文本情感分类-实验报告

于越洋

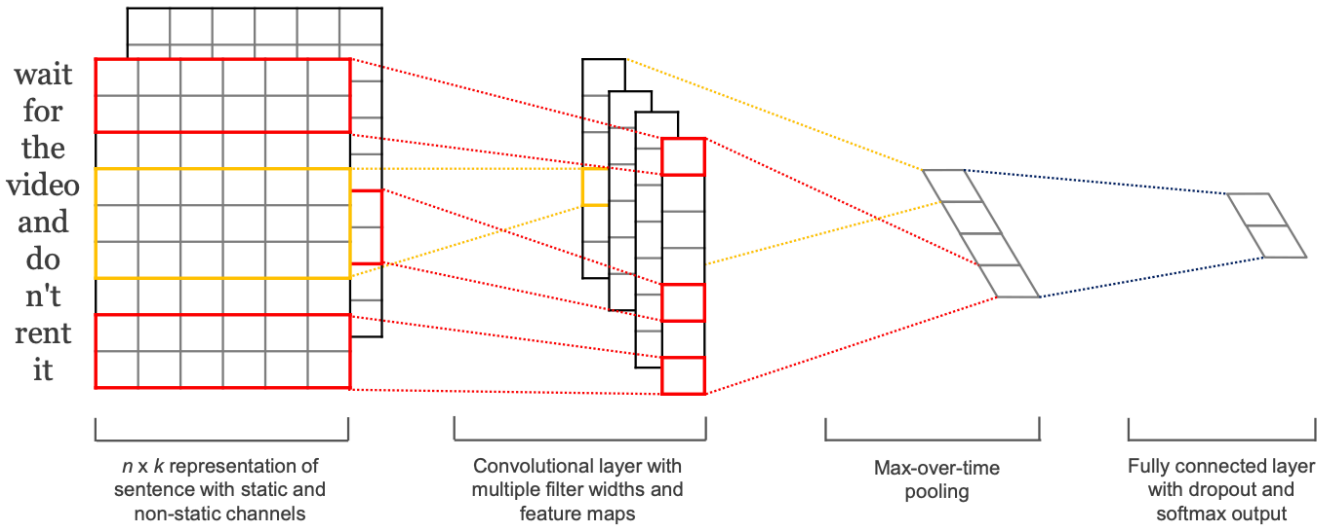
1. 模型结构

模型的实现主要在 `models.py` 文件中。

1.1 Text-CNN

模型设计

嵌入层：将词索引映射到词向量，每个向量表示一个词。
卷积层：设计了 4 个卷积层，卷积核大小分别为 `kernel_sizes = [2, 3, 4, 5]`，每个卷积层具有相同大小的通道数 `num_filters`，用于提取不同大小的文本特征。每个卷积操作后使用 ReLU 激活函数，并进行最大池化，获得最显著的特征。
全连接层：将提取的特征映射到分类空间。
Text-CNN结构如下图（使用参考文献中的示意图）



