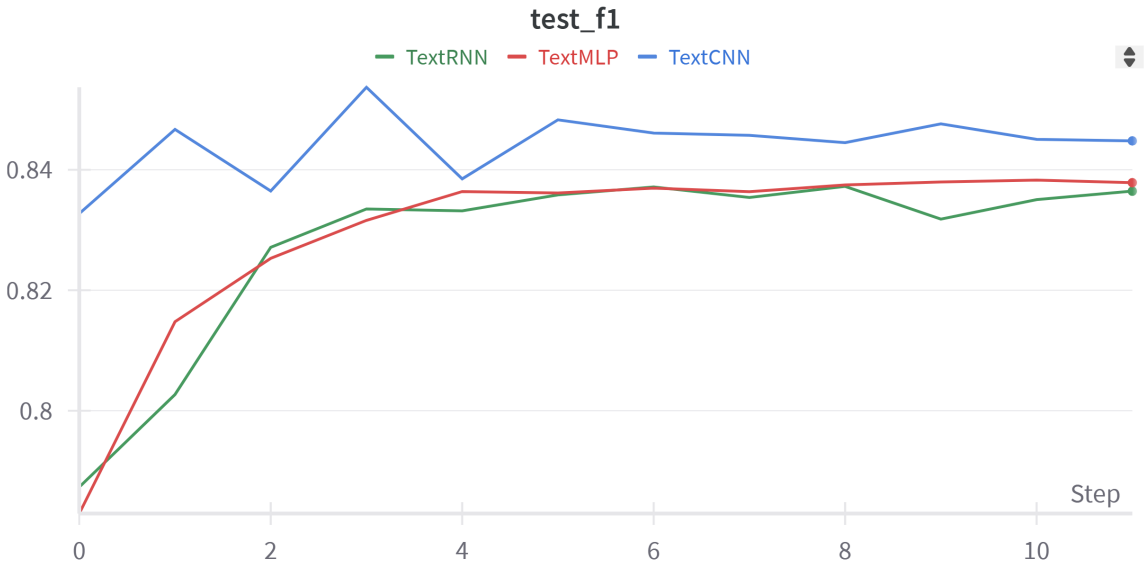
- 嵌入层：同上不变。
- 前向传播：
  - 嵌入转换：将预处理过的分词 id 输入句转换为张量组。
  - 线性层：进行第一层线性迭代（句长 -> 100），一层激活（ReLU），一层转置由（length, sentence, hidden）为（length, hidden, sentence），一层降维池化转为（length, hidden），第二层线性迭代得到标签预测的张量。
- 输出：直接输出最后的预测张量即可。

