基于深度学习的文本分类

日期：2023 年 4 月 9 日

摘　要

本次任务我使用 PyTorch，实现了 TextCNN 和 TextRNN 两种模型，使用论文 [1] 中方法以及参数实现 TextCNN，使用 RNN、LSTM、Bi-LSTM 实现模型 TextRNN，并通过实验探究模型一些超参数。后续将训练得到的 4 个模型在 Kaggle 上进行预测评分，结果并不理想，最后就此进行了后续的讨论与分析。

关键词： CNN，RNN，文本分类

1　问题简述

本次任务要求使用 pytorch 重写任务 2，实现 CNN、RNN 的文本分类。其中 embedding 部分需要使用随机初始化以及 glove 与训练 embedding 两种方法。数据来自 Kaggle 比赛Sentiment Analysis on Movie Reviews，比赛提供了烂番茄电影评价数据集，且要求在五个尺度上标注短语：消极、有点消极、中性、有点积极、积极。其中句子否定、讽刺、简洁、语言歧义等障碍使这项任务极具挑战性。

2　模型介绍

2.1　整体实现

本次任务整体可分为模型训练与预测验证两部分，模型训练部分有 data_prepare.py、text_model.py 等，其余的 shell 文件等均为对训练参数的探究。验证和预测部分则类似训练阶段的脚本，仅仅少了 param.py 文件且使用 main.ipynb 作为主文件，整体实现架构如图 1 所示。模型训练阶段，在 data_prepare.py 中实现

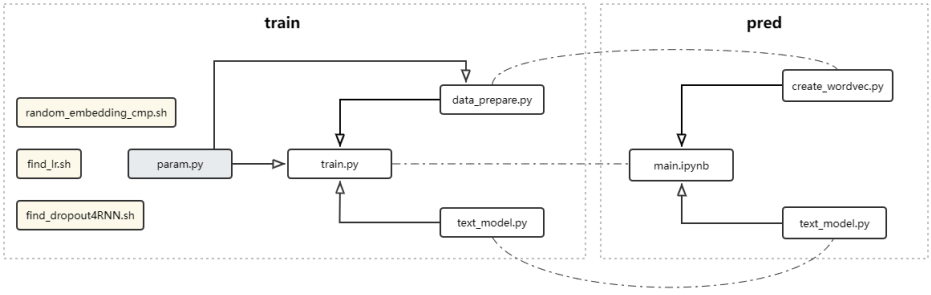

图 1: 实现框架

数据读取、词汇表构造、dataLoader 封装等。在 text_model.py 中实现 TextCNN、TextRNN 两个类，其中 TextRNN 中包括实现 RNN、LSTM 以及 Bi-LSTM 实现的模型 (根据传入的参数选择不同模型)。train.py 脚本则为训练的主脚本，通过 param.py 接受来自终端的输入后在 train.py 中赋值选择不同的参数。图中箭头从被调用脚本出发，指向调用脚本。左侧的 shell 脚本分别为探究随机 embedding 效果、寻找最优学习

率与对于 RNN 模型[1]寻找最优 dropout 的脚本。模型验证和预测阶段则使用与训练阶段平行的三个脚本实现，三个脚本实现类似的功能，仅有部分修改。main.ipynb 中使用 Kaggle 提供的 test.tsv 文件作为测试集，在 Kaggle 官方提交进行验证打分，此外还从 train.tsv 中各分类抽取 2000 条数据并用混淆矩阵验证。

2.2 Embedding

该部分使用 glove.42B.300 词向量，考虑到模型参数量与词汇表维度有关，直接使用 glove.42B.300d.txt.pt 模型在后续 CNN、RNN 等处需要的模型规模较大，训练需要数据集较多。对此根据同时在 glove 词汇表和 train.tsv 出现的单词[2]、代表未知单词的“unk”以及空白填充单词“<pad>”组成新的词汇表以及词向量。

在词向量模型的选择方面，比较了 glove.42B.300 比 glove.6B.50d，在 train.tsv 拆分的验证集上提升不大，且在 glove.42B.300 模型上计算资源已经消耗较大，因此未继续使用更大的词向量模型。

2.3 TextCNN

该模型架构以及超参数设置均参照论文 [1] 中最优模型，可训练的 embeddign 层使用 glove 词向量初始化，卷积层设置卷积核为 3，4，5 的三个平行卷积核，后续通过 max-pooling 化为一维张量，然后使用 0.5 抛弃率的 dropout 层连接全连接分类。

