第三次大作业：文本情感分类任务

计 71 张程远

2017011429

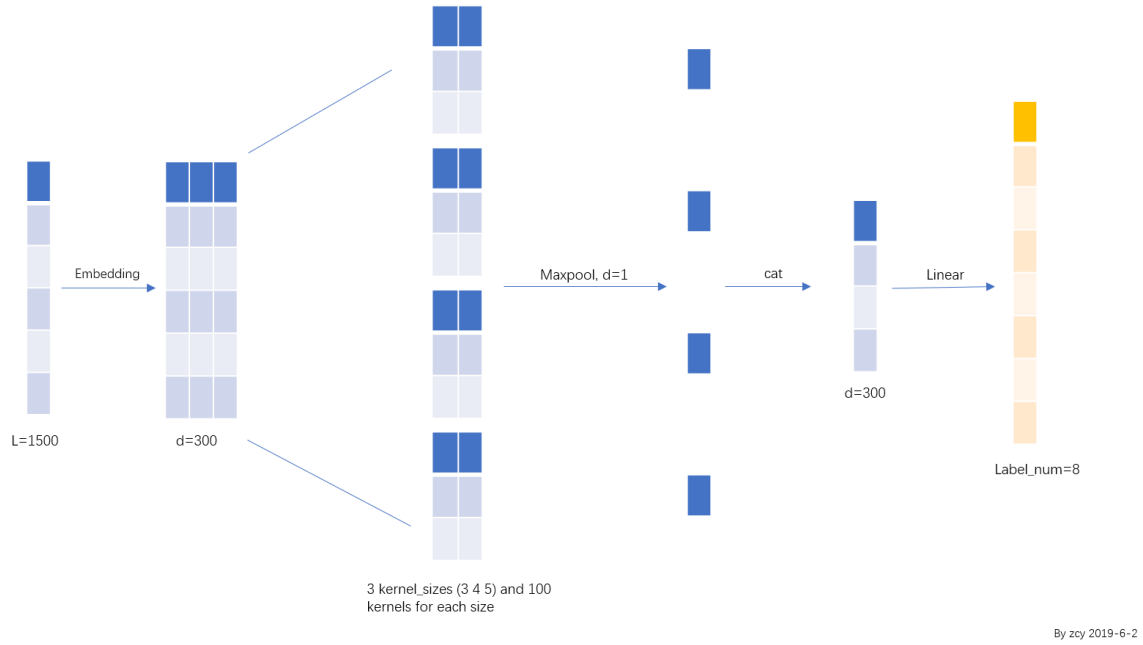
目录

1 问题描述

本次实验要求使用 CNN 和 RNN 两种模型，实现新闻文本的情感分类任务。每条新闻文本共有 8 个情感标签，任务要求模型在学习训练集给定的新闻和标签数据之后，能够对测试集中的新闻情感做出正确的判断，并计算相应的评估参数。文本对应的词向量已经在任务中给出。