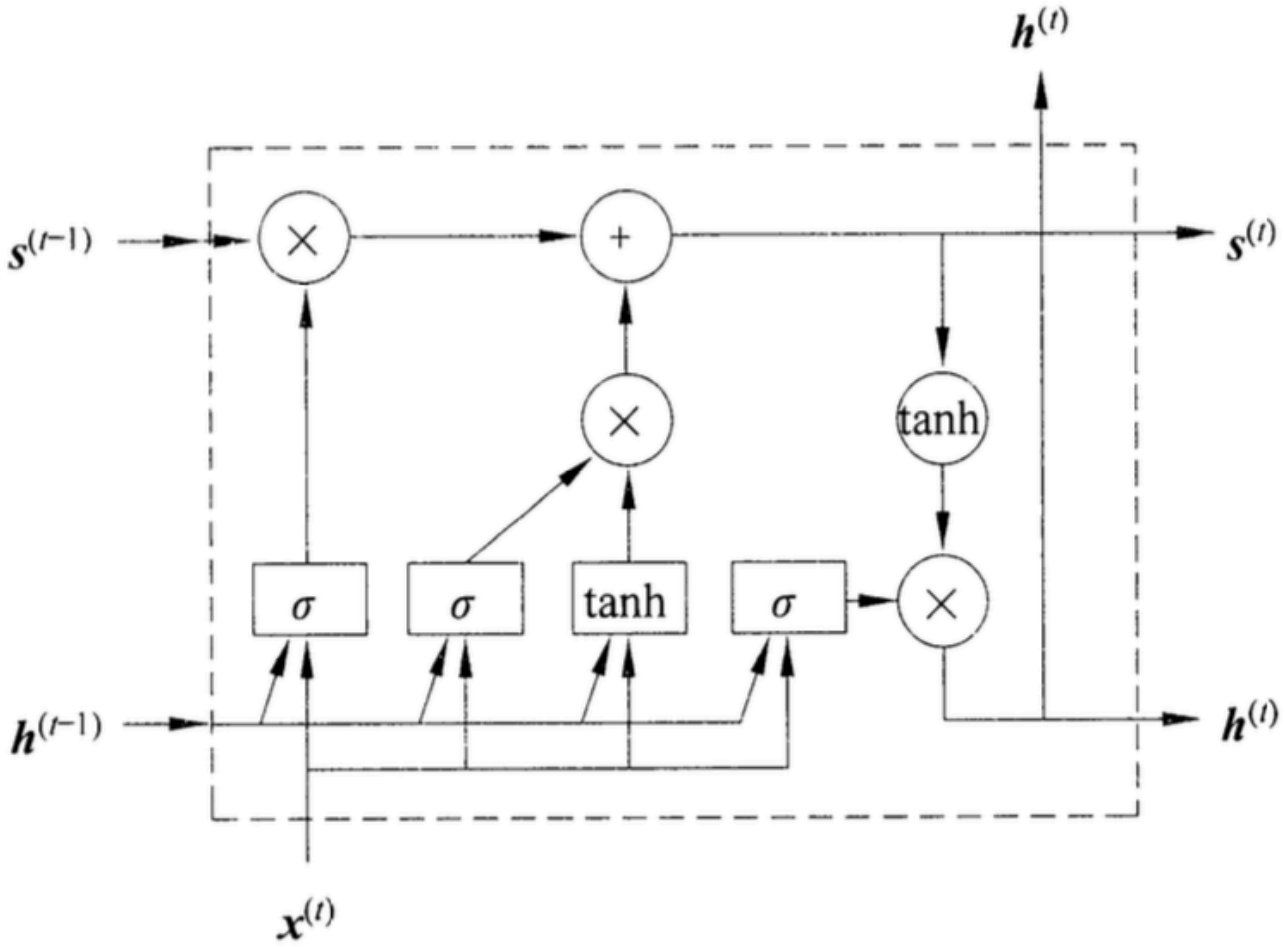
前向传播过程

1. 接收 `x: tensor` 作为参数；
2. 通过嵌入层将词索引转换为嵌入向量表示；
3. 进行卷积计算，并使用 ReLU 激活函数进行处理，引入非线性；
4. 对上一步的结果进行最大池化，并对结果进行拼接；
5. 使用全连接层进行分类，并使用 `softmax` 函数归一化，得到分类得分。

1.2 LSTM

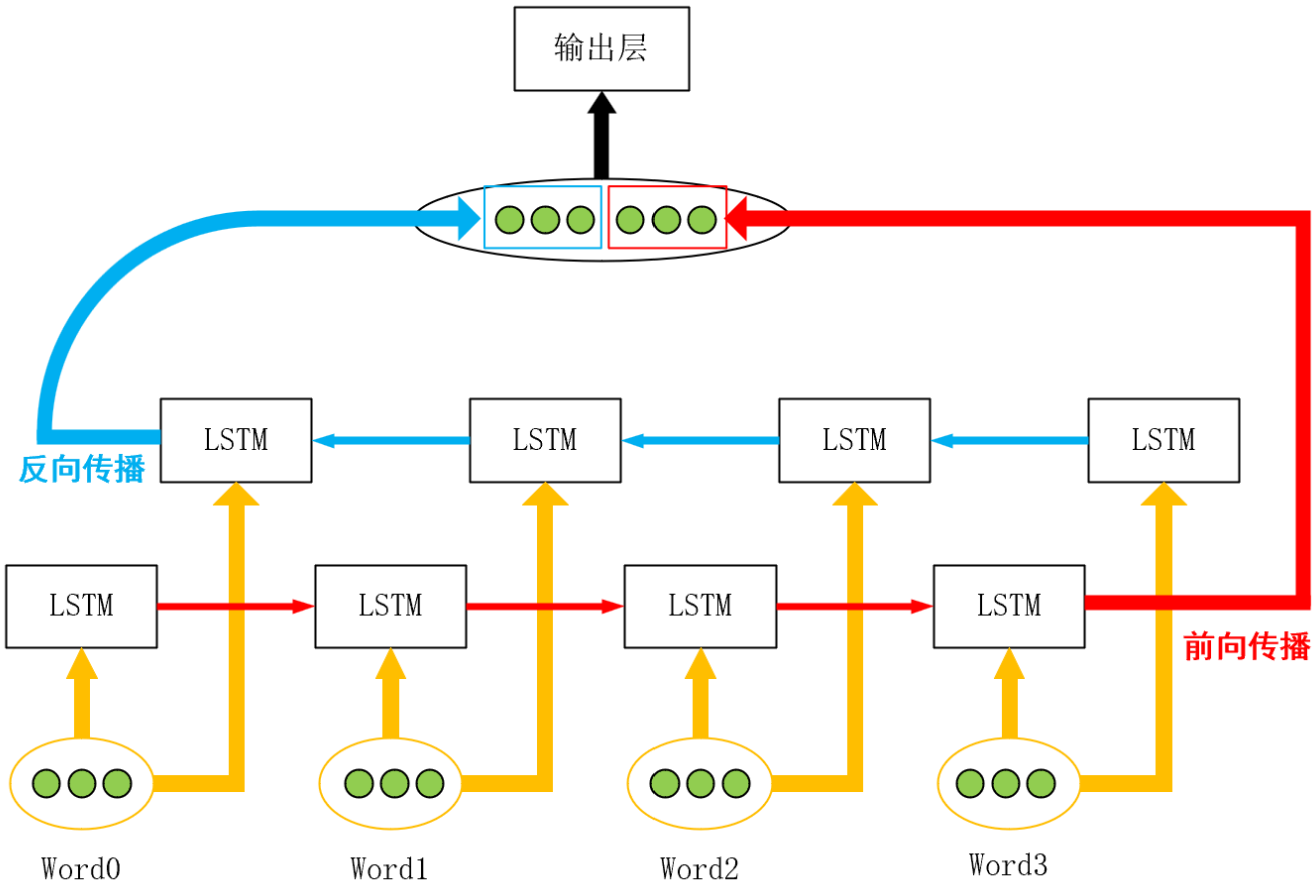
模型设计

嵌入层：将词索引映射到词向量，每个向量表示一个词。
LSTM层：以嵌入层输出作为输入，处理输入的序列数据，得到隐藏层输出。
全连接层：将 LSTM 的隐藏状态映射到分类空间。
LSTM 结构如下图（使用课本 98 页的示意图）



