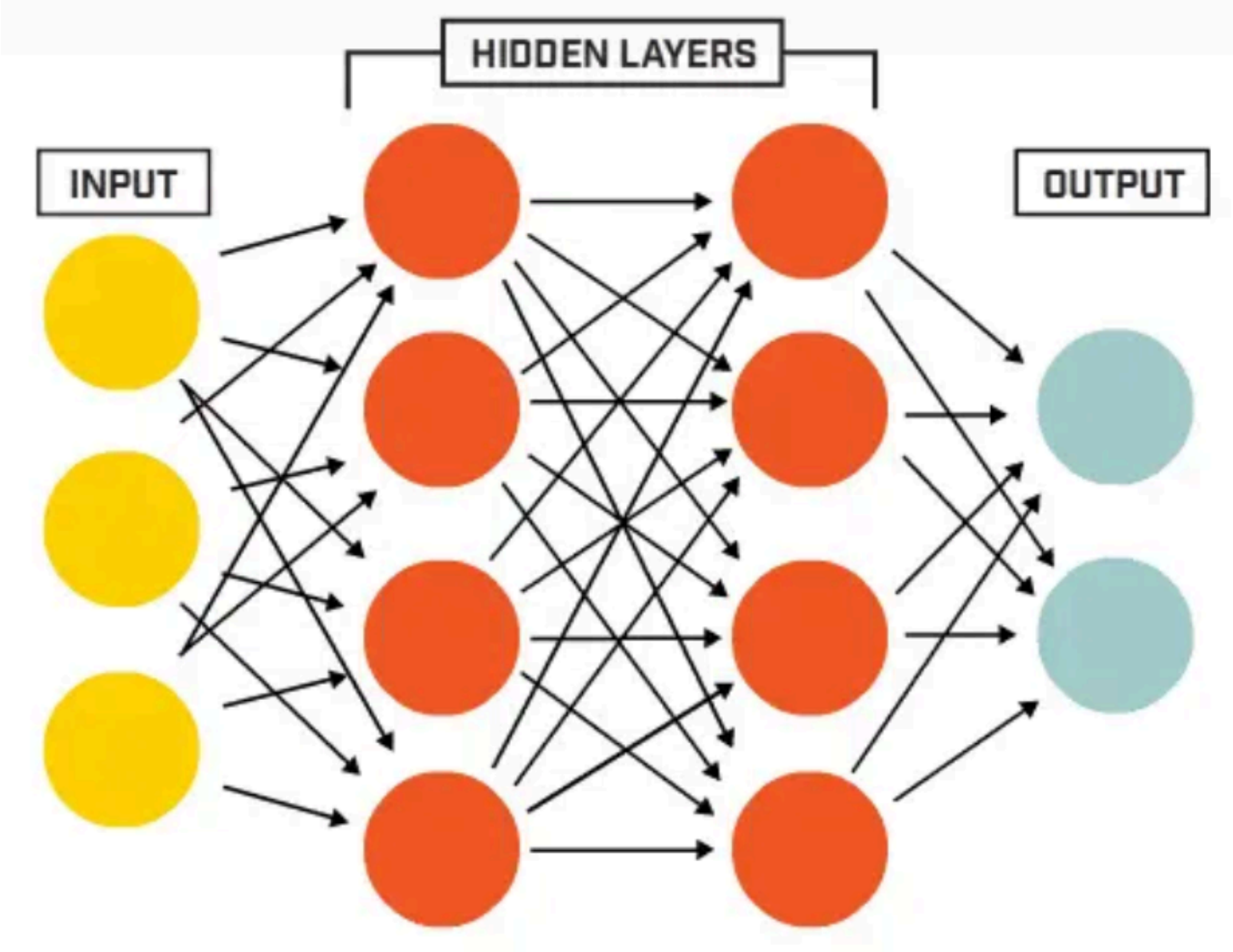
文本情感分类实验报告

张驰 2022010754 zhang-ch22@mails.tsinghua.edu.cn

一、模型结构

本次作业一共实现了三个模型，分别是双隐藏层的 MLP、text-CNN、和双向 LSTM。

1. MLP

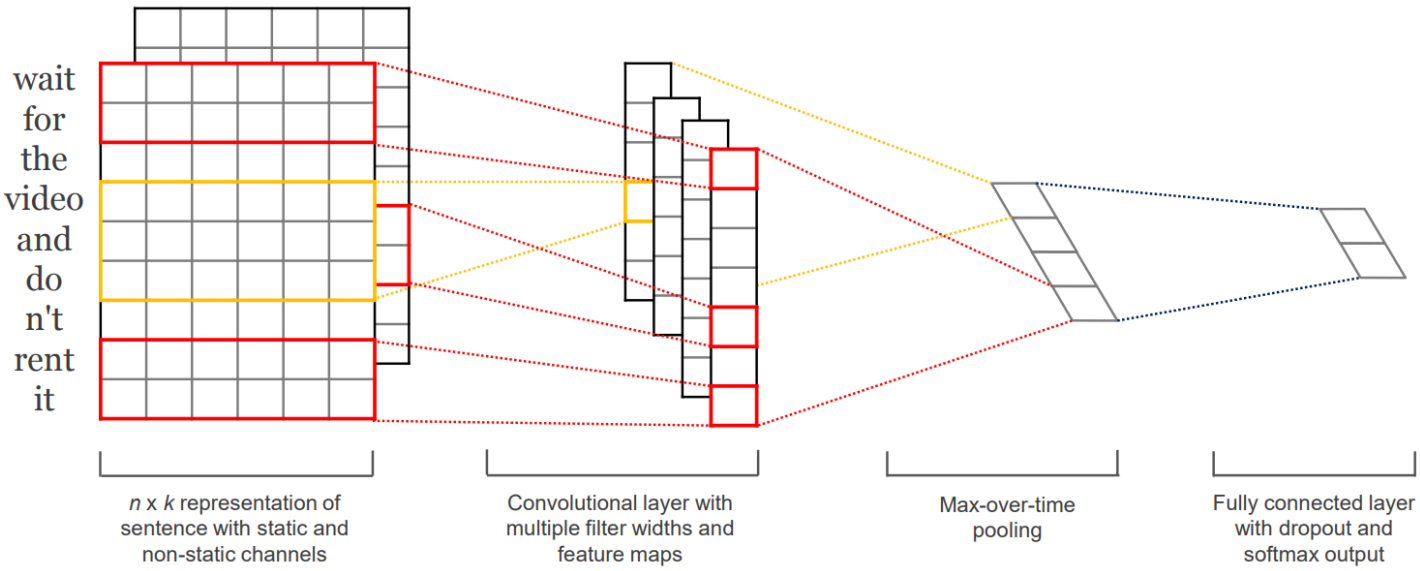


MLP 的模型结构较为朴素，相邻两层之间所有神经元互相连接。这次作业中，我采用了双隐藏层的 MLP 作为 baseline 模型。

模型的计算流程如下：

1. 以词向量作为输入，得到 `seq_len x embed_dim` 形状的输入。
