情感分类实验报告

刘雅迪
计 26

实验代码链接：https://cloud.tsinghua.edu.cn/d/7c24bad933824fc1b954/

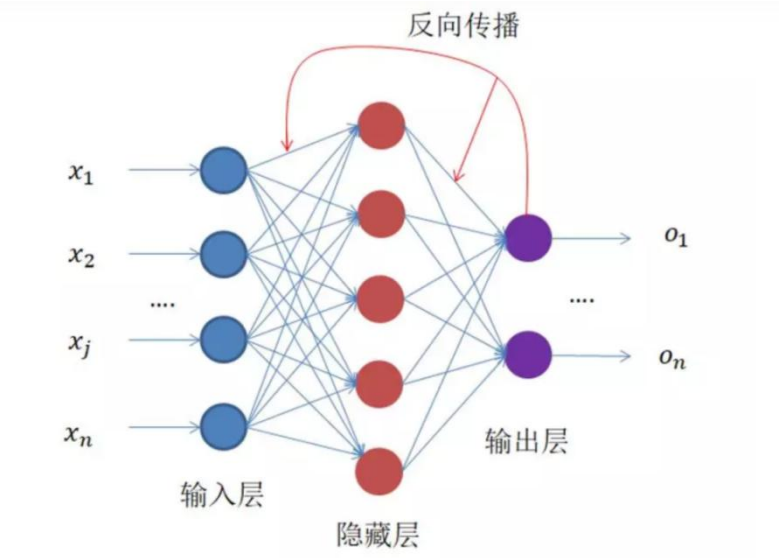
一、数据预处理

将 train.txt 和 validation.txt 中的词分配 id，将数据集从文字转换为固定长度的 id 序列表示，固定长度 50 为词向量的长度，对于句子长度不足的进行零向量补全，对于超出的进行裁剪。将处理得到的文件以.npy 的格式保存，在训练模型的时候可以直接调用。

二、模型结构与流程分析

（一）全连接神经网络（MLP）

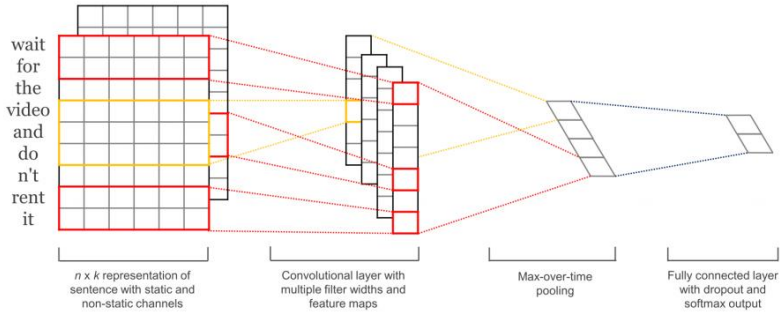
使用 MLP 模型为 baseline。



嵌入层：使用向量表示单词，向量长度一定，用于模型的输入
线性层 1：使用全连接的线性网络，将输入的向量拼接成指定大小的张量
池化层：选择激活函数为 ReLU 函数，并通过参数为 0.5 的 Dropout 层，再进行最大池化
线性层 2：再通过一层全连接的线性网络，将输出映射到二维数组，得到类别预测的向量

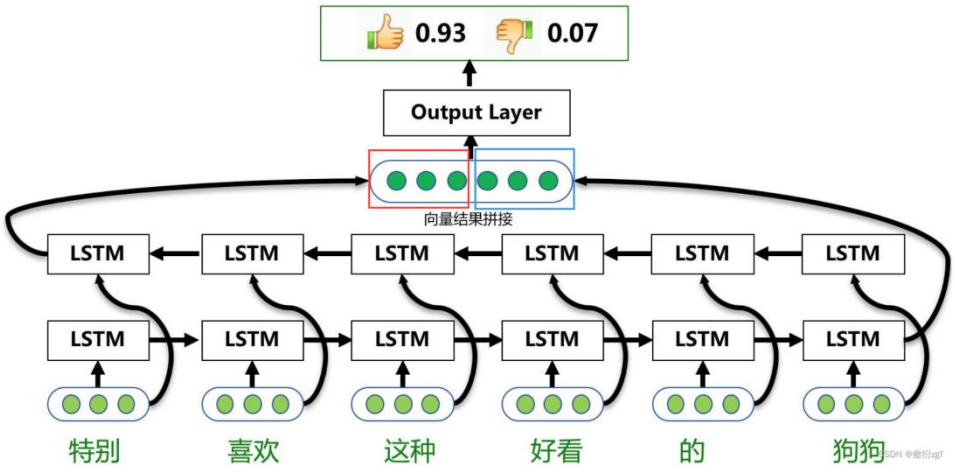
（二）卷积神经网络 CNN

使用 CNN 中的 TextCNN 模型，该模型更适合文本分类问题。



嵌入层：使用向量表示单词，向量长度一定，用于模型的输入
卷积层：对嵌入层得到的数据进行多通道一维卷积，长度为对齐后的句子长度，通道数为对应的词向量数，分别用宽度为 3、4、5 的卷积核做三次卷积，然后直接进行拼接，得到一维输出特征
池化层：选择激活函数为 ReLU 函数，并且使用 max_pooling
线性层：将经过池化层后的结果拼接在一起，并通过 Dropout 层防止过拟合，随后再通过一层全连接的线性网络，将输出映射到二维数组，得到类别预测的向量

（三）循环神经网络 RNN
使用 RNN 中的双向 LSTM 模型来进行文本分类。



嵌入层：使用向量表示单词，向量长度一定，用于模型的输入
双向 LSTM 层：将嵌入层得到的数据通过双向 LSTM 层，每个 LSTM 单元同时向左右隐藏自己的隐含状态，使用最左和最右侧的状态作为本层的输出
线性层：将上一层的两个状态直接拼接，并通过一层 Dropout 层防止过拟合，随后再通过一层全连接的线性网络，将输出映射到二维数组，得到类别预测的向量