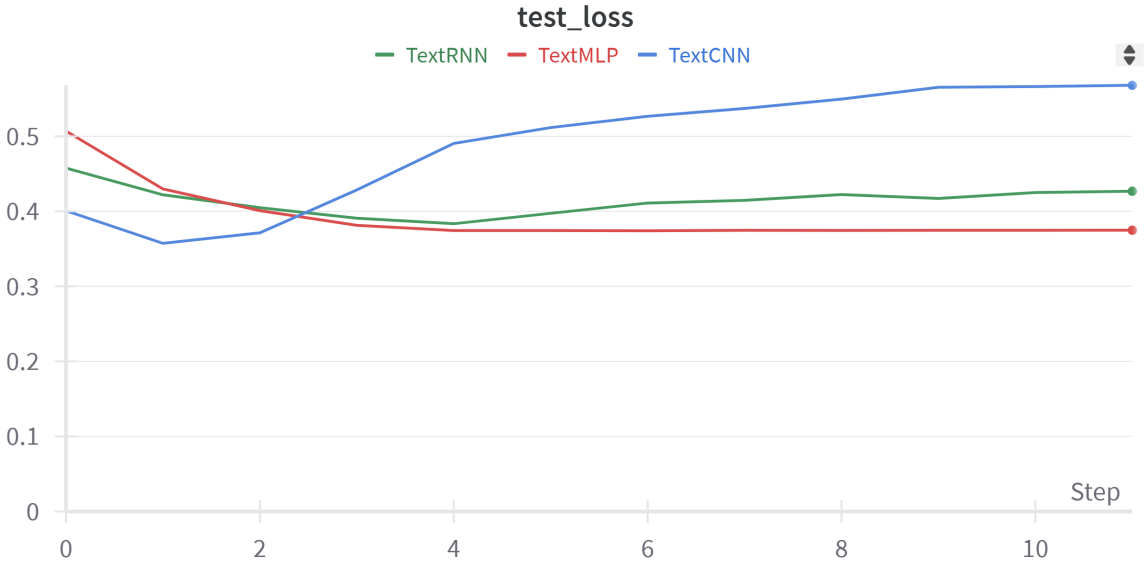
实验结果展示

如下九图，使用 `wandb` 绘制，节点为每个 epoch 测算一次，`f1` 的原型为 `sklearn.metrics.f1_score`（sklearn.metrics.f1_score — scikit-learn 1.4.2 documentation），`loss` 的原型为 `torch.nn.CrossEntropyLoss`（CrossEntropyLoss — PyTorch 2.3 documentation），`acc` 即为 `判定成功数/总判定数` 的基础算法：