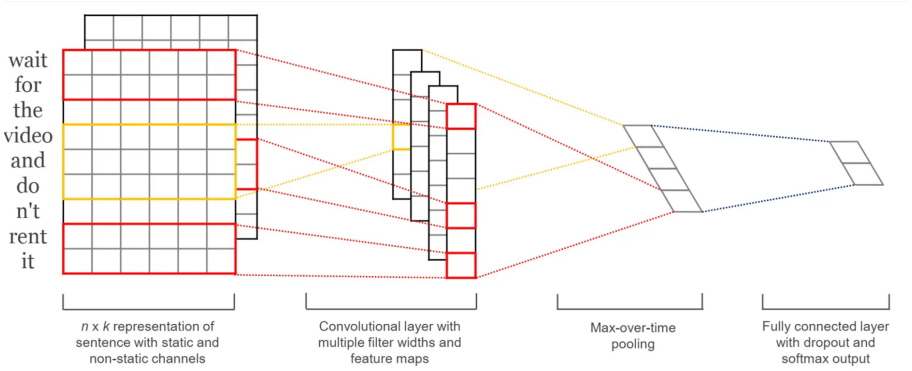

图 2: TextCNN 模型架构

2.4 RNN

模型整体由可训练的 embedding，循环网络层以及分类层组成。循环网络层使用 RNN、LSTM、Bi-LSTM 三种模型实现，通过终端输入的参数“rnn_type”选择。RNN 和 LSTM 形成的网络使用蕴含句子中所有单词信息的隐向量 h_n 作为句子特征。Bi-LSTM 则使用拼接了正向和逆向的最后一个单词隐向量 h_n 得到的 $output_n$ 作为句子特征。模型架构如图 3 所示。

3 实验

3.1 探究学习率

在此部分，使用 find_lr.sh 脚本对于 CNN、RNN、LSTM、Bi-LSTM 分别构建的文本分类模型，在 1e-7 到 1e-1 之间首先进行了较大跨度的探索，后续进行细化寻找，在 train.tsv 数据集上以 8:2 拆分的训练集

[1]对于 TextCNN 模型采取论文 [1] 中的 0.5

[2]此处因为不需要评估词向量的效果，故验证集中出现的单词也用来构造词汇表

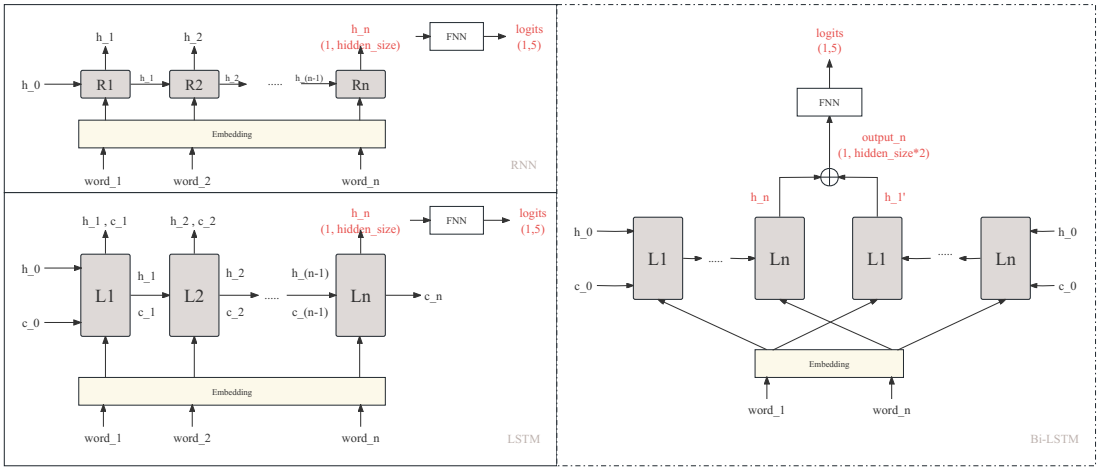

图 3: 循环网络模型架构

和验证集合上得到了如表 1 所示 (仅展示较优的数据)。

表 1: 不同学习率对模型影响

学习率	模型	最佳 epoch	最佳准确率 (%)
0.0001 0.0005	TextCNN	19 4	67.23 67.38%
0.0001 0.00004	TextRnn$_{RNN}$	48 50	65.32% 65.55%
0.0001 0.00004	TextRnn$_{LSTM}$	42 16	67.09% 67.46%
0.0001 0.00004	TextRnn$_{Bi-LSTM}$	13 49	67.24% 67.45%