三、实验结果与模型效果分析

通过 wandb 来辅助结果的可视化，且在验证集上运行时保留最佳模型。
下表为三种模型在初始参数下的训练时间，以及训练完成后最佳模型在测试集上的表现：

不同模型的训练效果

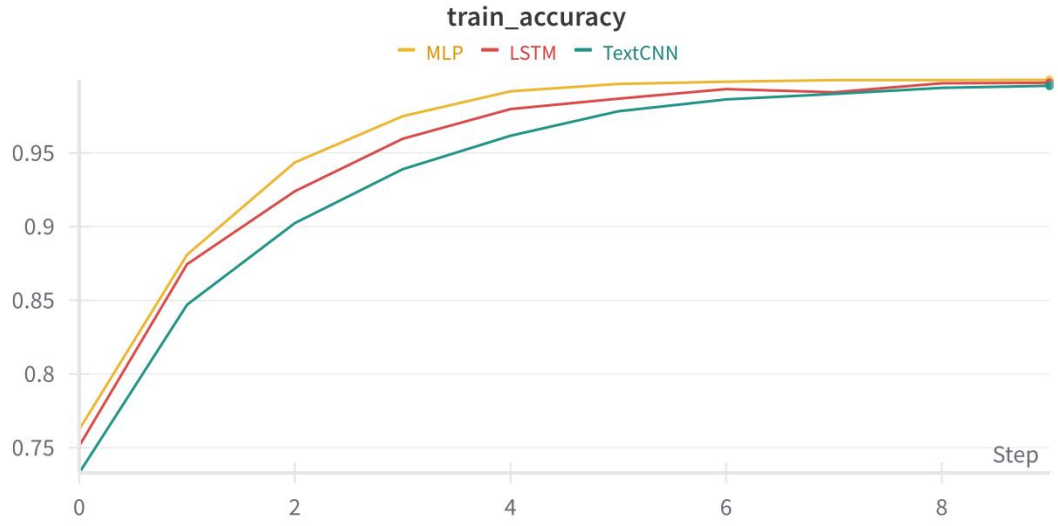
模型	训练用时	ACCURACY	F1-SCORE
MLP	204.63s	80.76%	80.76%

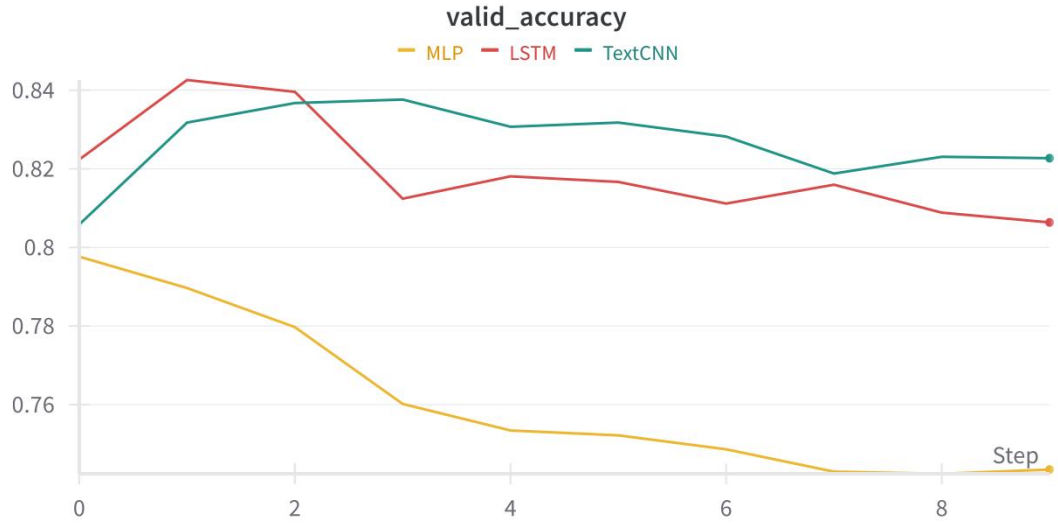
TEXTCNN	216.31s	84.28%	83.52%
LSTM	2231.24s	84.55%	84.03%

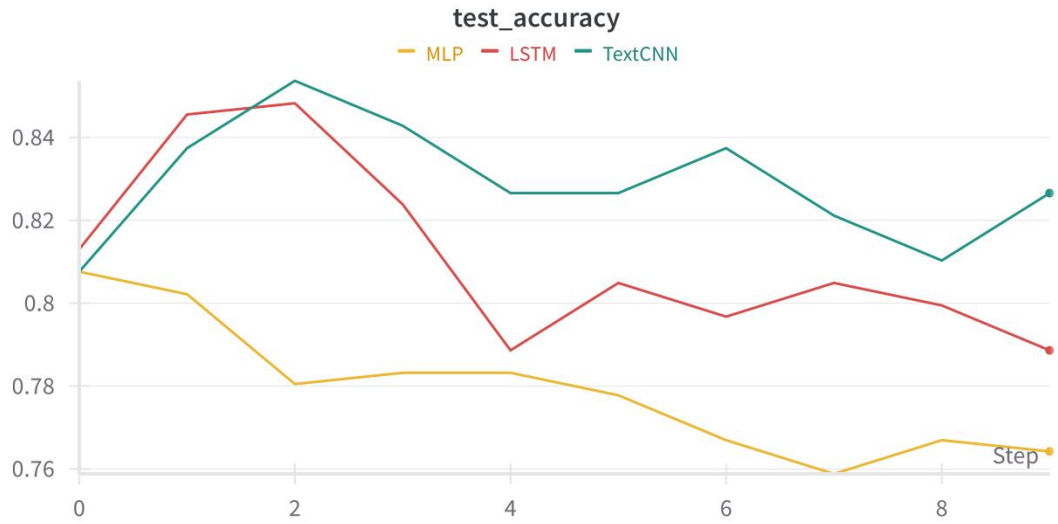
从上表可以看出，从 MLP 到 TextCNN 再到 LSTM，模型的训练时间逐渐变长，尤其是 LSTM 模型，在训练的时候可以明显地感觉到时间多用了好几倍。究其原因可能是 LSTM 的结构比较复杂，而 MLP 是最简单的结构。

此外，由于全连接网络近乎是线性分类，准确率和 F-score 明显比其它两类模型低，双向 LSTM 的效果略好于 TextCNN。因为双向的 LSTM 模型对于一个词，不仅会根据当前词和前一个词预测下一个词，还会根据当前词和后一个词预测下一个词，这种双向网络结构和循环模式使得双向 LSTM 能够处理更复杂的语言模式，也能更好低反应上下文语境的特点；而尽管 TextCNN 能通过卷积核反应上下文词与词之间的联系，但是在本次训练中由于卷积核的数量有限，故效果比双向 LSTM 略差一点。