此处为避免梯度消失等问题，使用双向 LSTM 模型，即在前向传播的基础上增加反向传播，模型

结构图如下



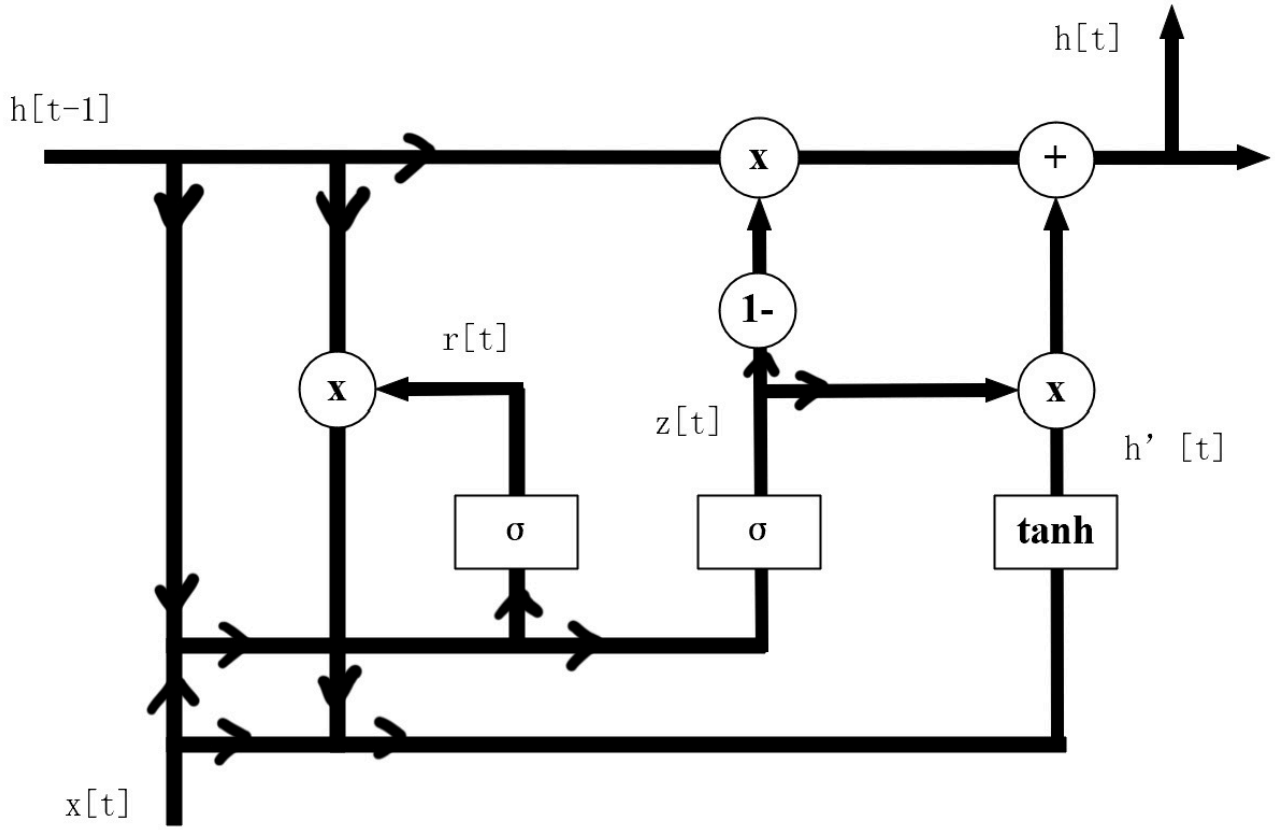
前向传播过程

1. 接收 `x: tensor` 作为参数；
2. 通过嵌入层将词索引转换为嵌入向量表示；
3. 将嵌入向量输入到 LSTM 层，得到所有时间步的隐藏状态，取最后一个时间步的前向隐藏状态 `x[:, -1, :self.hidden_dim]` 和第一个时间步的后向隐藏状态 `x[:, 0, self.hidden_dim:]` 进行拼接，得到 LSTM 层的输出。
4. 使用全连接层将上一步得到的输出映射到分类空间，并使用 `softmax` 函数归一化，得到分类得分。

1.3 GRU

模型设计

嵌入层：将词索引映射到词向量，每个向量表示一个词。
GRU 层：以嵌入层输出作为输入，处理输入的序列数据，得到隐藏层输出。
全连接层：将 GRU 的隐藏状态映射到分类空间。
GRU 结构如下图