2. 将每一个句子的二维词向量组拼接成一维向量，送入第一个线性层，投影到128维，用 ReLU 激活，添加 dropout。
3. 将128维的隐藏层送入第二个线性层，投影到64维，用 ReLU 激活，添加 dropout。
4. 将64维的隐藏层送入第三个线性层，投影到2维，并用 softmax 函数得到概率输出。

2. text-CNN

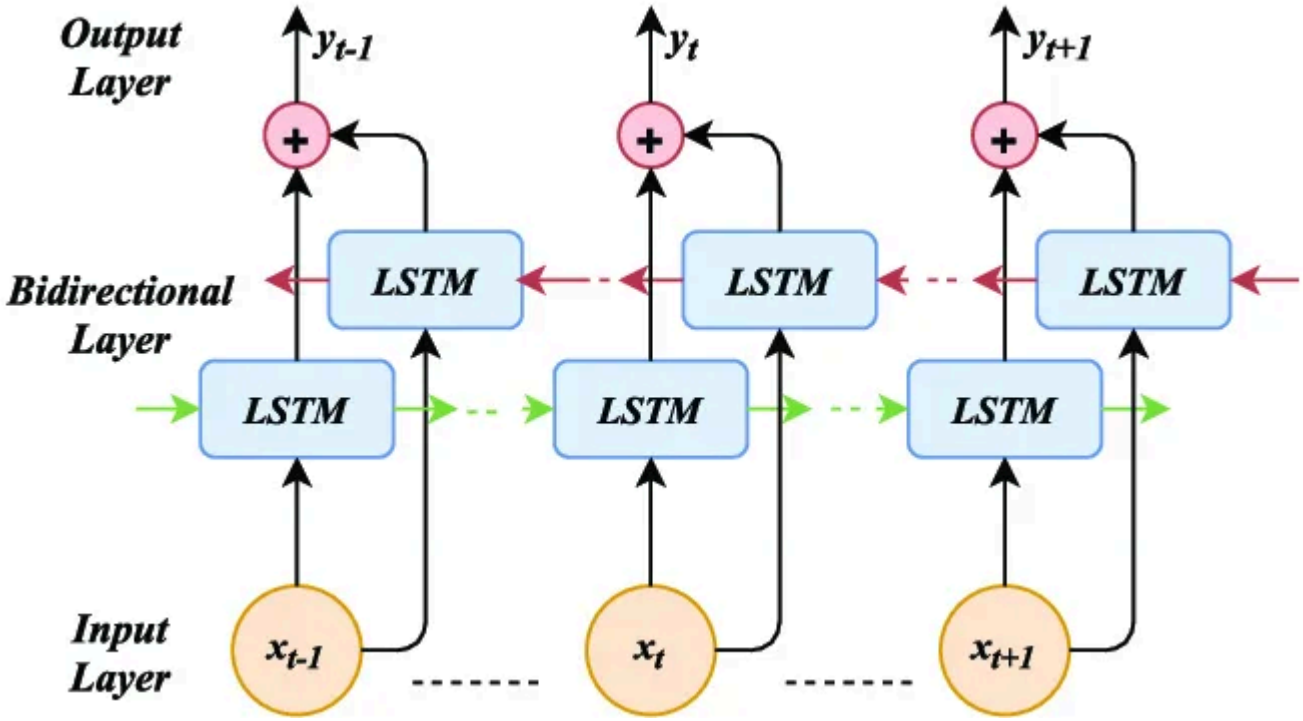


text-CNN 的模型结构仿照论文Convolutional Neural Networks for Sentence Classification实现。结构图如上。