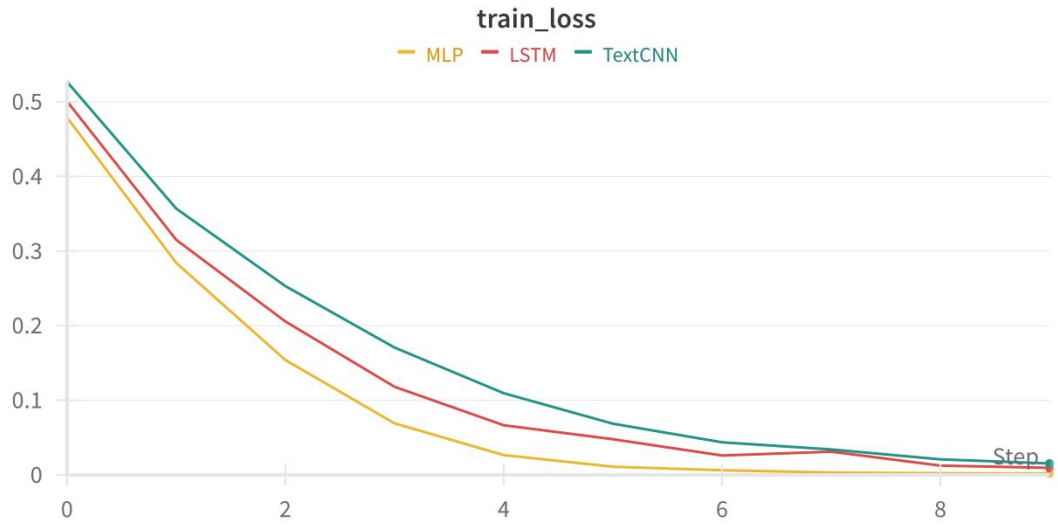
（一）Accuracy

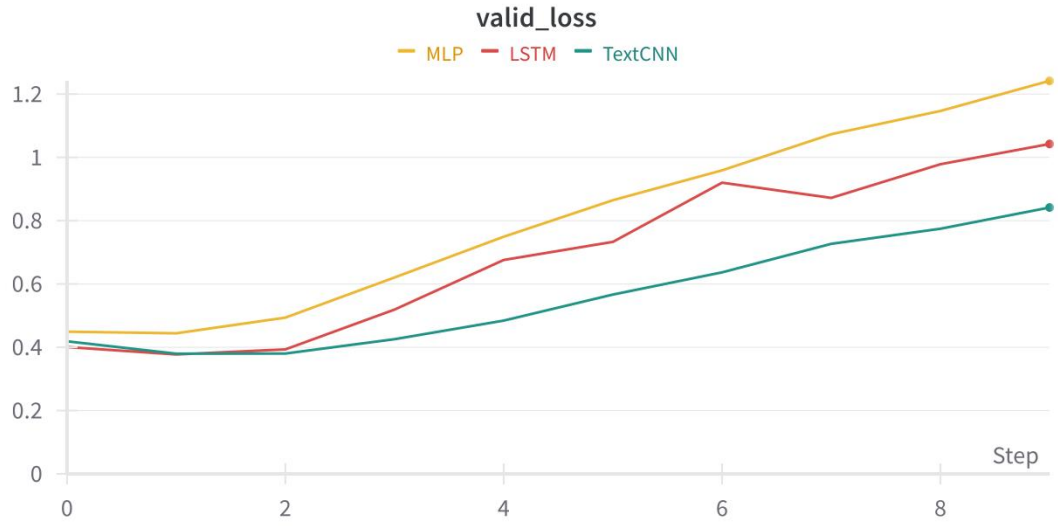


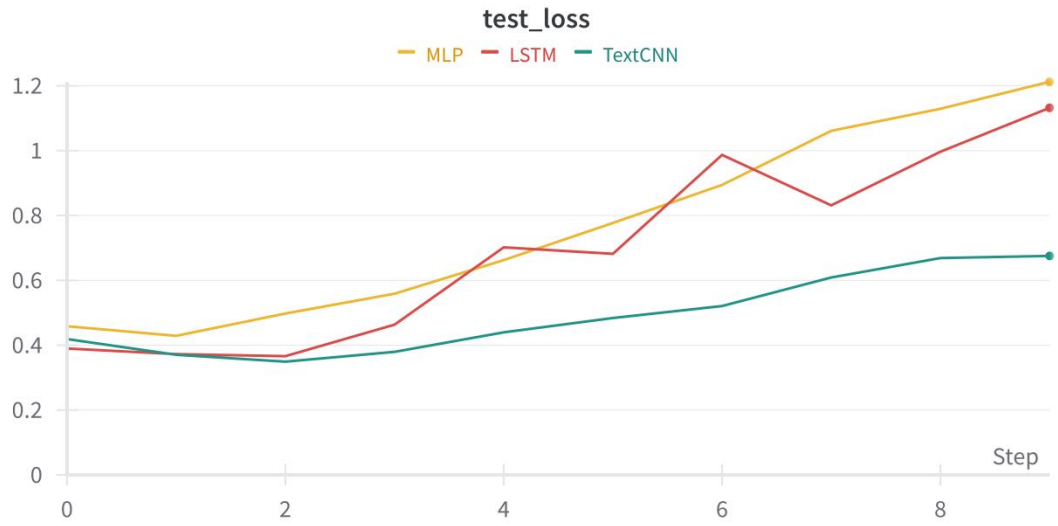



test_accuracy
MLP
LSTM
TextCNN
0.84
0.82
0.8
0.78
0.76
Step
0
2
4
6
8

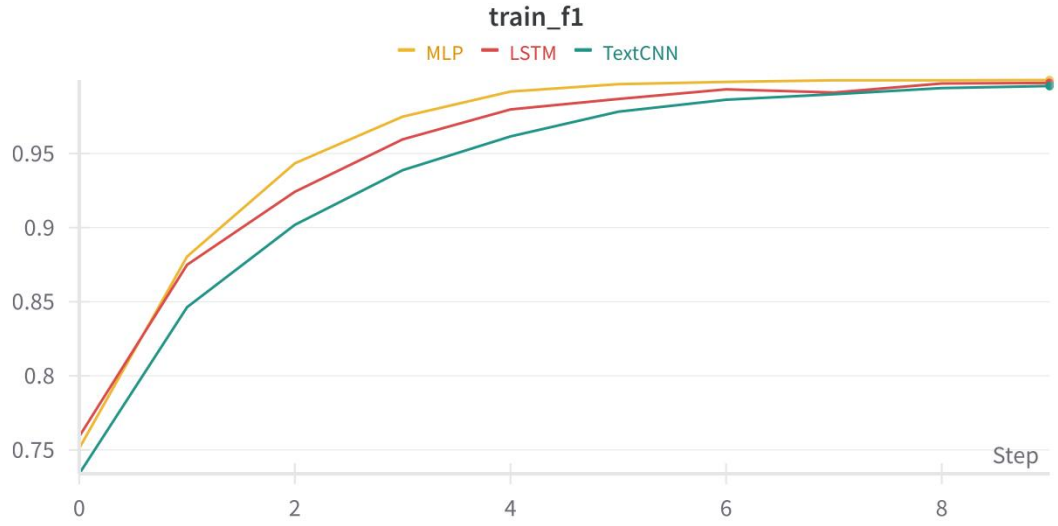
（二）Loss

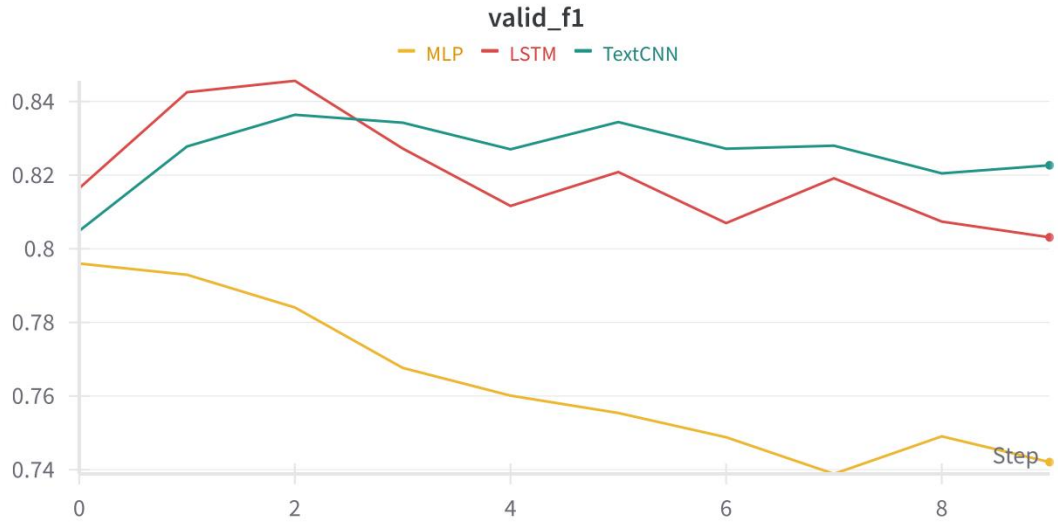

train_loss
MLP
LSTM
TextCNN
0.5
0.4
0.3
0.2
0.1
0
Step
0
2
4
6
8

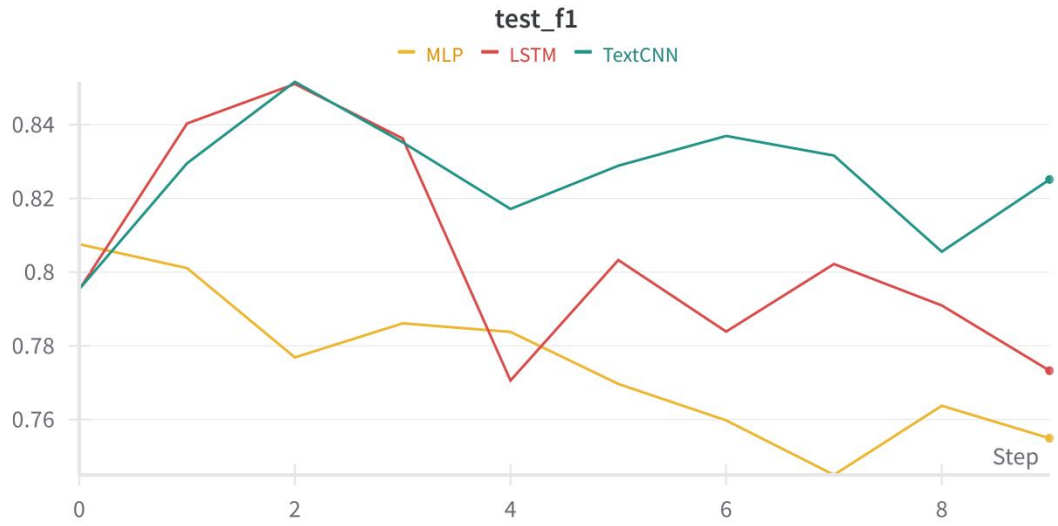

valid_loss
MLP
LSTM
TextCNN
1.2
1
0.8
0.6
0.4
0.2
0
Step
0
2
4
6
8


test_loss
MLP
LSTM
TextCNN
1.2
1
0.8
0.6
0.4
0.2
0
Step
0
2
4
6
8

（三）F1-score


train_f1
MLP
LSTM
TextCNN
0.95
0.9
0.85
0.8
0.75
Step
0
2
4
6
8





从上面的图和评价指标的含义可以观察出：

MLP：在训练集上损失值的下降收敛速度最快，但是准确率和 f 值最低，可能与模型的简单结构有关。