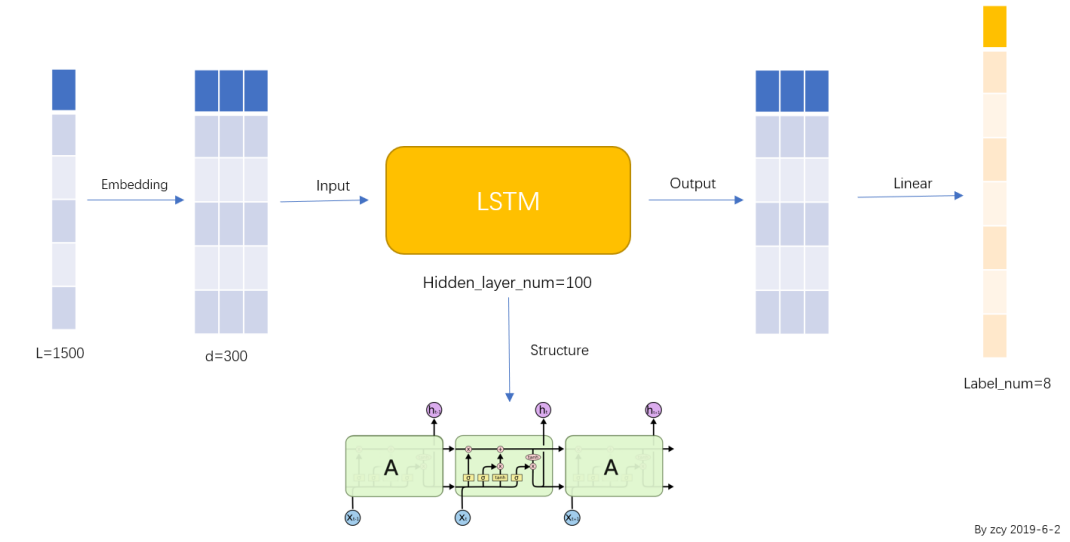
2 实验原理

2.1 CNN 模型



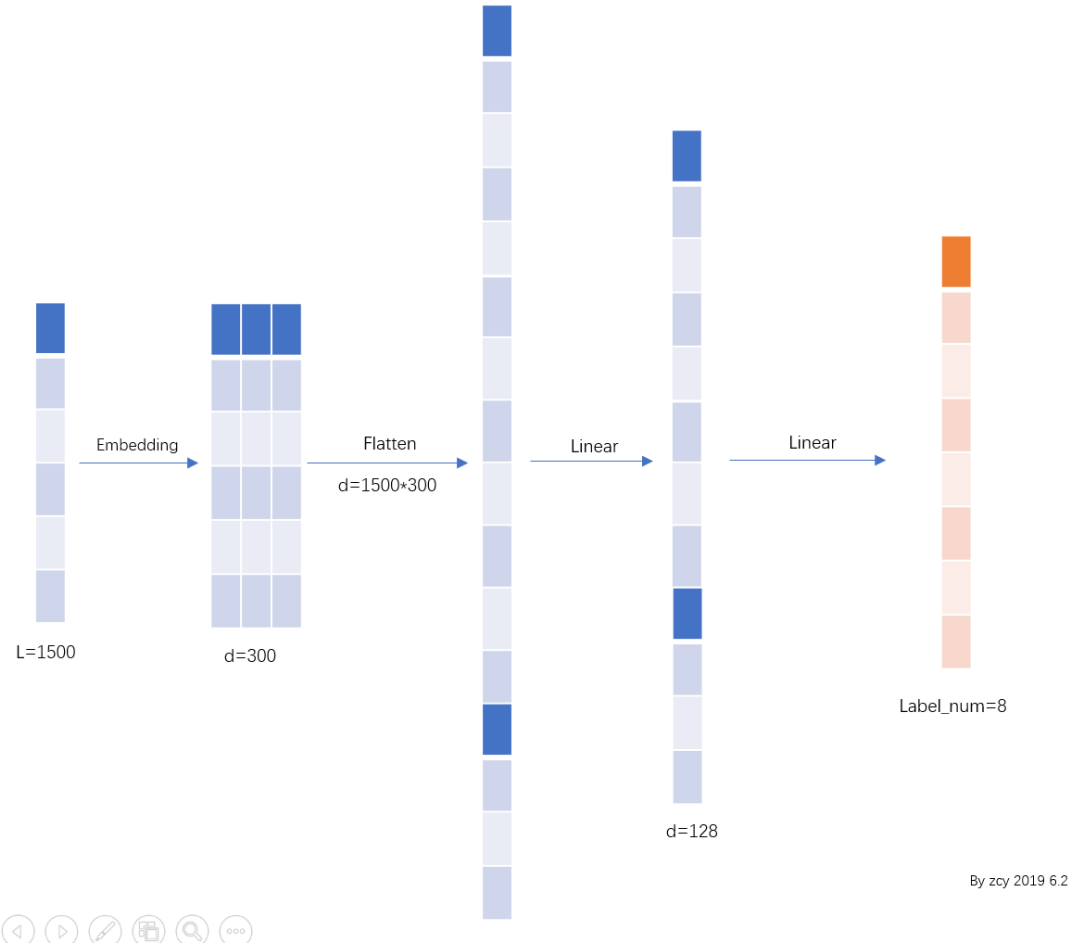
流程图如上图所示。文段首先被编码为 index，然后进行 embedding 转化为输入矩阵接下来进入为一层卷积层，卷积层采用了 3 种 kernel_size，每种 100 个 kernel 进行处理，然后分别池化，最终拼成一个向量，其维数为种数*每种 kernel 的个数。最后通过一层全连接层，产生 8 种标签的分类概率。

2.2 RNN 模型



文本经过与 CNN 相同的编码，进入到 LSTM 层中。这里使用的 LSTM 是 Keras 框架自带的 LSTM 实现，参数 hidden_num_layer 设置为 100。取 LSTM 最后输出的状态，经过一层全连接层得到 8 种标签的分类概率。