与上面的 LSTM 一样，此处也使用双向模型，模型结构见 `1.2 LSTM` 中所给出的结构图。

前向传播过程

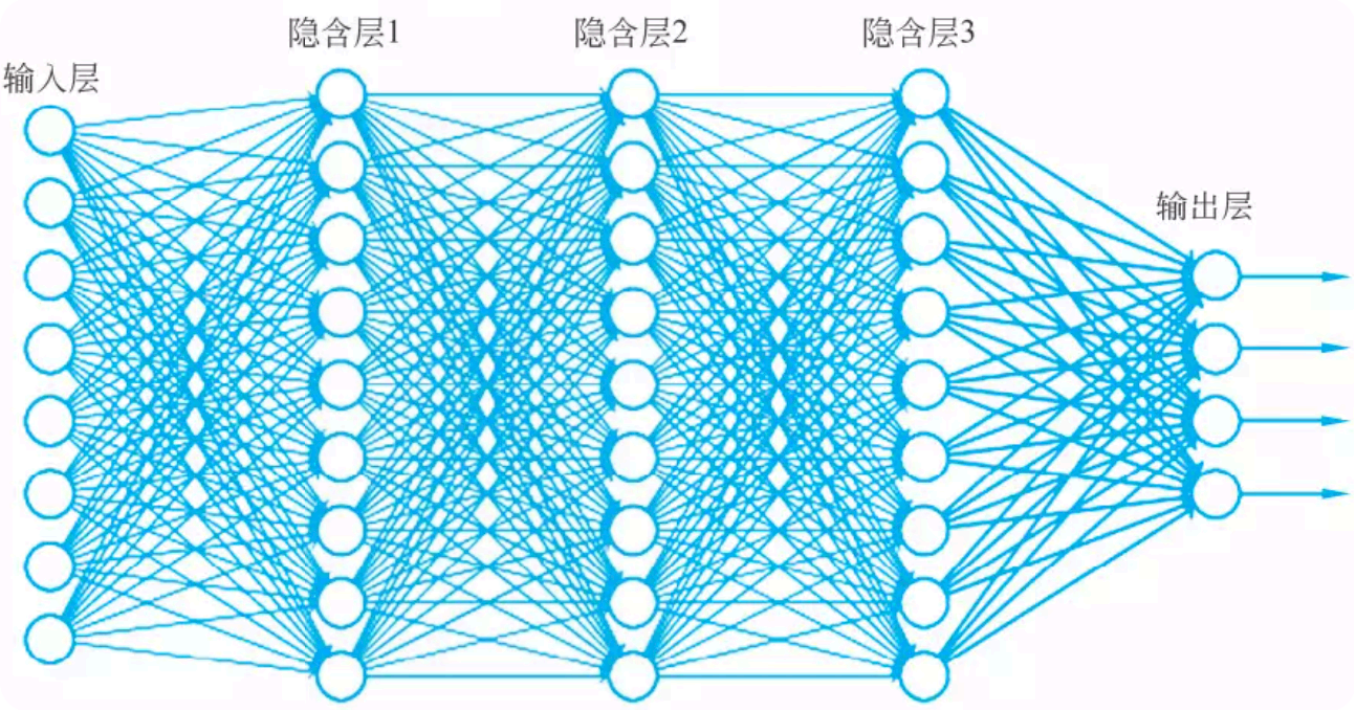
1. 接收 `x: tensor` 作为参数；
2. 通过嵌入层将词索引转换为嵌入向量表示；
3. 将嵌入向量输入到 GRU 层，得到所有时间步的隐藏状态，取出其中最后一个时间步的隐藏状态；
4. 使用全连接层将上一步得到的隐藏状态映射到分类空间，并使用 `softmax` 函数归一化，得到分类得分。

1.4 MLP (baseline)

模型设计

嵌入层：将词索引映射到词向量，每个向量表示一个词。
MLP 层（全连接层）：以嵌入层输出的平均池化结果作为输入，计算得到隐藏层输出。
全连接层：以经过 ReLU 激活函数处理后的隐藏层输出作为输入，将其映射到分类空间。

MLP 示意图如下（使用课本第 38 页配图，代码实现中仅有一个隐藏层）



前向传播过程

1. 接收 `x: tensor` 作为参数；
2. 通过嵌入层将词索引转换为嵌入向量表示；
3. 将嵌入向量输入到 MLP 层，得到隐藏层输出，并使用 ReLU 激活函数进行处理，引入非线性；
4. 使用全连接层将上一步的结果映射到分类空间，并使用 `softmax` 函数归一化，得到分类得分。

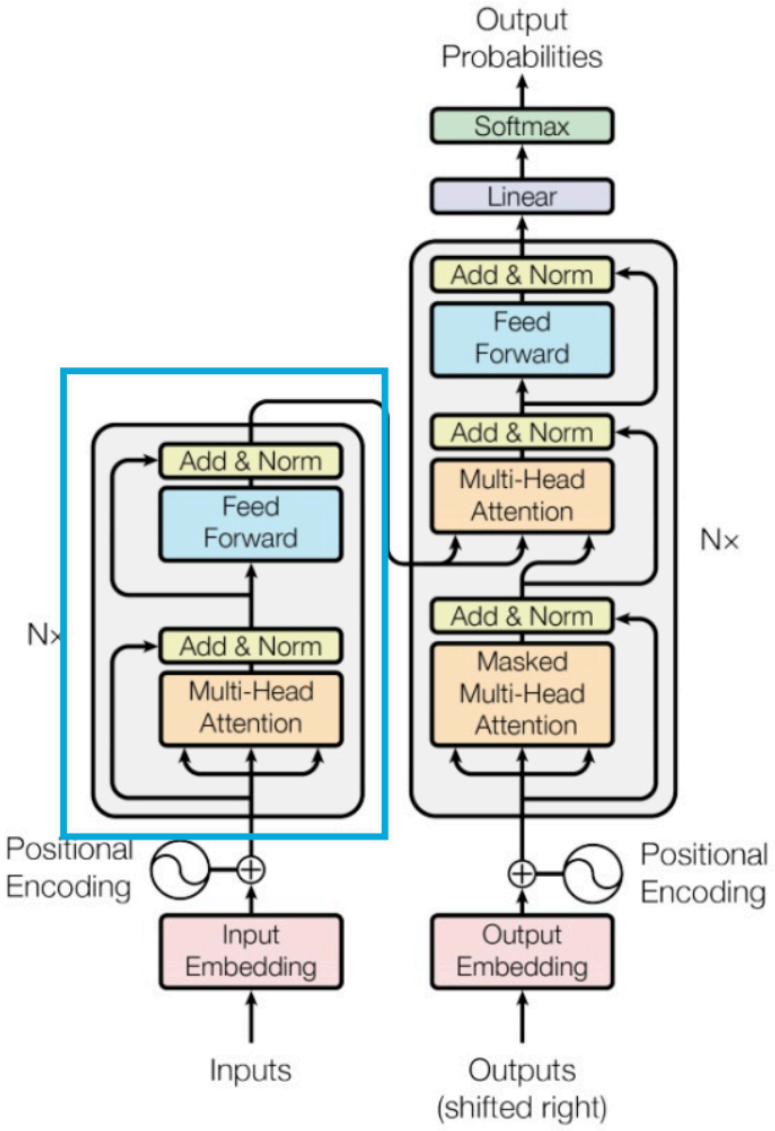
1.5 BERT（额外实现）

Bert 模型的原理参考了文章：
https://blog.csdn.net/u011412768/article/details/108015783
实现参考了仓库：
https://github.com/huggingface/transformers/tree/main/src/transformers/models/bert
中的 `configuration_bert.py` 和 `modeling_bert.py` 中注释提供的信息，以完成参数配置。