TextCNN（CNN）：在训练集上损失值的下降收敛速度最慢，准确率和 f 值均较高，且值相对比较稳定。

LSTM（RNN）：在训练集上损失值的下降收敛速度适中，但是准确率和 f 值的波动较大。

此外，TextCNN 和 LSTM 模型的准确率曲线走势基本相同，都是先上升一段时间，然后随着训练轮次的增加而逐渐出现过拟合现象。而 MLP 的准确率曲线则是一直下降，可能因为模型结构过于简单，所以最容易出现过拟合现象。

四、参数调整与比较

1. 调整模型训练周期 epoch

在本次实验中我分别尝试了 epoch 为 10、20、30、50，下面用表格的形式列出结果：

epoch	模型	训练时间	accuracy	F-score
10	MLP	204.63s	80.76%	80.76%
	TextCNN	216.31s	84.28%	83.52%

	LSTM	2231.24s	84.55%	84.03%
20	MLP	557.31s	80.49%	79.07%
	TextCNN	630.44s	85.09%	84.33%
	LSTM	8644.91s	82.93%	82.15%
30	MLP	1238.37s	81.03%	80.77%
	TextCNN	1332.29s	85.37%	85.33%
	LSTM	12711.69s	80.76%	78.68%
50	MLP	1975.94s	81.84%	81.02%
	TextCNN	2108.21s	83.74%	82.95%
	LSTM	12090.24s	83.74%	83.05%