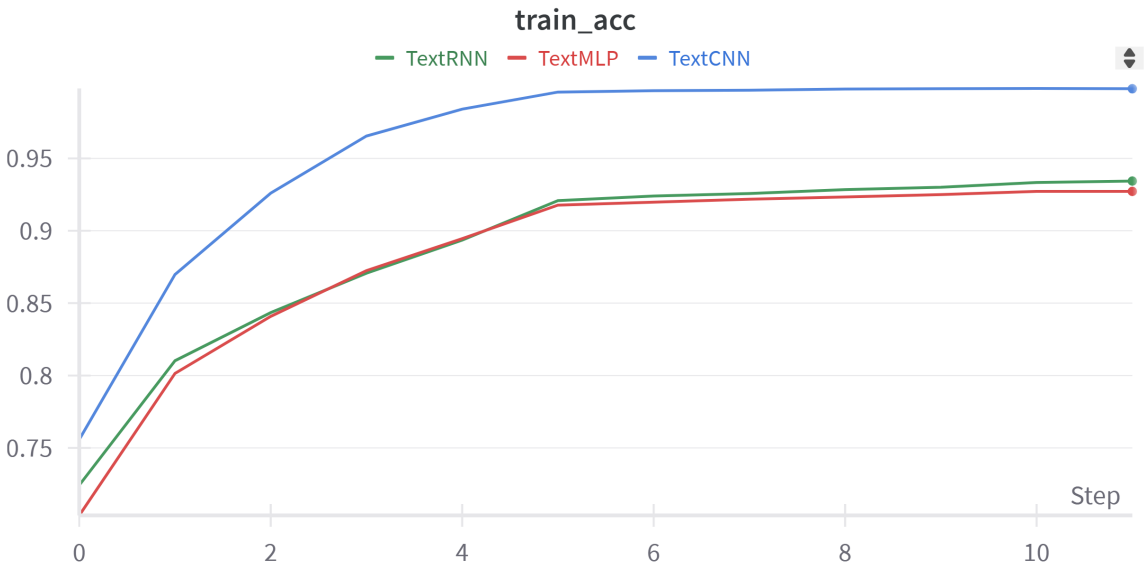
- test 部分：

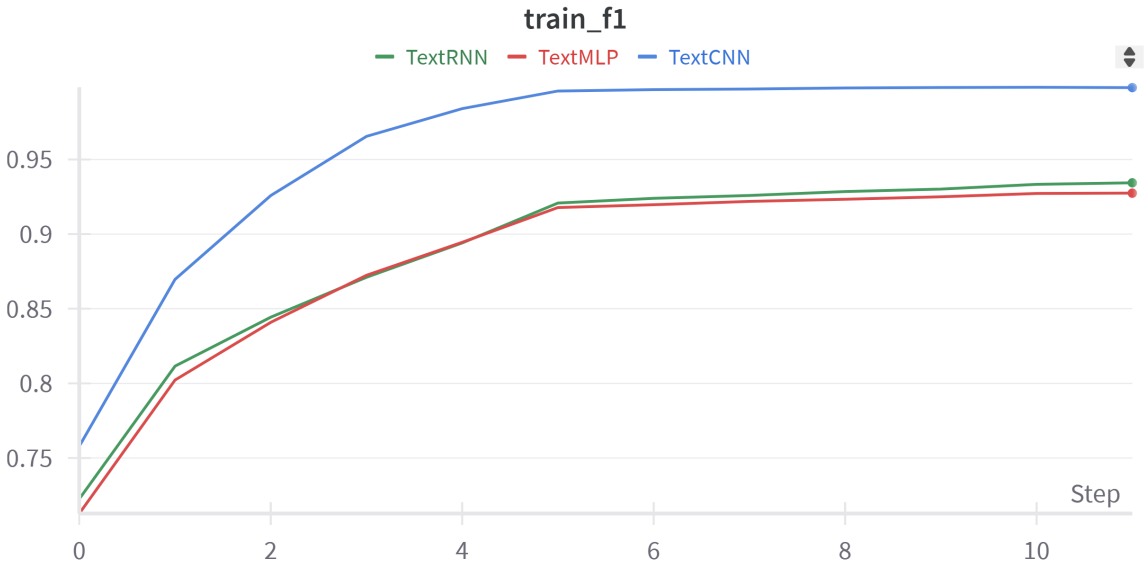

test_acc
TextRNN
TextMLP
TextCNN
0.84
0.82
0.8
0.78
Step
0
2
4
6
8
10


test_f1
TextRNN
TextMLP
TextCNN
0.84
0.82
0.8
Step
0
2
4
6
8
10

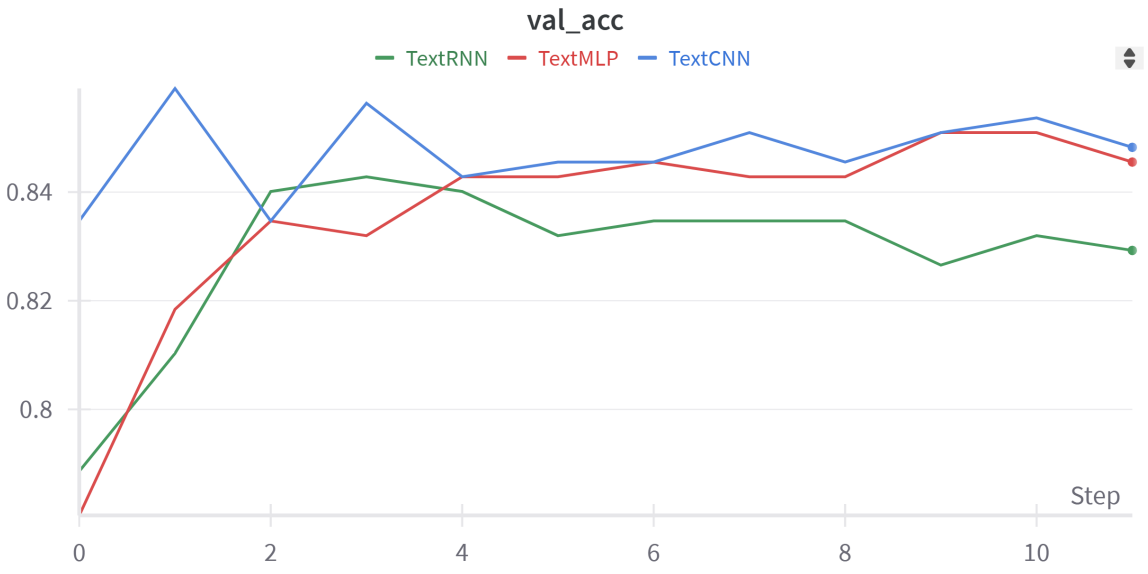

test_loss