模型设计

嵌入层：使用 `BertEmbeddings`，通过传入配置参数实现 Bert 的嵌入层；
编码层：使用 `BertEncoder`，通过传入配置参数实现 Bert 的编码层；
全连接层：以经过池化和 Dropout 后的 Bert 编码结果为输入，将其映射到分类空间。
Bert 模型基于 Transformer 的 Encoder 实现（下图中左侧圈出的部分），其实际由多个 Transformer Encoder 堆叠而成。

* 下图引用 Transformer 的经典结构图



本次实现基于 transformer 库中已经配置好的 Bert 模型，修改词表大小 `vocab_size`、隐藏层大小 `hidden_size`、隐藏层层数 `num_hidden_layers`、中间层大小 `intermediate_size` 这几个参数，然后使用 `BertEmbeddings` 和 `BertEncoder` 组成我的 Bert 文本情感分类模型。

前向传播过程

1. 接收 `x: tensor` 作为参数；
2. 通过嵌入层，将词索引进行嵌入；
3. 将嵌入后的结果输入编码层，得到 Bert 模型的编码输出，取最后一层 Transformer 的输出；
4. 对输出结果进行池化（取 `<CLS>` 标记），并进行 Dropout；
5. 将上一步的结果输入全连接层，映射到分类空间，并使用 `softmax` 函数归一化，得到分类得分。

2. 实验流程

2.1 数据加载

数据加载的相关方法主要在 `dataset.py` 和 `utils.py` 两个文件中。

- `utils.py` 中的 `load_word2vec` 函数用于处理预训练的词向量，其中使用 `gensim` 库提供的用于加载词向量模型的函数 `load_word2vec_format` 进行加载，直接返回加载好的词向量模型 `word2vec`。
- `utils.py` 中的 `create_vocab` 函数基于加载好的词向量模型创建词表，即由词到索引的映射表 `word_to_index` 和由索引到词向量的嵌入矩阵 `embedding_matrix`。为应对语句需要填充和部分词不在词表中的特殊情况，词表中还添加了 `<PAD>` 和 `<UNK>` 两种特殊字符，分别用于填充语句和替换不存在的词语，其对应的词向量分别初始化为全 0 向量和随机向量，以更好地完成后续的数据处理。
- `utils.py` 中的 `load_data` 函数用于处理训练、验证、测试数据，直接按行读取，分离语句和情感标记，返回语句列表 `texts` 和标记列表 `labels`。
- `utils.py` 中的 `texts_to_indices` 函数用于将输入词序列转换成在嵌入矩阵中的索引，用于模型训练。
- `dataset.py` 中继承 PyTorch 中的 `Dataset` 类实现了一个 `TextDataset` 类，用于构建用于本次文本情感分类任务的数据集。其中存储转换后的文本序列索引列表和对应的标记，用于训练。

2.2 模型训练和评估

模型训练的主要逻辑在 `train.py` 中。

- 首先进行超参数设置，接收主函数传入的训练参数，如 `batch_size`、`learning_rate` 等；
- 然后进行数据加载，调用 `utils.py` 中定义的函数来加载预训练词向量、训练数据、验证数据、测试数据，并使用 `dataset.py` 中定义的 `TextDataset` 类来构建数据集，然后使用 PyTorch 提供的 `DataLoader` 作为数据加载器来加载数据；
- 然后根据选择的模型名，调用 `models.py` 中实现的对应模型，设定损失函数为交叉熵损失函数，设定优化器为 Adam 优化器，即可开始训练；
- 训练过程中，每个 epoch 内都进行训练、验证和测试，仅在训练过程中更新梯度，验证和测试过程中不更新梯度，每次训练、验证、测试都计算损失函数、准确率和 F-Score 以进行效果评估；
- 训练结束，保存模型。