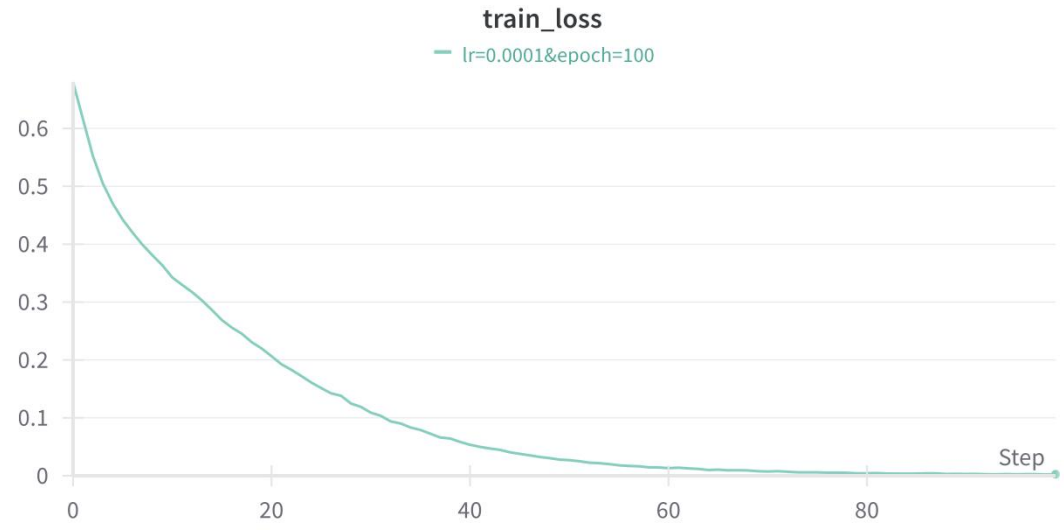
综合训练时间和评估指标，选择 epoch 为 10 是一个合适的值，此时模型在训练集上的损失值收敛，且模型的准确率和 f 值均较高。

2. 调整学习率 learning rate

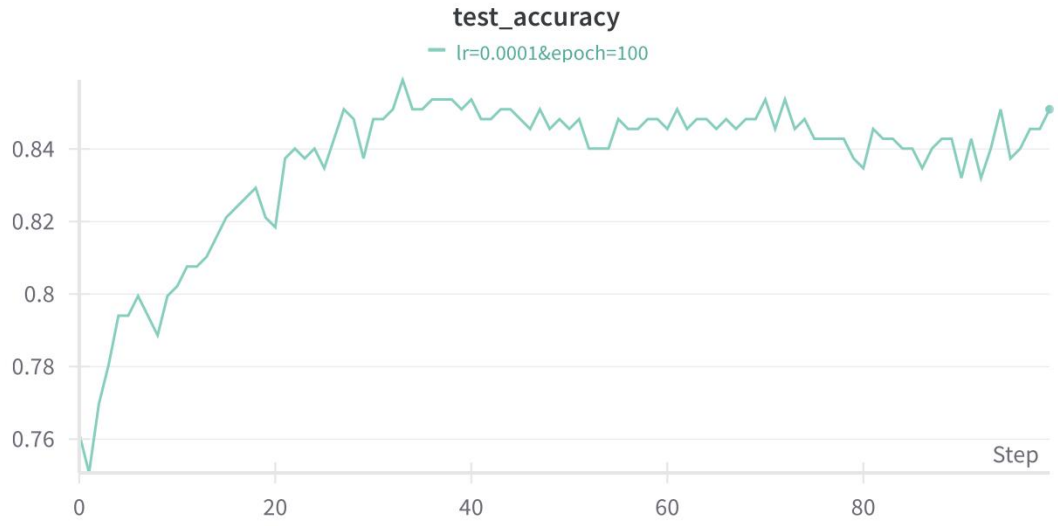
由于 LSTM 的训练时间过长，而我的电脑不支持 cuda 训练，故以下对参数的尝试只针对效果较好的 TextCNN 模型。

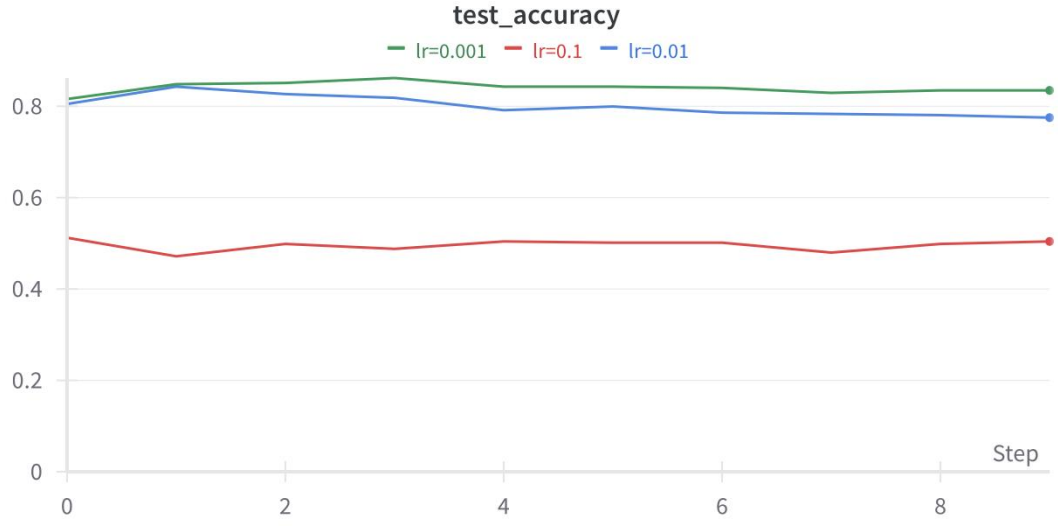
此外，当 learning rate 等于 0.0001 时，为了避免达到固定迭代次数后模型在训练集上的损失值没有收敛，将 epoch 设为 100。由于 epoch 有所不同，故在此不列出训练时间。