3.2 随机 embedding 与使用 glove 初始化

该部分，使用 random_embedding_cmp.sh 脚本，仅比较 TextCNN 模型，在学习率为 0.0005，一共训练 20 个 epoch，设置 batch_size 为 128 下训练使用 GloVe 赋值的词向量以及随机正态分布初始化的词向量的两个模型，在拆分的训练集和验证集上得到了如表 2 所示的结果。发现在词向量可以更新的情况下使用 GloVe 初始化词向量能相对随机词向量有较大的提升。

表 2: 不同词向量初始化方法对模型影响

学习率	词向量	最佳 epoch	最佳准确率 (%)
0.0005	GloVe	4	67.38%
0.0005	随机	7	65.11%

3.3 探究循环网络模型的 dropout

该部分使用 find_dropout4RNN.sh 脚本，对于 RNN、LSTM、Bi-LSTM 三个模型分别构成的 TextRNN 从 dropout 为 0.1 到 0.5 遍历。发现对于 $TextRNN_{RNN}$，使用 dropout 为 0.4 能使其准确率增加到 65.65%，而对于 $TextRNN_{LSTM}$ 和 $TextRNN_{Bi-LSTM}$，使用 dropout 反而导致准确率下降，因此后续也仅对 $TextRNN_{RNN}$ 添加 dropout 为 0.4 的 dropout 层。

3.4 探究循环网络模型的 hidden_size

该部分使用 find_hidden_size4RNN.sh 脚本，对于 RNN、LSTM、Bi-LSTM 三个模型分别构成的 TextRNN 从 hidden_size 从 64，128 到 256 遍历，发现对于 $TextRNN_{RNN}$ 增大 hidden_size 准确率可以有较小提升，从 64.39% 提升到了 65.85%。对于 $TextRNN_{LSTM}$ 则是在 hidden_size 在 192 时达到最大的 67.13%，对于 $TextRNN_{Bi-LSTM}$ 则是在 192 和 256 时相差无几，为 66.95% 和 66.96%。

4 模型表现

最终在验证集上 TextCNN 模型在 0.0004 的学习率上在第 6 个 epoch 取得最大准确率 67.73%，而 TextRNN 模型则均使用 0.0001 的学习率，hidden_size 为 192，对 $TextRNN_{RNN}$ 设置 0.4 的 dropout 层。$TextRNN_{RNN}$、$TextRNN_{LSTM}$ 和 $TextRNN_{Bi-LSTM}$ 分别达到 65.88%、67.1% 与 67.22% 的准确率。后续使用模型对 test.tsv 文件展开预测，并提交 Kaggle，4 个模型分别得到 0.37464、0.37888、0.40863 与 0.39549 的得分，与上次使用机器学习实现文本分类结果相仿。从 train.tsv 中每个类抽取 2000 条数据进行预测并计算混淆矩阵，可视化后发现仍然存在之前的问题——模型“故意”将大部分数据预测为训练数据中最多的 neutral。

5 后续探究

对于本次任务，在观察到 4 个模型都出现之前的“故意”加大对 neutral 分类的预测后，尝试尽可能平衡数据样本。第一次尝试是在训练数据中随机截取 30000 条 neutral 分类样本替换原数据中该分类的数据，在 TextCNN 模型训练出来后，在 Kaggle 上评分为 0.37336，较之前的 0.37464 偏低。且混淆矩阵出现“故意”将大部分数据预测为训练数据中最多的 neutral、somewhat negative 和 somewhat positive 三类，与之前的结果类似。因此再次尝试做到 5 个类别的训练数据平均，即对每个类别取 7000 条数据[3]。最后训练结果在 Kaggle 上评分 0.25621 果然因为极少的数据量出现了训练欠佳。混淆矩阵虽没了之前的问题，但这次整体也不理想，预测结果出乎意料得集中在了 somewhat negative 以及少数的 neutral 上。

整体来看，要对于本任务中的数据有较好的训练以及预测效果，或许不应该减少训练数据数量以改变数据分布，一个可行的方法就是改变训练中模型的偏好，从损失函数入手，不能仅使用交叉熵损失，或许可以通过添加额外惩罚系数鼓励模型平等地对待每个分类的数据。

参考文献

[1] Yoon Kim. *Convolutional Neural Networks for Sentence Classification*. 2014. arXiv: 1408.5882 [cs.CL].

[3] 数据量最少的一类样本有 7015 条数据