TextRNN
TextMLP
TextCNN
0.5
0.4
0.3
0.2
0.1
0
Step
0
2
4
6
8
10

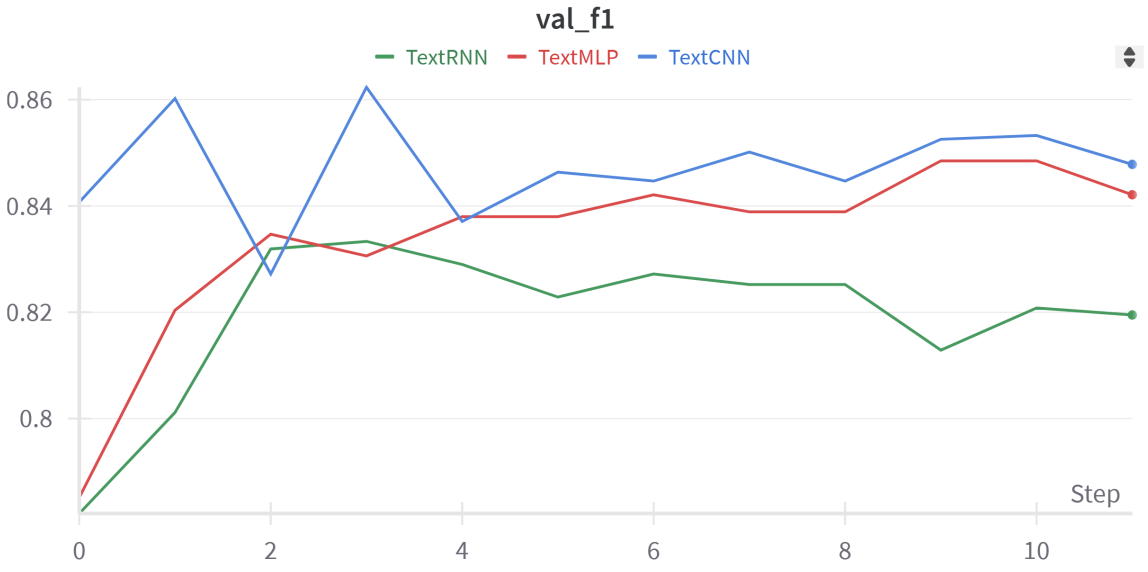
- train 部分：

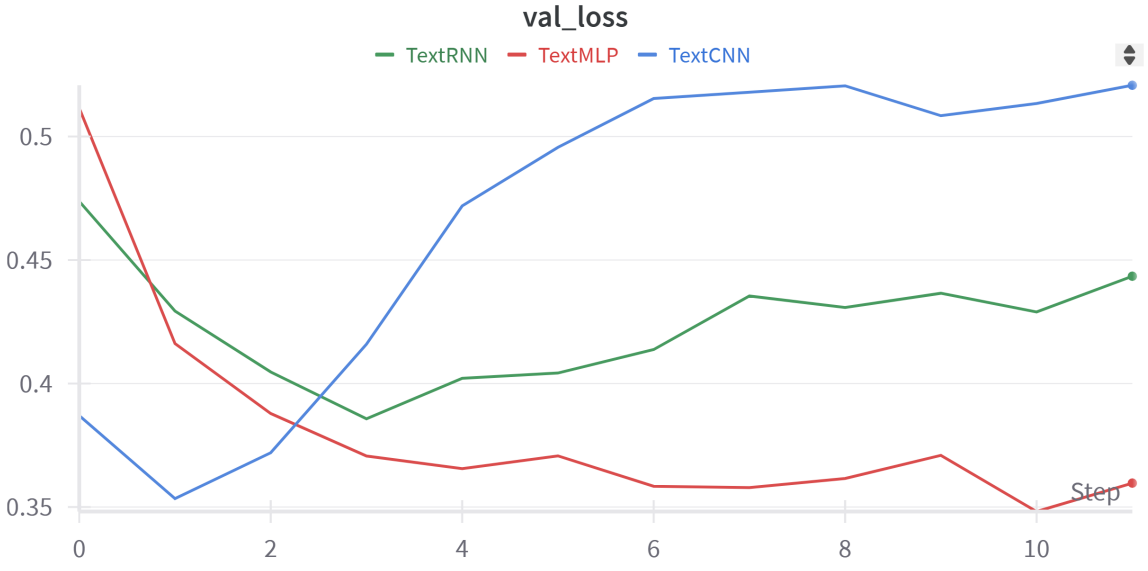




- validation 部分：







由于我们对于一个模型的最终性能分析，最主要看中的是 `test` 部分，因此特别整理这个部分图像情况如下：

Models	test_acc	test_loss	F-Score
CNN	最优 0.8479，图像正常	最优 0.3574，图像回升	最优 0.8537，中间浮动但整体正常
RNN	最优 0.8385，图像正常	最优 0.3835，图像回升但不明显	最优 0.8372，图像正常
MLP	最优 0.8382，图像正常	最优 0.3741，图像正常	最优 0.8383，图像正常

由图像与数据来看：

- 在所有的准确度层面上，`CNN` 都显然优于其他二者，`MLP` 正确率与 `RNN` 相当。
- 从损失层面上，CNN和RNN虽然从图像来看都有所过拟合，但是实际上在测试时准确率依旧很高，也即是过拟合现象不明显；MLP几乎没有过拟合现象。