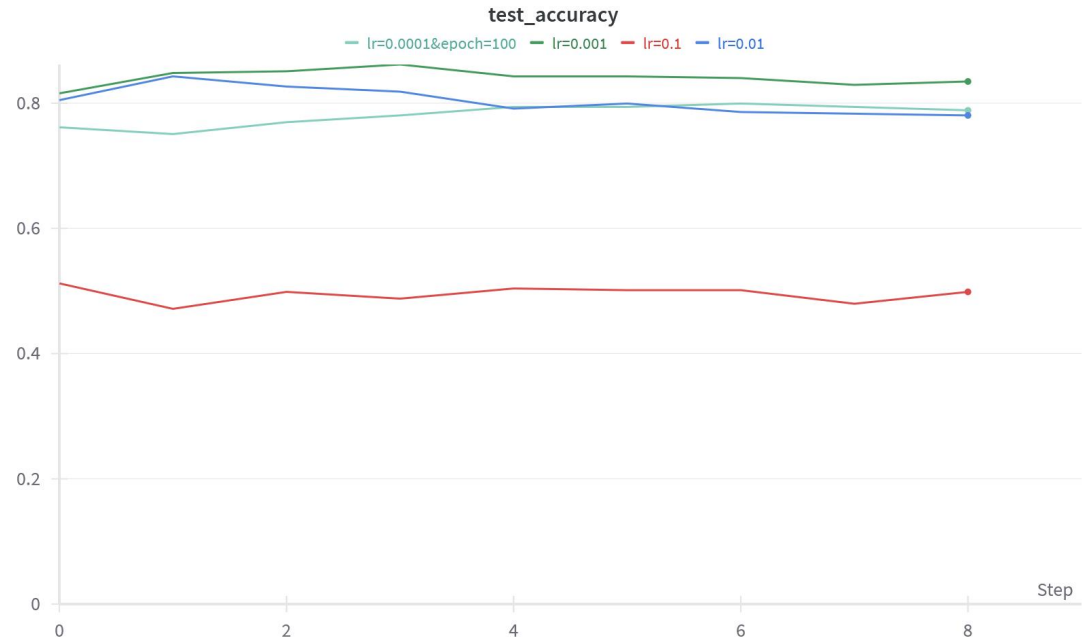
learning rate	accuracy	F-score
0.1	50.41%	11.59%
0.01	80.49%	79.55%
0.001	84.28%	83.52%
0.0001	83.74%	83.24%




train_loss
lr=0.001
lr=0.1
lr=0.01
30
25
20
15
10
5
0
Step
0
2
4
6
8


test_accuracy
lr=0.0001&epoch=100
0.84
0.82
0.8
0.78
0.76
Step
0
20
40
60
80


test_accuracy
lr=0.001
lr=0.1
lr=0.01
0.8
0.6
0.4
0.2
0
Step
0
2
4
6
8

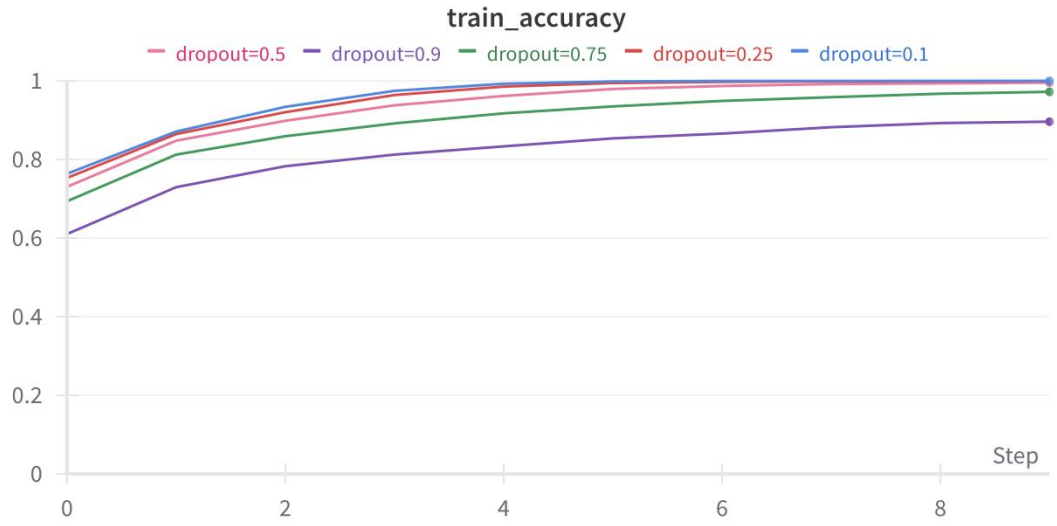


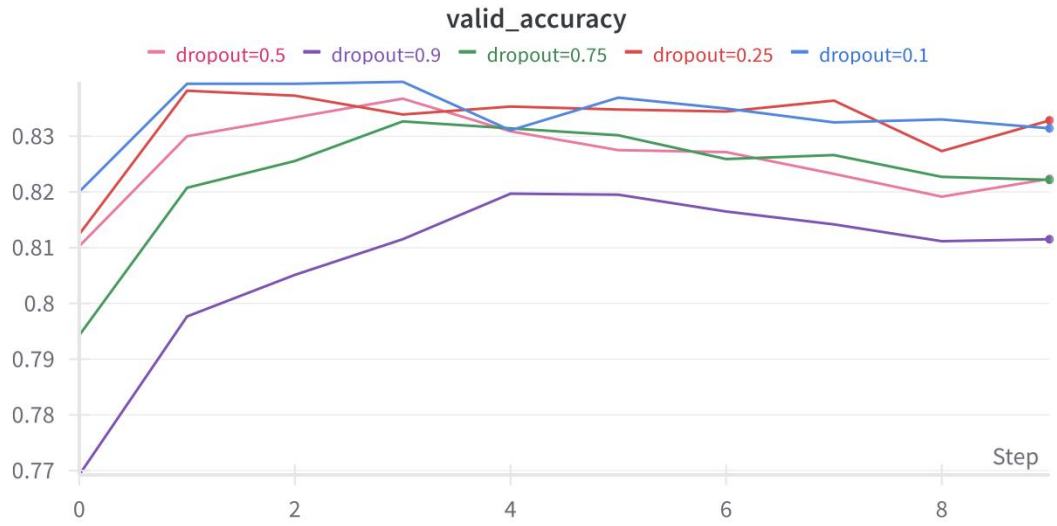
由上图和表可以看出，当 lr 较大时（如 lr=0.1），由于模型每次需要调整参数的幅度过大，导致模型的准确率和 f1 值均较低，跳跃幅度过大也导致模型较难收敛，模型效果一般。当 lr 较小时（如 lr=0.0001），模型在训练集上的损失值较难收敛，训练时间较长，如本次实验中当 epoch 大于 80 时模型才有收敛的趋势，但是由于模型每次参数调整的幅度较小，找到更好的模型对应的参数较为容易，在测试集上的准确率和 f 值也相对较高。