模型的计算流程如下：

1. 以词向量作为输入，得到 `seq_len x embed_dim` 形状的输入。
2. 在 `seq_len` 的维度上做一维卷积，使用多个不同大小的卷积核，获得多个（卷积核数量个）长度不同（由卷积核大小决定）的特征向量。
3. 对每一个特征向量，做 max pooling 保留最大特征，用 tanh 激活，添加 dropout。
4. 将这些特征通过一个线性映射投影到二维，用 softmax 函数得到概率输出。

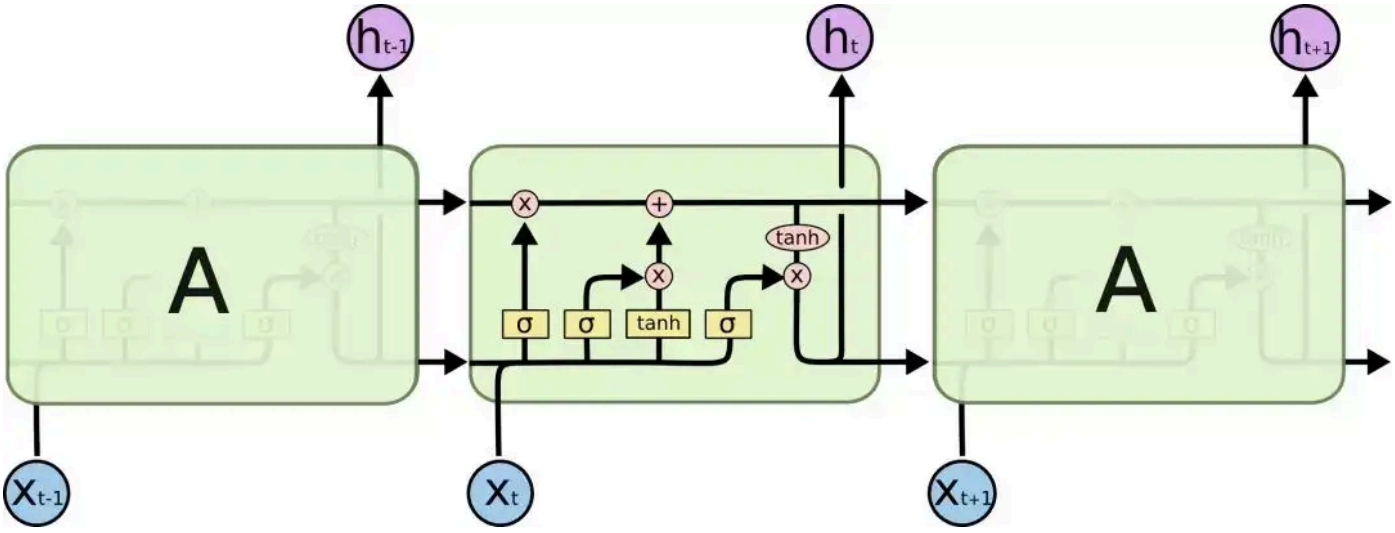
3. BiLSTM



BiLSTM 的模型结构仿照论文Bidirectional LSTM-CRF Models for Sequence Tagging实现。结构图如上。

模型的计算流程如下：

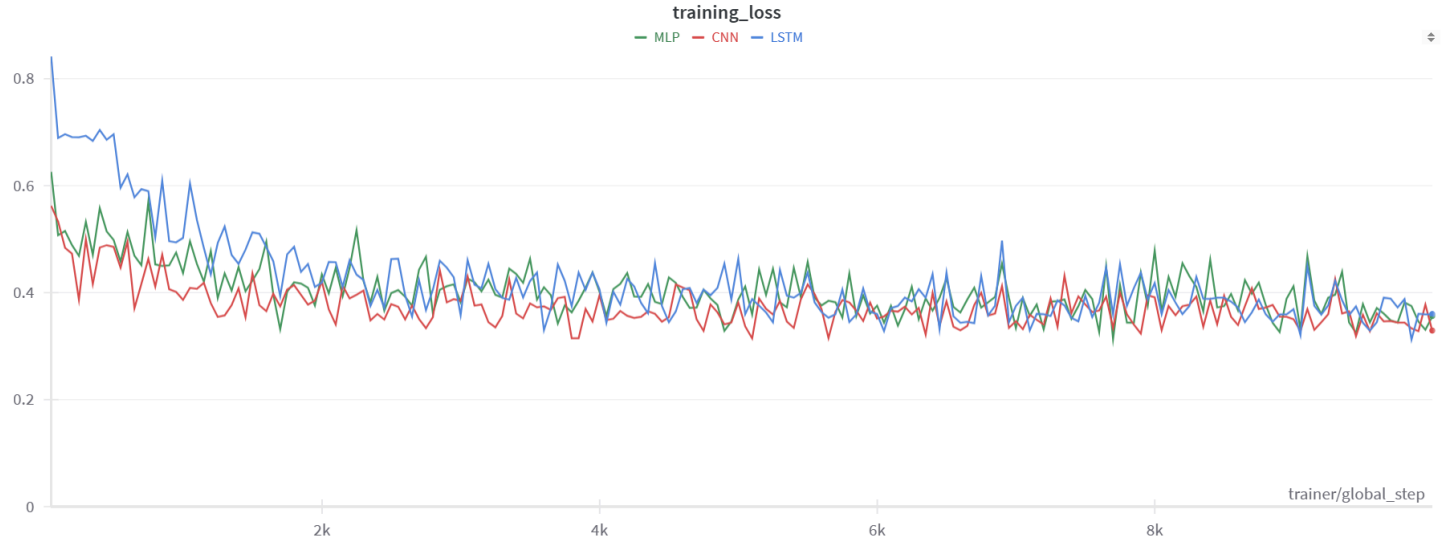
1. 以词向量作为输入，得到 `seq_len x embed_dim` 形状的输入。
2. 双层前向 LSTM：对一个句子中的每一个词向量，送入一个 LSTM 模块，同时接收上一个词的 LSTM 前向输出。每一个 LSTM 模块的内部结构如下图：