2.3 MLP 模型



经过与 CNN 相同的编码，首先拉平为 1500*300 长度的一维向量，再经过 128 维和 8 维的全连接层得到输出。

注：以上三个模型使用的 Loss 均为交叉熵。

3 实验结果

下表是对不同模型及参数的实验效果进行的比对。

Model	**Dropout /LR**	Accuracy	F-Score	Corr.	Epoch
CNN	0.5/0.001	60.01%	21.07%	57.62%	20
RNN	0.5/0.001	57.27%	20.33%	57.46%	20
RNN	0.2/0.001	56.24%	21.00%	57.96%	20
RNN	0.5/0.1	47.76%	08.08%	43.03%	15
MLP	0.5/0.001	55.43%	21.33%	55.91%	20

注：LR 即学习率（Learning Rate）；Batch size 均为 128；由于我实现的 CNN 运行速度较慢，所以主要采用 RNN 进行参数测试；测试集数据处理采用了 argmax，即相同数量下取最靠前的标签用作计算。

首先比较不同参数的执行效果。我们看到，相同 LR 下，Dropout 取 0.5 相比于取 0.2 时获得正确率要好；相同 Dropout 下，LR 取 1/1000 比取 1/10 要好，取 1/10 的模型几乎没有学到什么东西——因为本次任务中全部输出“3”就可以获得 47.76%的正确率。分析其原因，我认为还是因为数据量小、训练集与测试集关联性不强导致的，这需要模型在学习时不能太沉迷于训练集，要有较高的 Dropout 和较低的 LR，才能让模型具有较好的泛化能力。

其次比较不同模型的效果。实验中的 MLP 模型实现非常简单，只有两层全连接层，但效果却出乎我意料，正确率不算低，而且在 F-Score 参数上甚至超过了其他模型。分析其原因，可能主要在于本次实验数据量小，学习的特征数也较小，复杂深度学习网络相比于简单的网络，无法体现出其优势。CNN 与 RNN 的差距不大，CNN 的正确率稍高，可能是因为网络结构稍复杂，采用了多种大小的卷积核进行处理的缘故。

4 几个问题

4.1 内存超限

由于 Python 的文件读写需要耗费较高的内存（约为文件大小的三倍），而我最初的想

法是将文章全部转换为词向量并将结果保存下来，而且文章长度设定值为 1500（只有一篇长度超过 1500 词，保险起见设置为 1500），所以处理时一直报 Memory Error。后来我了解到 Torch 可以采用 Embedding 方法导入训练好的词向量，这样只需要把文章内容输入模型即可，大大减少了内存的消耗。不过即便如此，运行 Torch 消耗的内存量还是占到了设备内存的 90%。

4.2 Validation 与 Test

在使用 Keras 框架写 RNN 的时候，fit 函数需要提供 validation set 作为模型的衡量指标。最开始我使用了给定的 test set 作为验证集，后来通过查询资料发现，虽然验证集不用于更新网络内部的权重，但是使用测试集作为训练集仍然会导致信息的泄漏，使得模型对于测试输入更加熟悉，从而一定程度上影响实验结果[1]。因此最后我划分了 10%的训练数据作为验证。

4.3 Loss 变为 nan

最开始训练 RNN 的时候 loss 一直输出 nan，查阅资料发现有两种情况：第一种情况是 loss 过大无法训练，另外一种是出现了$0 * \log 0$的情况。后来将 Dense 层的 ReLU 改为 sigmoid 函数问题便得以解决。[2]

5 思考题

（1）实验什么时候停止是最合适的？陈述实现方式，并分析不同方法的优缺点。

答：停止方式有三种：一是固定迭代次数后停止，二是在验证集的 loss 最小时停止，三是在验证集 acc 最大时停止。我在不同模型里采用了不同的实现方式：CNN 中，由于模型的 save 和 load 都需要耗费内存，而使用 torch 运行 CNN 又使得可用内存十分有限，因此我采用了固定迭代次数停止，即共 20 个 epoch，每个 epoch 后都将模型保存下来，再统一加载 20 个模型进行测试；RNN 则采用了 loss 最小时停止，即只保存 valid loss 最小时的模型参数。

第一种实现方法的优点在于，可以确定我所拿到的所有模型中效果最好的是哪一个，只需分别测试便知；缺点在于运行和测试的耗时较长，并且保存多个模型需要消耗大量资源；第二种实现方法的优点在于耗时较短，并且能在验证集与测试集数据分布较一致的时候取得很好的效果，缺点在于，这样得到的模型只是在验证集上的最优解，如果验证集分布与测试

[1] 参见博客 https://www.jianshu.com/p/0c7af5fbcf72。
[2] 参考博客：https://www.jianshu.com/p/79ea75c47004