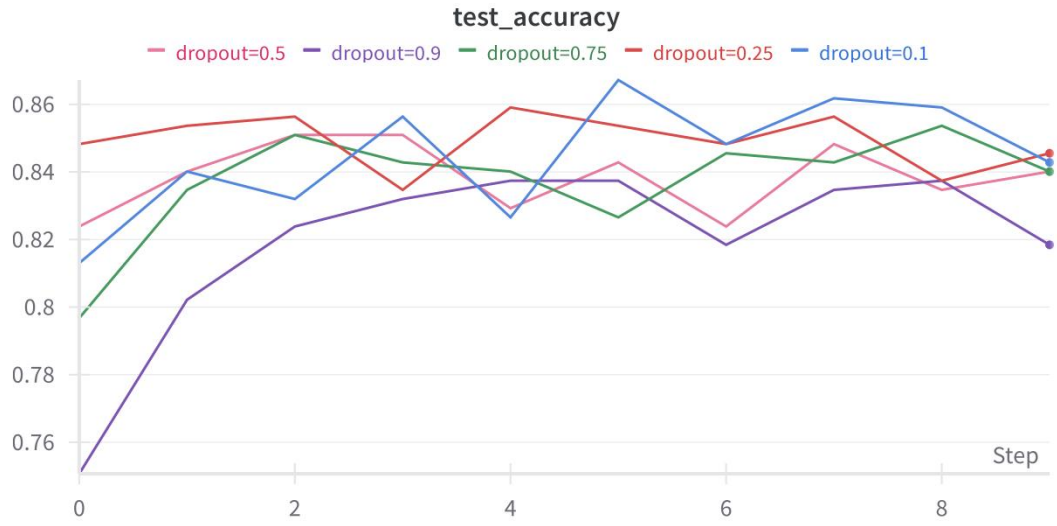
由此可见，默认学习率 0.001 是一个很合适的值，模型收敛速度较快，并且模型效果也很好。

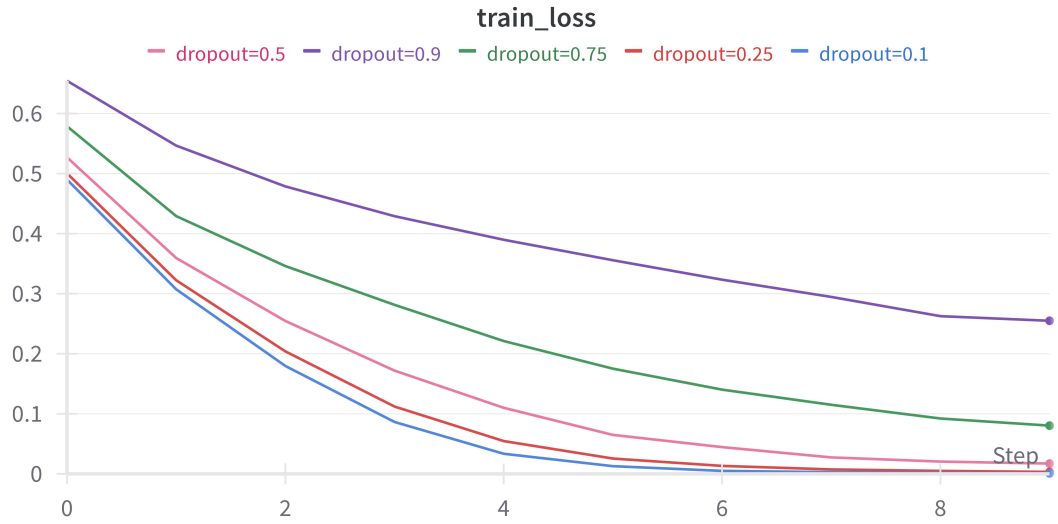
3. 调整 Dropout 参数

Dropout	训练时间	accuracy	F-score
0.1	328.04s	85.64%	85.40%
0.25	379.38s	84.28%	83.89%
0.5	286.98s	85.09%	85.09%
0.75	374.01s	84.28%	83.89%
0.9	353.84s	85.09%	85.09%


train_accuracy
dropout=0.5
dropout=0.9
dropout=0.75
dropout=0.25
dropout=0.1
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
Step


valid_accuracy
dropout=0.5
dropout=0.9
dropout=0.75
dropout=0.25
dropout=0.1
0.83
0.82
0.81
0.8
0.79
0.78
0.77
0
2
4
6
8
Step


test_accuracy
dropout=0.5
dropout=0.9
dropout=0.75
dropout=0.25
dropout=0.1
0.86
0.84
0.82
0.8
0.78
0.76
0
2
4
6
8
Step



由上图可以看出，当 dropout 值较大时，模型较难收敛，可以认为当 dropout 为 0.75 和 0.9 时，在现有的 epoch 为 10 的条件下模型训练的损失值还没有收敛。令我感到惊讶的是当 dropout 为 0.1 时模型的效果最好（虽然也只是相对好一点点），猜测可能还是因为目前的模型比较简单，所以较低的 dropout 值就足够了。

五、问题思考

1. 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。

我采用的是固定迭代次数的方式，综合训练时间和准确率等值，取 epoch=10，此时训练结果收敛且没有明显的过拟合现象。

在实际训练中，当训练集和验证集的准确率或损失值收敛时，可以停止训练。此外，当模型出现过拟合时需要停止训练，可以通过验证集的准确率和损失值变化来判断模型是否出现过拟合。

固定迭代次数的优点是，在资源充沛的条件下可以多跑一些 epoch，从而可以非常直观地观察比较不同模型的性能。缺点是时间可能耗费过长，比如在本实验中训练双向 LSTM 模型时。

通过验证集调整的优点是可以有效地提高模型训练的效率，在任务时间不足的条件下将模型的功能最大化。缺点是无法直观地观察比较不同模型的性能，且无法有效保证最后训练好的模型是全局最优解。

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？（现有的初始化方法为零均值初始化，高斯分布初始化，正交初始化等）

本次实验中 MLP 模型采用的是高斯分布初始化，其余两个模型采用的是 Pytorch 的默认初始化。

对于不同的初始化方法：

零均值初始化将参数初始化为零，并不常用，因为尽管这样可以加快模型训练时的收敛速度，但是由于所有参数都是一样的，可能导致模型的对称性问题。

高斯分布初始化将参数初始化为服从高斯分布的随机数，可以打破参数之间的对称性，使得模型更容易学习。适用于深度神经网络，特别是在使用激活函数时，如 ReLU。但是需要注意初始化参数的标准差，过小会造成梯度消失，过大会造成梯度爆炸。