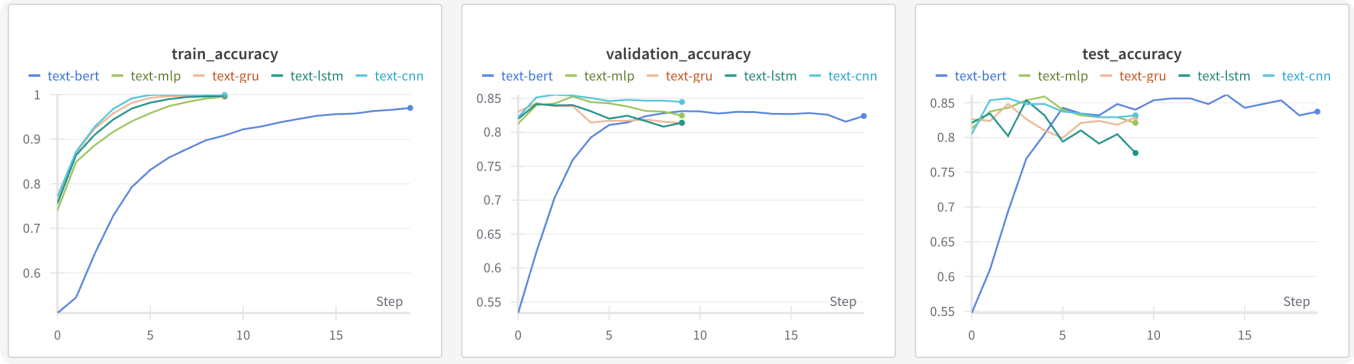
3. 实验结果

3.1 训练曲线

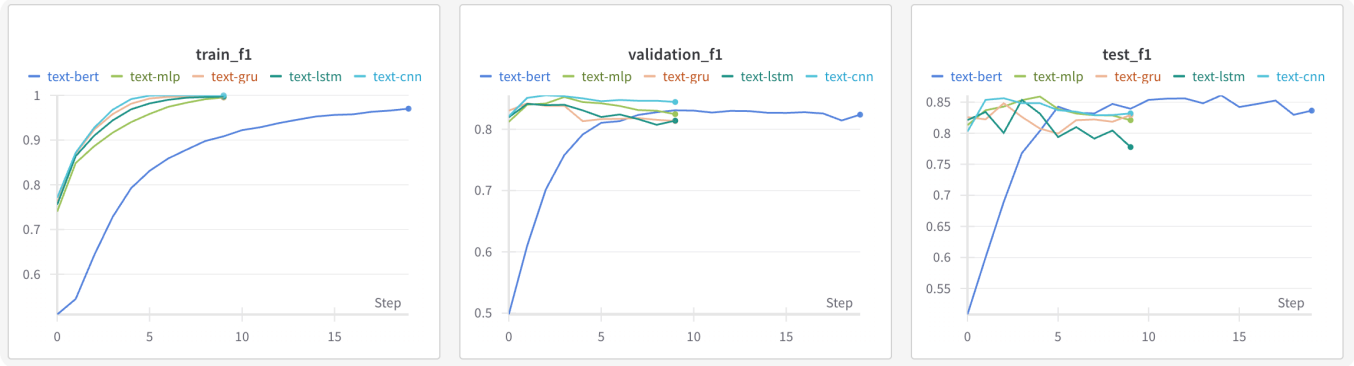
原本所有模型均迭代 10 次，后发现 BERT 模型收敛速度较慢，因此补充了一次迭代 20 次的 BERT 模型训练。

Loss 曲线

Accuracy 曲线



F1-Score 曲线



3.2 结果数据

由上面的训练曲线可见，所有模型的训练损失函数都是收敛的，但验证损失和测试损失先降后升，说明模型训练出现了比较明显的过拟合情况，可能是训练数据集规模较小导致的。
取所有模型在测试集上的最佳表现，可列出如下表格

Model	CNN	LSTM	GRU	MLP	BERT
Test Accuracy	0.85637	0.85366	0.84824	0.85908	0.86179

Model	CNN	LSTM	GRU	MLP	BERT
Test F1 Score	0.85626	0.85352	0.84820	0.85899	0.86097

4. 调参效果

模型参数配置在 `config.py` 文件中，训练参数也可在运行 `main.py` 时以命令行参数的方式提供，以下调参均以 CNN 模型为例进行。

4.1 调整Batch Size

调整模型训练时使用的 `batch_size`，分别取值为 `32, 48, 64, 72, 96, 128`，监测模型的训练损失、验证损失和在测试集上的准确率，得到如下图所示的结果

从上图可以看出，损失曲线的收敛速度基本与 `batch_size` 的大小成反比，这可能是因为 `batch_size` 越大，每次梯度估计基于的样本越多，估计得到的梯度更加稳定，波动较小，使得模型训练相对稳定。

但是结合测试准确率来看，`batch_size` 也并非越大越好，图中 `batch_size` 取值为 `48` 的模型也出现了很好的性能表现（最高得分 0.86179，为组中最高），这可能是因为 `batch_size` 较小时梯度更新更加频繁，容易让模型跳出局部最优解，获得更好的泛化能力，从而在测试集合上有更好的表现，而 `batch_size` 较大可能会使模型陷入局部最优解，导致表现不佳。

总体上来看，`batch_size` 适中的模型普遍有比较良好的准确率表现，因此在实际选择 `batch_size` 时，应该尽可能选择适中的 `batch_size`，以获得良好的效果。