集数据差距较大，那就会导致过拟合的现象。本实验中我使用 RNN 做了测试，dropout 取 0.2，学习率设为 0.001，验证集取训练集的随机 10%，并取了 loss min 附近的 2 个 epoch 进行测试，结果发现所得的正确率差距不大，均在 53%-56%之间，效果可以接受。

（2）实验参数初始化选择是怎么做的？不同参数适合哪些地方？

答：两种模型我均没有特别设置初始化参数，都是用的自带参数——CNN 中，Torch 的 linear 和 Conv2D 都采用了$[-\text{limit}, \text{limit}]$的均匀分布（Xavier 初始化），其中$\text{limit} = 1/\sqrt{N}$，N为输入单元的数量。而 Keras 的 LSTM 则是采用了正交初始化的方法。

正交初始化常用于 RNN 模型，因为考虑代价对于初始参数矩阵的分量求导，每一维求导的结果都含有分量的幂次。如果矩阵特征值大于 1，则 loss 难以收敛；小于 1 则导致梯度失活。因此采用正交矩阵，让特征值为 1，能够避免梯度爆炸或消失的现象。采用 Xavier 方法可以保持输入和输出一致，这能够避免输出值都趋于 0，是一种较为通用的方法。而对于使用 ReLU 激活函数的模型而言，Xavier 初始化将导致一半的数据梯度为 0，会导致参数方差产生波动，所以高斯分布法是更好的选择，它将方差$\text{Var(W)} = \frac{2}{\text{N(in)}}$放大了一倍，以此保持方差的相对平稳。

（3）有什么方法能够防止深度学习陷入过拟合？

答：有 3 种方法。第一种是 early stop，也就是设置验证集，在训练到验证集的 loss 反弹时就停止学习（因为 valid loss 可以衡量模型预测没见过的数据的能力）；第二种是设置 dropout，即训练时每次随机导致一部分神经元失效，使得模型每次都与之前的步骤“不一样”，从而提高模型的应变能力。第三种则是扩展数据集，因为本次实验的数据集较小，所以在 15 个 epoch 以后过拟合的现象就比较明显了，不过本次实验无法扩展数据集。前两种方法在实现的模型中有体现，early stop 用于 RNN，dropout 设置值为 0.5。

（4）分析 CNN、RNN 与 MLP 的优缺点。

答：CNN 与 RNN 原理不同，两者应该适应不同类型的问题。RNN 比较注重序列信息，LSTM 的功能可以让 RNN 感受到词语上下文之间的联系，所以 RNN 应该更加适合根据文段预测下文的任务；CNN 则注重词语本身的分类，像本次实验中文本的情感常常由几个词就可以定性，CNN 检测到这些词就可以做出正确的结果。这也从另一个角度解释了 CNN 的效果好于 RNN。MLP 同样没有使用上下文信息，不过它参数少、结构简单、训练较快，得到的结果也不是很差，与此相比 RNN 和 CNN 消耗资源较多，需要的时间也较长。

除此之外，RNN 在本次实验的表现似乎说明其更容易过拟合——CNN 通常在 15 个

epoch 内其 valid loss 不再下降，MLP 则是 20 个，而 RNN 一般在 10 个 epoch 内。这可能还是由于 LSTM 本身更注重序列信息的特点导致的，因为学习到这样的信息其实用处不大，过于依赖这些信息反而会导致过拟合。

6 心得体会&Honor Code

这次作业应该是我投入时间和精力最多的作业了，从零学习了 Torch 和 Keras 的使用，了解了很多深度学习的基本原理，对于我这种只是粗略了解过深度学习却从未实践过的学生来说收获非常大，而且特别让我意料之外是改换了 Keras 框架的 RNN 取得了较好的效果，让我倍受鼓舞。Keras 确实强大，代码量小，使用方便，运行时占用内存也很小，属于封装较好的模型，我以后实践深度学习算法可能会更多选用类似的高级框架。

非常感谢计 76 的周子栋同学，他告诉了我可以在 Torch 中使用 embedding 的方法，让我解决了内存危机；还要感谢 75 的罗峻骁和 74 的陈果，在 CNN 有奇怪 bug 的时候帮助我检查了模型，让我最终找出了 bug；感谢 73 的赵亮，给我讲解了 Keras 的很多知识，让我获益匪浅，在 Torch 版 RNN 过慢且效果不佳的情况下很快搭出了 Keras 版的 RNN。真的非常感谢以上这些同学，也感谢助教的答疑解惑!