正交初始化将参数矩阵初始化为正交矩阵，有助于避免梯度消失或梯度爆炸问题，同时保持了参数矩阵的稳定性。适用于循环神经网络（RNN）等需要长期依赖的模型。

Xavier 初始化会根据输入和输出节点数量自适应调整初始化参数，旨在保持信号在前向传播过程中的方差恒定，以避免梯度消失或梯度爆炸。适用于全连接层或卷积层。

He 初始化由 Kaiming He 等人提出，与 Xavier 初始化类似，但适用于使用 ReLU 激活函数的网络。He 初始化根据激活函数的斜率来调整初始化参数，以更好地适应 ReLU 的特性。

3. 过拟合是深度学习常见的问题，有什么方法可以防止训练过程陷入过拟合。

可以通过下列几种常见方式防止训练过程陷入过拟合：

（1）数据扩充：增加现有数据量，更多的数据可以提供更全面更准确的信息，增强模型的泛化能力，减少数据偶然性的影响，提供更多的多样性。

（2）交叉验证：将数据集分为训练集和验证集，再随机选择数据进行训练和验证。这样可以更全面地评估模型的泛化能力，减少模型对某种特定数据的拟合。

（3）正则化：在模型的损失函数中添加额外的惩罚项来限制模型参数的大小，可以有效地降低模型的复杂度，防止模型在训练数据上过于灵活地拟合噪声和细节，从而提升其在未见过数据上的泛化能力。

（4）早停 Early Stopping：在模型训练过程中监控验证集性能，在验证集性能达到最佳时停止训练，能够有效地帮助找到一个适当的训练轮数，避免过拟合与浪费资源。

（5）Dropout：在训练过程中随机地丢弃一部分神经元的连接，从而降低模型的复杂度，减少神经网络的过拟合风险。

4. 试分析 CNN，RNN，全连接神经网络（MLP）三者的优缺点。

CNN：

优点：

• 局部感知性：CNN 通过使用卷积核进行局部感知，能够更好地捕捉图像中的局部特征。这种局部感知性使得 CNN 在处理图像时能够有效地提取细节信息，并具备良好的空间不变性。

• 参数共享：同一个卷积核可以在整个输入图像的不同位置上进行特征提取，减少了网络的参数量，降低了过拟合的风险，并且使得模型更具有泛化能力。

• 空间层次结构：CNN 采用多层卷积和池化操作，从低级到高级逐渐提取图像中的抽象特征。这种空间层次结构使得 CNN 能够处理不同层次的特征，并捕捉到图像中不同尺度上的信息。

缺点：

• 计算复杂性：由于 CNN 的层数较多，参数量较大，模型的计算复杂性较高，会增加一定的时间和资源消耗。

• 数据需求：CNN 对训练数据的要求较高，若训练数据集较小或不平衡，可能会导致过拟合或模型泛化能力不足。

RNN：

优点：

• 处理序列数据：适用于处理序列数据，能够有效地捕捉序列中的时序信息。

• 共享权重：RNN 在每个时间步都使用相同的参数，可以有效地共享权重，减少模型的复杂度和训练的参数数量。

• 上下文依赖建模：RNN 能够记忆之前的信息，并在后续时间步中利用该信息进行预测或决策，适合于处理依赖于上下文的任务。

缺点：

• 梯度消失/爆炸：RNN 在反向传播时，由于参数共享和多次连乘的特性，容易出现梯度消失或梯度爆炸的问题，导致模型难以训练或无法收敛。

• 长期依赖问题：由于梯度消失，RNN 在处理长序列时难以捕捉到长期依赖关系，只能有效利用较短的上下文信息。

MLP：

优点：

• 训练速度快：由于 MLP 基本是线性结构，对于简单的数据集，MLP 的处理和训练训练很快，时间效率高

• 结构简单，利于了解和调试。

缺点：

• 容易过拟合：由于全连接网络的参数量较大，容易出现过拟合现象。

• 空间层次结构：对于具有较为复杂的空间结构的数据，如图片和语音等，MLP 的效果可能不太理想。

六、心得体会

刚开始本次实验时我感到十分迷茫，因为之前没有一点神经网络的基础，虽然课堂上老师讲了一点关于这几个模型的知识，但我仍然是一头雾水，不知道从哪里开始。不过好在由于现在 AI 和神经网络的飞快发展，网上已经有很多资料供我参考与上手，框架都是现成的，用的时候直接套用就可以了。

不过真正开始做的时候我才发现事情没有我想象的那么简单，由于对 torch 的操作不太熟练，在 torch 向量的长度对齐上我踩了很多雷，几乎整个实验的 debug 过程都用来调整向量的长度了。此外，在写 MLP 模型时我一开始并没有使用词向量嵌入的方式，得到的准确率和 f-score 值都只有 50%左右，在修改了模型结构后提升到了 80%左右，我体会到了虽然模型的框架都是现成的，但是其中的一些小细节还需要自己对模型和任务足够了解才能清楚地实现。

本次实验对我来说另一个困难点在于参数的调整，由于参数的种类和数量颇多，我一开始有种无从下手的感觉，在查阅了一些资料后才选择了几个相对来说重要点的参数进行调整。

在数据的预处理过程上我也花费了很多时间，尽管这次不需要我们进行分词，但需要把词转换成词向量，再进行训练。由于对词向量的了解不够充分以及对 numpy 库使用的熟练度有待提升，数据预处理对我来说算是一个大工程，也让我认识到数据的重要性。在实际的工程中，数据量可能会更庞大更复杂，所以需要我们更细致地处理，才能发挥模型最大的效果。

本次实验从老师课堂上讲授的知识出发，我完成了一次完整的训练神经网络的过程，包括数据预处理、模型搭建、训练与评测，对于三类模型有了更深刻的体会，也认识到了自己在相关知识点上的疏漏与在 numpy 和 pytorch 等工具上使用的不足，希望在以后的学习中能够不断提升自己的能力。

七、参考资料

1. TextCNN

https://blog.csdn.net/u013421629/article/details/100523262

https://blog.csdn.net/Tink1995/article/details/104708678/

https://github.com/morningmoni/HiLAP/blob/master/TextCNN.py

2. 双向 LSTM

https://blog.csdn.net/xifenglie123321/article/details/132666978

3. 评价指标与工具

   https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

   https://blog.csdn.net/qq_44864833/article/details/131295460

4. 避免模型过拟合

   https://blog.csdn.net/qq_34160248/article/details/132390263

5. 模型优缺点分析

   https://www.eefocus.com/e/1611711.html

   https://blog.csdn.net/m0_68178753/article/details/135003960