4.2 调整 Learning Rate

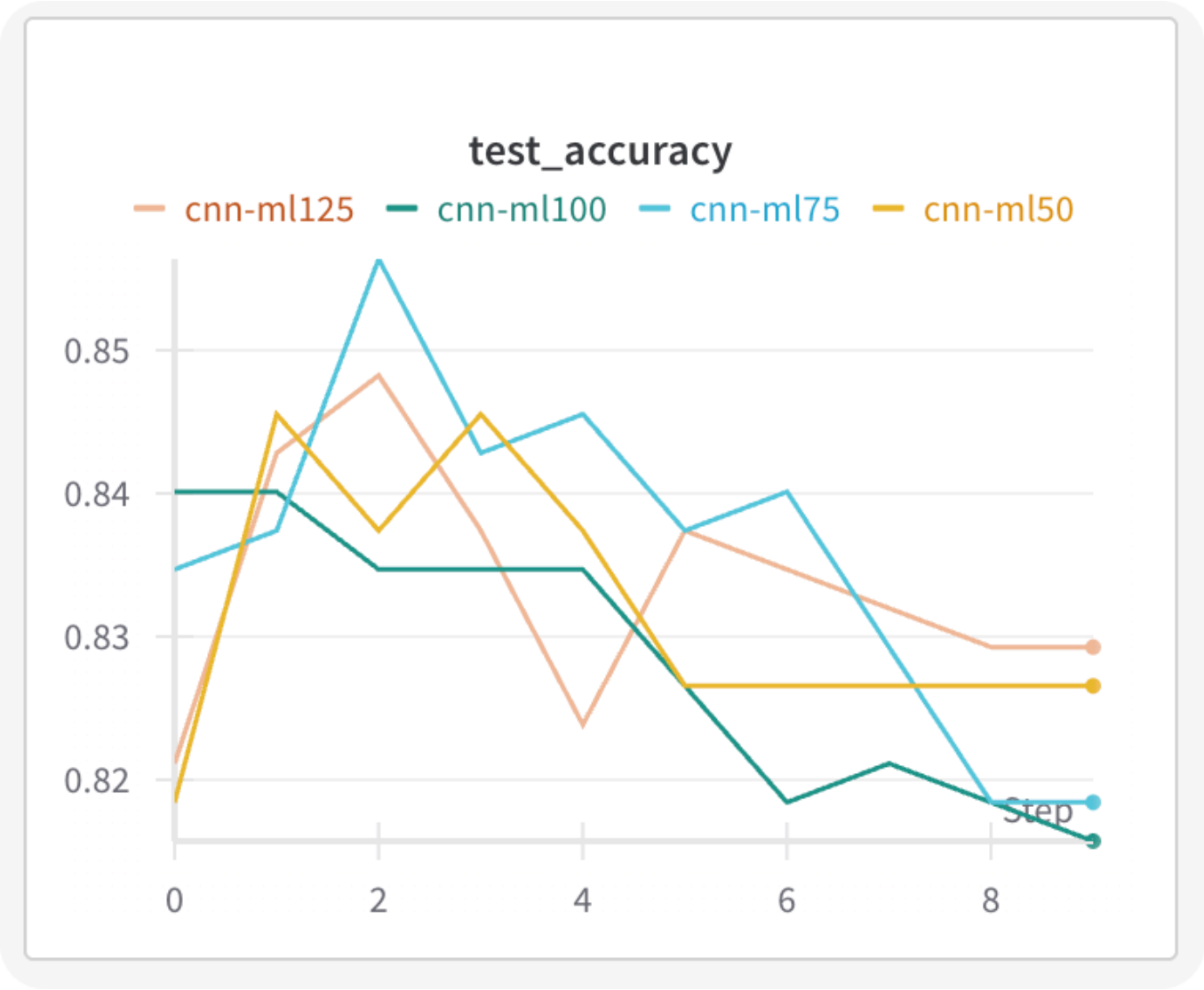
调整模型训练的初始学习率 `learning_rate`，分别取值为 `0.01, 0.001, 0.0001, 0.00001`，监测模型的训练损失和在测试集上的准确率，得到如下图所示的结果

* 实际上本次还测试了 `learning_rate=0.1` 时的训练效果，但此时训练曲线波动极大，不能收敛，准确率也非常低，因此并未展示。

从以上结果可以看出，学习率大小对模型训练的影响非常大。学习率过大可能使每次梯度更新过大，导致损失函数无法收敛，模型拟合效果极差；学习率过小可能使每次梯度更新过小，损失函数收敛速度较慢，需要更多的迭代次数才能获得较好的拟合效果。根据实验结果，`0.001` 是比较好的初始学习率选择。

4.3 调整 Max Length

调整模型数据处理时对输入序列进行截断的长度 `max_seq_length`，分别取值为 `50,75,100,125`（其中 125 已经超出几乎所有数据的序列长度），监测模型在测试集上的准确率，得到如下图所示的结果（见下页）



从以上结果可以看出，`max_seq_length=75` 时模型的表现最好，这可能是因为汉语的表达有时会带有铺垫，过早截断语句可能会丢失一些截断位置后方的重要情感信息，导致情感判断失误，所以 `max_seq_length=50` 时模型表现略差；而汉语中很少出现极长的语句，重要情感信息也往往不会只出现的语句末尾，当截断长度足以包括大部分汉语语句中的重要情感信息时，再增大截断长度只会增加模型的计算复杂度，并不会取得更好的表现。

5. 模型比较

根据 `3.1 训练曲线` 和 `3.2 结果数据` 中列出的模型性能得分，可以发现：

1. 作为 baseline 的 MLP 模型准确率和 f1-score 得分最高，结合训练曲线可以看出 MLP 模型的训练损失收敛速度最慢，虽然也出现了过拟合的情况，但是验证损失和测试损失低于其他模型，这可能是因为 MLP 网络的参数量较少，使得其过拟合情况较轻，泛化能力较强，因此在测试集上获得了较好的输出。
2. CNN 和 LSTM 模型得分相近，都与 MLP 相差不大，这是因为 CNN 通过卷积和池化操作能够很好地提取输入序列中的局部特征，对于分类任务有比较好的效果，而 LSTM 能够捕捉序列中的时间依赖关系，在自然语言处理工作上有一定的优势。CNN 的得分略高于 LSTM，从损失曲线可以看出 CNN 的过拟合情况要轻于 LSTM，得到这样的结果是符合常理的。

3. GRU 在以上模型中得分最低，GRU 本身与 LSTM 类似，但参数更少，可能使得它对文本特征的捕捉能力减弱，因此得分略低于 LSTM 和其他模型，但同时，参数更少也使得 GRU 的计算效率更高，训练速度有一定的提升。
4. 基于 Transformer 的 BERT 模型毫无疑问是所有模型中效果最好的，有着最高的准确率和 F-Score 得分，这是因为 BERT 在理解文本语义上有良好的效果，比较适用于此次文本情感分类任务。从训练曲线上还可以看出，相比于其他模型，BERT 模型的训练曲线比较平滑，可见具有成熟架构的 BERT 模型在训练时有很好的稳定性。