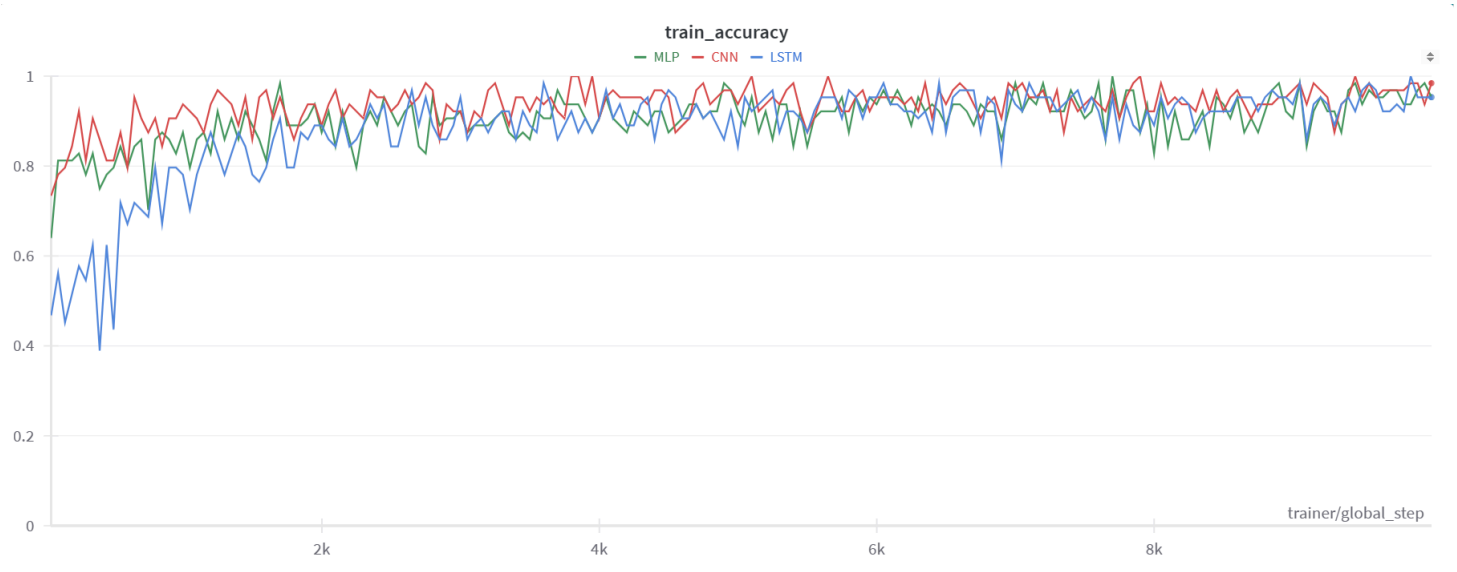


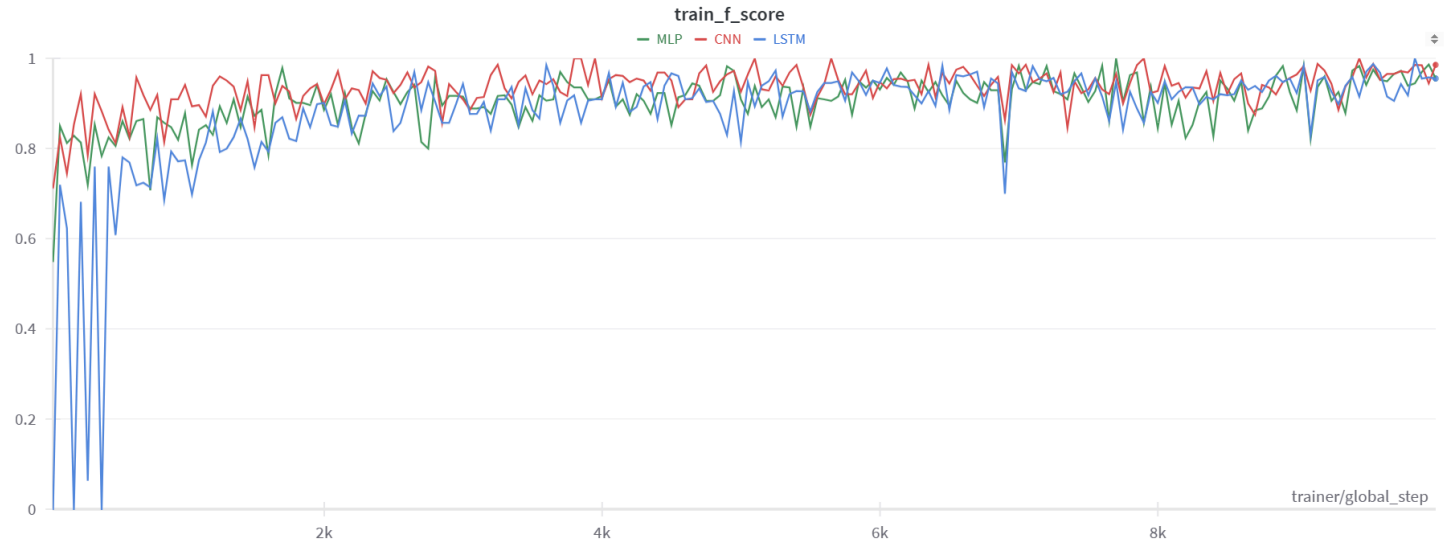
3. 将这一层 LSTM 的输出（$h_1, ..., h_n$）送入第二个 LSTM 层作为输入，得到输出。
4. 同理得到双层后向 LSTM，即每个 LSTM 模块接收后一个词的前向输出。同样重复两层。
5. 将前后向 LSTM 输出结果拼接起来，通过一个线性映射投影到二维，过 softmax 函数得到概率输出。