参数调整与对比

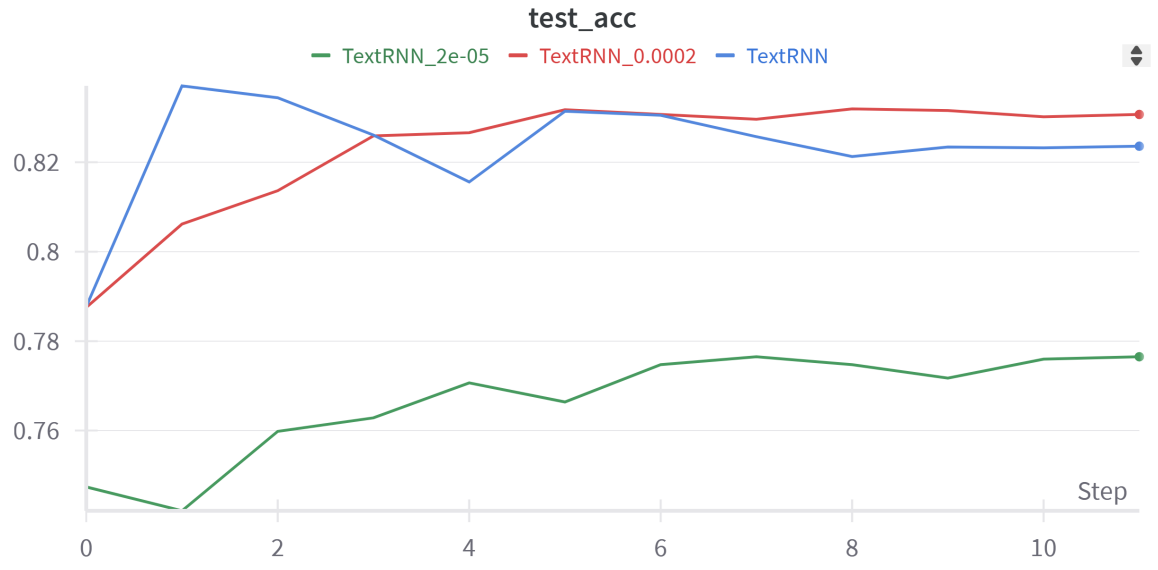
一言以蔽之，极尽玄学之本分。

学习率

初始我将三个模型的学习率都设为 `1e-3`，后来发现只有 `CNN` 在这个学习率下是可行的。

考虑到过高的学习率可能会导致梯度下降过快最终达到强过拟合，因此对于其他两个模型，我试图降低学习率。

对于RNN，我尝试了 3 种学习率，分别为 `2e-5, 2e-4, 1e-3`，之后确定了 `2e-4` 的学习率（`-5` 级别太蜗牛爬了，收敛速度太慢，估计一次需要训十倍以上的 `epochs`，于是放弃）：



由于实际上**我的MLP学习方式和RNN的结尾部分相似**，所以我直接使用了 `2e-4` 作为学习率，发现可行。

最想不到的一集：词汇库的囊括问题

- 一开始，将训练集、验证集和测试集出现的可被标准向量库识别的词全部加入词汇库中，结果**经常性过拟合。**
- 之后，根据 AI 模型给予的提示，将词汇库改为**只包括训练集**，解决了此问题。
- 这是因为模型应该能够处理在训练阶段未遇到的新词汇，这是其泛化能力的一部分。如果在验证集上重新构建词汇表，可能会引入模型在训练阶段未见过的词汇，这会降低模型的泛化能力导致过拟合。

问题思考

实验训练什么时候停止是最合适的？简要陈述你的实现方式，并试分析固定迭代次数与通过验证集调整等方法的优缺点。

我使用的是**手动测量 5 - 50 的 epochs 测试，之后确定 12 为固定迭代次数**，并且进行 checkpoints 留存。

对于固定迭代次数：

- 优点
  - 简单: 实施起来非常简单，只需要设置一个预定的迭代次数。
  - 可复现性: 确保每次实验的迭代次数相同，有助于结果的复现。
- 缺点
  - 效率低下: 可能在模型已经收敛后继续训练，浪费计算资源。否则就需要大量时间去确定不使模型收敛的时间，同样是对时间的浪费。
  - 不灵活: 不同的数据集和网络结构可能需要不同的训练时间才能达到最优性能。
  - 空间占用：如果和我一样选择留存 checkpoints ，会占用一定的存储空间。

对于通过验证集调整：

- 优点
  - 动态决策: 根据验证集上的性能动态决定是否停止训练。
  - 避免过拟合: 如果验证集性能不再提升，可能是模型开始记忆训练数据，过拟合发生，这时可以及时停止止损。
- 缺点
  - 计算成本大: 需要频繁评估验证集，可能会增加计算成本。
  - 决策本身需要经验: 需要根据经验确定基于验证集性能达到何种条件时，做出停止决策。

对于使用学习率衰减：

- 优点
  - 促进精细收敛: 随着训练的进行逐渐减小学习率，有助于模型细致地逼近最优解。
  - 自动停止: 学习率降低到某个阈值以下时，可以认为模型已经足够接近最优，可以停止训练，不需要额外的预估调整。
- 缺点
  - 决策本身需要经验: 需要精心设计学习率衰减的策略和衰减率。
  - 可能欠拟合: 如果学习率降得太快，可能导致模型未能充分学习数据特征。

实验参数的初始化是怎么做的？不同的方法适合哪些地方？

对于我的模型来说（`nn.Conv2d`，`nn.Linear`，`nn.LSTM`），都使用了默认的零均值初始化。

不同的初始化方法一般适用于不同的激活函数和神经网络算法模型下。

- 对于零均值初始化，当激活函数是 ReLU 或其变种时，零均值初始化可以防止训练初期梯度消失的问题。
- 对于正交初始化，对于某些类型的循环网络，如LSTM或用于处理序列数据的网络，正交初始化可以保证梯度不会爆炸或消失，这里我没有使用，而是通过修改学习率的方式控制了梯度范围。
- 对于高斯分布初始化，允许用户自定义高斯分布的均值和标准差，这使得该方法能够根据不同的网络结构和数据分布自适应地进行参数初始化，灵活性较强。