6. 问题思考

6.1 问题 1

在模型效果不再提升或提升十分缓慢，即训练损失曲线几乎不再下降，而验证集上的正确率几乎不再提升甚至开始下降时，应该停止训练。
在我的实现中，我发现除 BERT 模型外，其余模型在 10 个 epoch 以内都可以达到比较好的效果，而 BERT 模型可以在 20 个 epoch 内达到比较好的效果，继续训练模型性能不增反降，因此我最终使用固定迭代次数的方法，将 BERT 模型的迭代次数设为 20，其余模型迭代次数设为 10。

固定迭代次数

- 优点是简单、便于实现，不需要单独监测验证集上的性能来决定是否停止训练。
- 缺点是难以确定迭代次数，有可能因为迭代次数过少导致模型学习不完善，或因迭代次数过多导致模型过拟合严重，导致模型的效果不佳。

通过验证集调整

- 优点是灵活性强，能够根据模型在验证集上的表现来决定是否停止训练，既能让模型得到充分的学习，也可以在过拟合发生时及时止损，终止训练，得到效果较好的模型。
- 缺点是每次迭代都需要单独计算模型在验证集上的性能表现，提高了计算开销，减慢训练的速度。

6.2 问题 2

在模型设计上，我只使用实验要求中提供的词向量对嵌入矩阵进行初始化，而未显式对模型其余各层的权重进行初始化，因此其余各层的初始化均使用 PyTorch 框架的默认初始化方式。经查阅资料得知，PyTorch 的默认初始化方式为 Kaiming 均匀分布初始化，即使用公式 $weight \sim U(-\sqrt{k}, \sqrt{k})$ 对权重进行初始化。
对于其他初始化方式，通过查阅资料，我得到如下理解：

零均值初始化

将网络的权重设置为 0 均值，即权重的整体期望为 0，有助于梯度对称传播，避免偏向某一方向。当激活函数以 0 为中心时，零均值初始化效果较好，往往可适用于所有的模型。

高斯分布初始化

从高斯分布（正态分布）$N(0, \sigma^2)$ 中采样参数值，其中均值为 0，标准差 σ 可以根据输出维度设定，这有利于保持前向传播过程中激活值方差稳定，避免梯度消失或梯度爆炸，从而使网络快速收敛，因此往往用于 MLP、CNN、RNN 模型。

正交初始化

初始化时生成一个正交矩阵作为权重，该矩阵往往通过对某矩阵进行 QR 分解得到，这样可以保证前向传播和反向传播过程中信号的幅度大小不变，便于梯度稳定传播，因此往往用于深层网络或 RNN 模型。

6.3 问题 3

防止过拟合的方法主要有以下几种：

数据集扩充

一方面，可以收集更多的训练样本，增加数据的多样性，有助于模型学习一般性特征；另一方面，在难以增加样本数量的情况下，也可以对现有样本进行处理，进行变换、裁切等操作，产生更多的训练数据，也能达到数据集扩充的效果。

正则化

通过在损失函数中添加惩罚项，使得模型趋向于更为简单的解，进而降低模型的复杂度，提高模型的泛化能力。

Dropout

在训练过程中，以一定的概率将一部分神经元的输出置为 0，从而减少神经元之间的依赖关系，降低模型的复杂度，增强泛化能力。

早停机制

训练过程中监测模型的 Loss 值和在验证集上的性能表现，当性能不再提升（或开始下降）时，提前停止模型训练，从而获得没有发生明显过拟合的模型。

6.4 问题 4

结合查阅的资料，我得出如下理解：

CNN

- 优点：擅长提取局部特征，可拓展性强，容易通过添加不同的卷积层、池化层来构建更深层次的模型，应对更复杂的文本分类任务，且支持并行计算；
- 缺点：不具有时间上的局部建模能力，不能很好地处理序列数据，也不便于处理变长数据，在固定数据长度时可能会丢失语义信息。

RNN

- 优点：具有记忆特性，可以处理变长数据，在具有序列特征的任务（比如自然语言处理）上有较好的效果；
- 缺点：计算是顺序进行的，每个时间步的计算依赖于前一个时间步的结果，导致难以并行计算，且当时间步较多时容易出现梯度消失或梯度爆炸问题。

MLP

- 优点：模型简单，且高度支持并行计算，计算速度快；
- 缺点：模型参数量小时效果较差，而参数量大时又容易发生过拟合，设计效果较好的模型相对困难。

7. 心得体会

本次实验使我获得了巨大的收获，让我对深度学习模型的理解大幅加深。在课上学习 CNN、RNN 等模型仅是了解了其工作的原理，对其训练方式理解并不是很深刻，而本次试验恰好通过实践的方式让我清楚地理解了模型训练的过程，使我对训练的过程、参数作用、模型差别等方面都有了更深刻的理解。

在实验中，通过对比实现 CNN、RNN、MLP 等模型，我切实认识到了不同结构的模型可能适用于不同类别的任务，在选择模型时应该先考虑要完成的任务类型，而不是随意选择。

通过调参，我理解了 `batch_size`、`learning_rate` 等参数在训练过程中所起到的不同作用，并且对于参数应该如何取值有了一定的了解。

工具使用上，本次实验使用 PyTorch 框架来实现模型训练，这使我对深度学习框架的功能和使用方式有了基本的了解，也感受到现有的深度学习框架为模型训练提供的极大便利。

本次实验中最大的遗憾是，由于时间有限，仅仅实现了一个可以“跑通”的 BERT 模型，却未能更加细致深入地研究其原理，感觉对 BERT 模型的理解还不够透彻，希望未来的学习过程中可以进一步探究。

总之，本次实验历时多日，过程中不乏许多尝试失败的遗憾，也充满通过一次次调整使得模型分数一点点提高的惊喜，在遗憾与惊喜中所学到的知识对我来说更是宝贵的财富。