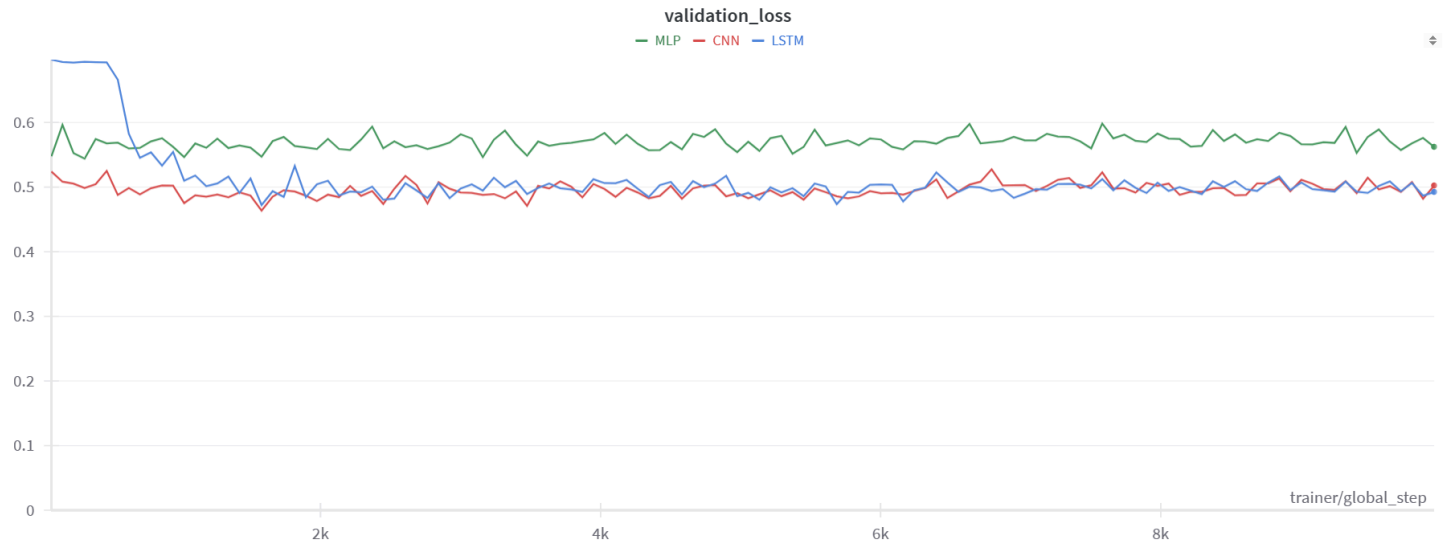
二、实验结果

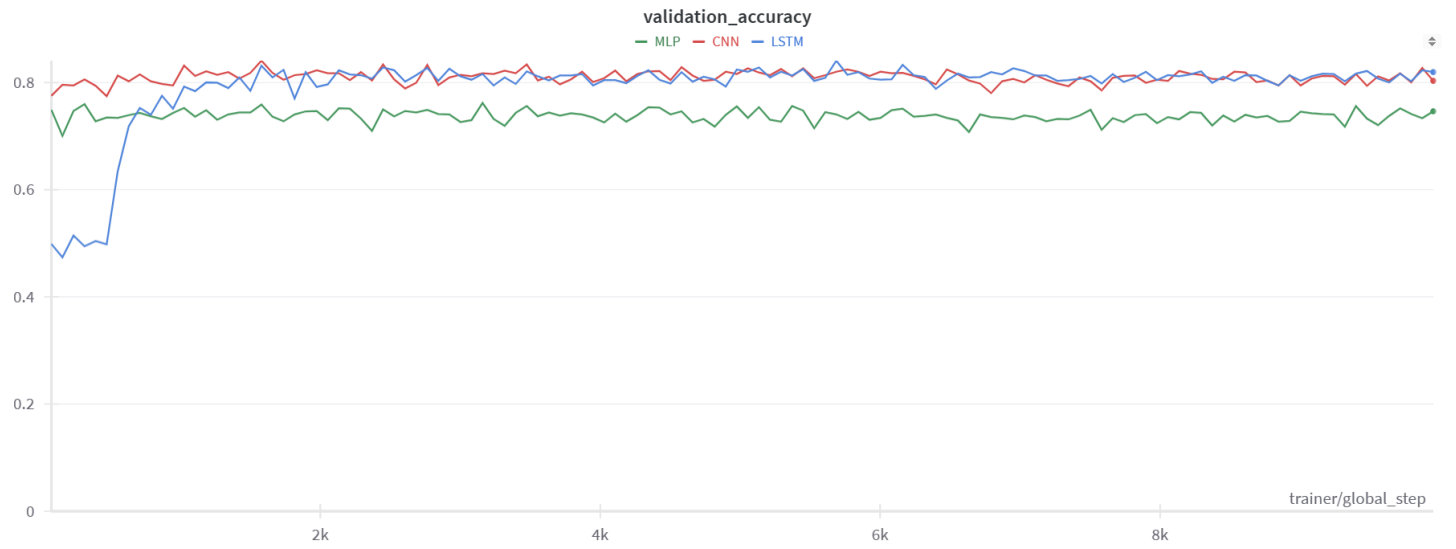
在给定数据集的 train 部分上训练，同时监控 validation 部分的测试情况。结果如下图：






train_f_score
MLP
CNN
LSTM
1
0.8
0.6
0.4
0.2
0
2k
4k
6k
8k
trainer/global_step


validation_loss
MLP
CNN
LSTM
0.6
0.5
0.4
0.3
0.2
0.1
0
2k
4k
6k
8k
trainer/global_step


validation_accuracy
MLP
CNN
LSTM
0.8
0.6
0.4
0.2
0
2k
4k
6k
8k
trainer/global_step

最终在 test 集上的表现如下：

模型\指标	accuracy	F-score
MLP	0.7097	0.681
text-CNN	0.871	0.8775
BiLSTM	0.8387	0.8605