过拟合是深度学习常见的问题，有什么方法可以方式训练过程陷入过拟合？

我们需要针对原因找答案。

1. 训练数据量过少，导致模型过拟合——增大训练数据量。
2. 训练参数量过大，模型过于复杂导致提炼出了一些不属于训练数据的数学特征，导致过拟合——直接简化模型；或者采取 dropout 减少训练神经元权重数目（我在三个网络中都使用了这个操作）。
3. 训练时间过长，导致模型对训练数据过度学习——限制学习率、`batch_size`、`epochs`，避免训练时间过长；或者使用学习率衰减等方式达成早停。

试分析CNN，RNN，全连接神经网络（MLP）三者的优缺点。

卷积神经网络（CNN）

优点：

1. 局部感知能力：CNN通过局部感受野捕捉局部特征，适用于文本中局部模式（如短语）对情感的影响。
2. 参数共享：卷积层的权重在整个输入文本上共享，减少了模型的参数数量，有助于避免过拟合。
3. 计算效率：在固定大小的输入时（例如本次所有的句子都是填充为定长的），CNN通常更高效。

缺点：

1. 全局依赖性：CNN可能不如RNN擅长捕捉文本中的长距离依赖关系。
2. 固定窗口大小：传统的卷积操作通常有一个固定的感受野，可能不适合捕捉不同长度的上下文信息，如果使用的卷积核过多反而也会导致特征混乱。

循环神经网络（RNN）

优点：

1. 捕捉长距离依赖：RNN通过其循环结构能够捕捉文本中的长距离依赖关系，这对于理解情感上下文很重要。
2. 序列数据的自然处理能力：RNN设计用来处理序列数据，能够处理任意长度的文本。

缺点：

1. 梯度消失 / 爆炸：RNN在处理长序列时可能会遇到梯度消失或梯度爆炸的问题。
2. 计算效率：RNN在处理长序列时可能比CNN慢，因为不能像CNN那样高效地并行化。
3. 过拟合风险：由于RNN的参数较多，且对长序列的依赖性，可能会有更高的过拟合风险。

全连接神经网络（MLP）

优点：

1. 简单性：MLP是最简单的神经网络结构，易于理解和实现。
2. 快速训练：由于缺少循环结构，MLP通常可以快速训练。

缺点：

1. 局部特征捕捉能力弱：MLP不具备捕捉局部特征的能力，通常需要将文本转换为固定长度的向量，可能会丢失一些上下文信息。
2. 序列数据处理能力有限：MLP不擅长处理序列数据，需要额外的技术（如词嵌入）来处理文本。
3. 缺乏长距离依赖捕捉能力：MLP通常不能捕捉文本中的长距离依赖关系。

心得感受

如果我生活在魔法大陆，一定是炼金术士里面最拉胯的。

调参数玄学性实在拉满，由于我没有服务器资源，只能在自己的电脑上跑，每次对一个模型的一个参数的改动都要浪费生命的五分钟，等待调试的过程实在是过于煎熬。不过仔细想想，科研不都是这样吗，也就逐渐释怀了一些。

偶然发现的 wandb 可视化网站十分简洁精炼，以后做神经网络我还会继续使用它。

上一次在大一小学期，我实际上做的是数据收集和预处理阶段的工作，和这次组合起来刚好是一个独立完成神经网络训练的工作，还是很有成就感的。上一次我深刻体会到了数据的重要性。清洗、预处理、特征工程，每一步都至关重要，它们直接影响着模型的最终性能。选择合适的网络架构，调整超参数，这些决策需要基于对问题本质的理解以及实验结果的反馈。这一次我又体会到了模型本身的重要性，二者就正如一切的源头——向量乘法一样，两个向量必须都足够有效才能得出对的结果。