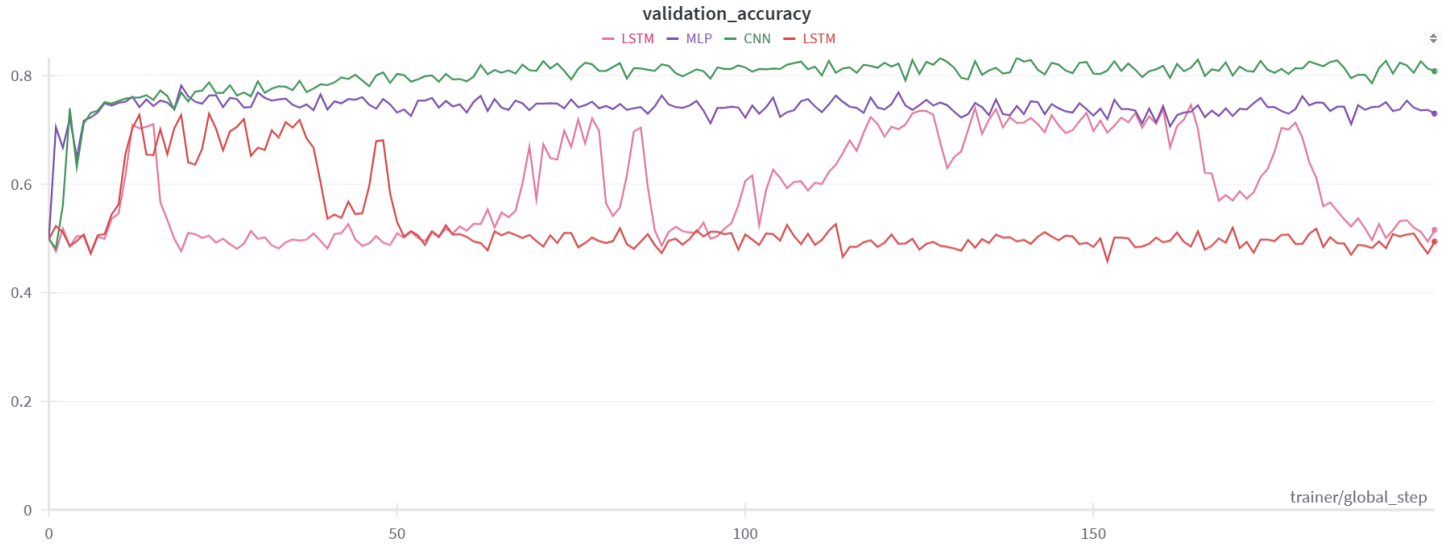
三、参数效果

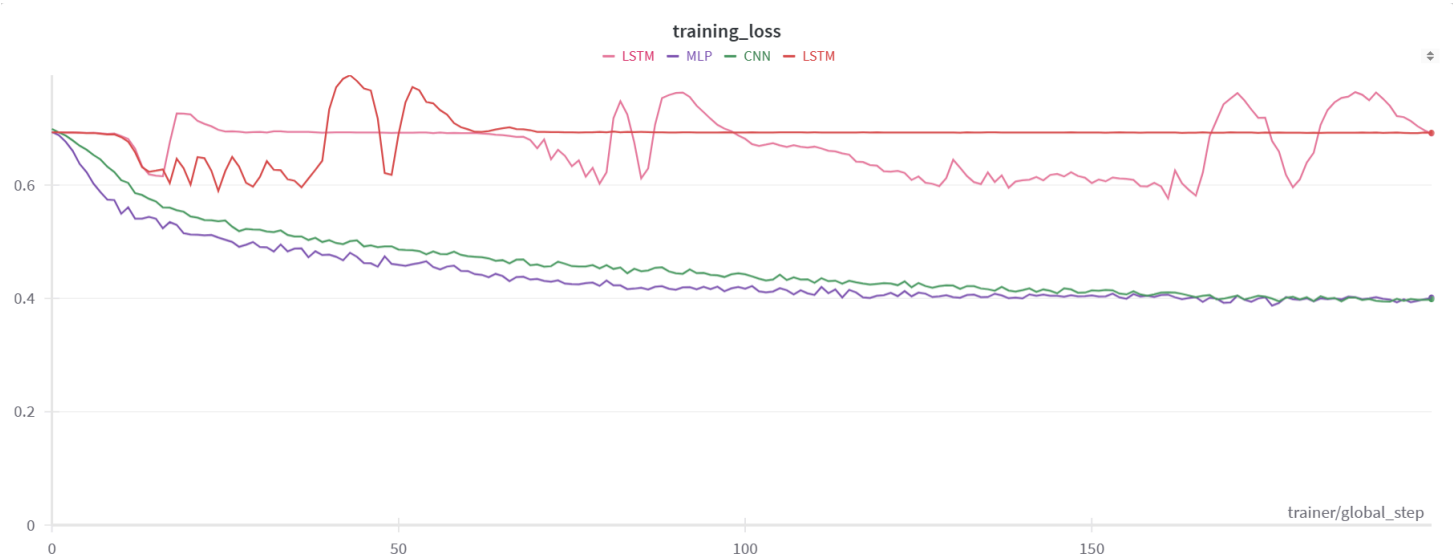
1. batch_size

由于数据集较小且显存充裕，我尝试让模型的一个 batch 直接包含整个数据集（batch_size = 20000），我认为，这样模型能够克服“随机梯度下降”的“随机”性，取得更好的训练效果，同时由于充分利用显存进行并行计算，能够加速收敛速度。

事实上，训练速度的确有很大提升。显卡支撑这些计算绰绰有余，速度的瓶颈出现在 dataloader 上。然而，模型的效果并没有显著提升，MLP 和 CNN 的效果基本没有显著变化，而在训练 LSTM 时甚至出现了 train loss 不再下降，疑似梯度消失的情况。

如图：





上面两张图分别显示了 validation accuracy 和train loss，包含了两次训练LSTM的结果。我猜测这可能是在过大的 batch 中做平均，导致梯度数值太小导致梯度消失。

2. pad_len

由于句子的长度本身是不均匀的，我们在将 embedding 送给模型前要先将他们补齐到同一长度。这就可能会带来一些语义的噪声或者缺失。

目前实验的 pad_len 根据普遍句子长度设置为64。我尝试将它扩大到120，结果如下：

模型\指标	accuracy	F-score
MLP	0.7527	0.7292
text-CNN	0.8172	0.8091

模型\指标	accuracy	F-score
BiLSTM	0.7903	0.7741

可见，多出的冗余 padding 会干扰到模型训练。

四、模型差异

可以看出，在上面的三个模型中，text-CNN 表现最好，BiLSTM 其次，作为 baseline 的 MLP 表现最差。

详细的分析见第五部分：三种模型的优缺点。

五、问题思考

1. 停止训练

由于我们的数据集规模并不大，如果让模型长时间的训练很有可能出现过拟合问题。即使没有过拟合，也容易造成算力的浪费。对此，我的策略是在模型产生过拟合状况时停止训练。我们检测 validation set 上的准确率，当准确率连续10个epoch低于之前的最高准确率时，判定模型出现过拟合，从而停止训练。此时在 test set 上进行测试。

test set 上的最高准确率未闭和 validation set 同时出现，如果在 test set 上进行早停判定可以把测试准确率提升的更高，但这样做有违 test set 的使用原则。

然而经过测试，这样的早停效果并不好。validation accuracy 抖动变化很大，早停策略的要求会很早地被满足，而此时模型还没有训练充分，效果不如训练很长的模型。尝试将可容忍的 epoch 数量从10调大到50或100，结果有所改善（因为训练总步数增多），但仍然不够。

2. 参数初始化

零均值初始化：将所有参数初始化为0。这种做法是不适合神经网络训练的。如果将所有参数初始化为0，那么每一层的所有神经元就拥有了数值上的对称性。在训练过程中，因为每一个神经元上的梯度回传公式是相同的，这种对称性无法破坏，导致模型在训练中始终是对称的。

如下：设第 k 层的神经元为 z_i^k，$z_i^0 = x_i$ 为输入，且 $z_i^{k+1} = \sigma(\sum_j w_{ij}^k z_j^k + b^k)$，则初始化后，$\forall k > 0, z_1^k = z_2^k = z_3^k = ...$ 每一次优化，$\frac{\partial L}{\partial w_{ij}^k} = \frac{\partial L}{\partial z_i^{k+1}} \frac{\partial z_i^{k+1}}{\partial w_{ij}^k} = \frac{\partial L}{\partial z_i^{k+1}} z_j^k$。由于 z_i^{k+1} 之间的对称性，知这个值与 i 无关。这回导致下一次前向运算时，z_i^{k+1} 依然全同。

这样的话，一个有多个神经元的神经网络层只能发挥一个神经元的功能，很难完成需要的任务。

高斯初始化：将所有的神经元初始化为均值为0，标准差为较小的数的高斯分布，这种方法可以有效地引入不对称性，让模型能够充分地优化，适用于多种神经网络的训练。这种方法的好处在于高斯分布易于采样，同时能方便理论计算。

正交初始化：在初始化中保证参数矩阵的列向量相互正交。这有助于防止梯度消失或梯度爆炸问题，并且可以提高网络的收敛速度和稳定性。适用于各种循环神经网络。

Xavier初始化：根据每一层神经元数量调整初始化采样的方差。由于神经网络每一层神经元数量不同，可能导致不同层神经元的取值方差发生过大变化。如下：

$z_i^{k+1} = \sigma(\sum_j w_{ij}^k z_j^k + b^k)$，考虑激活函数为线性或者其他保方差的函数，且所有参数独立，则 $\mathrm{Var}(z_i^{k+1}) = \sum_j \mathrm{Var}(w_{ij}^k)\mathrm{Var}(z_j^k)$。设第 k 层有 m_k 个神经元，若要使得这个方差稳定不变，则需要在初始化时让 $\mathrm{Var}(w_{ij}^k) = \dfrac{1}{m_k}$。

由于梯度反向传播时也会在反向出现同样的问题，此时需要让 $\mathrm{Var}(w_{ij}^k) = \dfrac{1}{m_{k+1}}$，故在实践中往往取两者的调和均值。

3. 防止过拟合

如果模型参数量过大，而数据集规模太小，那么训练中模型很可能过拟合到训练集上，而应用中实际作用一般。对此的防止策略有：

1. 扩大数据集或者缩小模型。
2. 为模型添加 dropout，让模型的多个部分分别向训练数据拟合，也能缩小单次训练的模型规模，防止过拟合。
3. 在损失函数中添加正则化项，防止模型产生过于偏离平均的训练参数，从而防止模型因为数据噪声或者非主要特征而过拟合到训练集上。
4. 早停，在 validation 效果达到最好时，提前停止训练。

4. 三种模型的优缺点

可以看出，在上面的三个模型中，text-CNN 表现最好，BiLSTM 其次，作为 baseline 的 MLP 表现最差。MLP 表现不佳容易理解，因为模型本身结构不能够体现任务的特殊信息，存在大量没有实际意义的权重和联系，导致模型效果较差。同时，这些权重可能会导致模型注意到训练集中的噪声特征，导致过拟合。可以注意到，在实验数据中，三种模型的训练 loss 与 accuracy 接近，但 MLP 在验证时出现了很大差距，表现出过拟合情况。

text-CNN 与 LSTM 都是负责自然语言处理的专用模型。前者可以用卷积和池化的方式快速提取句子的局部和全局特征，后者通过循环网络的方式捕捉语义信息。两个模型表现相近。注意到，多次实验表明 test 测试有一定的随机性，可能导致了一些差异。另外，text-CNN 表现更好可能是我严格参考了原论文的模型结构和参数，而 LSTM 的结构采用了很多更随意的设置。

六、心得体会

本次实验让我对自然语言处理有了更深刻的理解。LSTM 我早有耳闻，但是用 CNN 的方法来解决 NLP 问题是我原来没有考虑过的。当我看到这个作业我认为它的表现会不如 LSTM，没想到结果恰相反。这拓宽了我的学术眼界。

同时，这次几个模型完全为从零开始手写，让我对 pytorch 及相关的深度学习包更为熟练。实验报告的文字量略有些多。不过整体来看，仍是一次收获颇丰的作业。