自然语言通顺判定实验报告

2019 年 1 月 20 日

1 任务分析

自然语言通顺判定是自然语言处理中常见的一项任务，使用的范围非常广。在本次试验中，我们的任务是进行自然语言通顺与否的判定，即给定一个中文句子，要求判定所给的句子是否通顺。该任务是一个二分类任务，使用 F1-score 作为评测指标。

2 数据集

本次实验使用的数据集分为训练集、额外训练集和测试集。数据的描述如表 1所示。

表 1: 数据集描述

数据集	语句条数	最大长度	正样例数	负样例数	正负比
训练集	146950	790	135931	11019	12.34:1
额外训练集	22040	510	11020	11020	1:1
测试集	19990	1354	-	-	-

3 模型简介

3.1 词向量表示

基于神经网络的词的分布表示一般称为词向量、词嵌入 (word embedding)。神经网络词向量表示技术通过神经网络技术对上下文，以及上下文与目标词之间的关系进行建模。由于神经网络较为灵活，这类方法的最大优势在于可以表示复杂的上下文，同时可以规避传统方法在词序信息增大时可能会遭遇的维度灾难问题。

在本次试验中我采用了 BERT [1] 在简体中文数据集上训练的词向量模型 (bert-base-chinese)[1]。BERT 模型的结构如图 1所示：

[1]https://github.com/huggingface/pytorch-pretrained-BERT

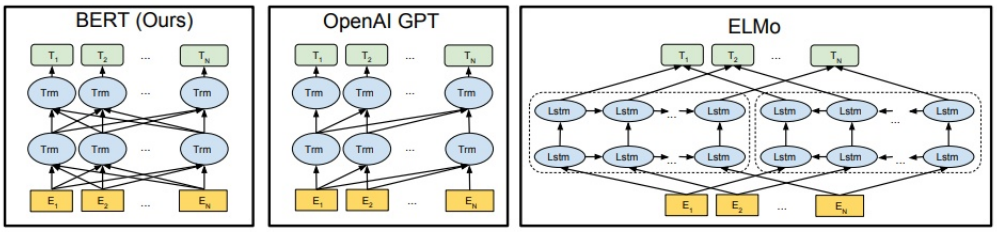

图 1: BERT 模型结构

BERT 的主要贡献在于：证明了双向预训练对语言表示的重要性。与之前使用的单向语言模型进行预训练不同，BERT 使用遮蔽语言模型来实现预训练的深度双向表示。此外论文表明，预先训练的表示免去了许多工程任务需要针对特定任务修改体系架构的需求，BERT 是第一个基于微调的表示模型，它在大量的句子级和 token 级任务上实现了最先进的性能，强于许多面向特定任务体系架构的系统。

3.2 分类模型

由于 BERT 模型的双向性，所以语句中第一个字符的词向量就包含了整个语句的语义信息。我采用该词向量表示（768 维）作为分类模型的输入，考虑到该任务是二分类问题，所以通过一个大小为 768×2 的全连接层将词向量映射到 2 维的概率向量，进而使用交叉熵损失函数作为训练深度神经网络的目标函数。交叉熵损失函数的定义如下：

$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right) \tag{3.1}$$

3.3 类别不平衡的处理

训练集中的正负样例比约为 $12:1$，所以直接进行训练会导致分类器出现严重的类别倾向，即分类器只要全部预测为正例就可以在训练集上获得较高的精度，但这样的模型具备的泛化性能是非常弱的，通常会存在较高的泛化误差，所以我们需要重点处理类别不平衡的问题。关于类别不平衡的问题，我尝试了两种方案：

(1) 在交叉熵损失函数中，为不同的类别设置不同大小的权重，具体地，假设正负样例的数目比是 $12:1$，则将正负例对应的权重设置为 $1:12$，即负例的数目虽然少，但是更新的力度大。修改权重后的交叉熵损失函数的定义如下：

$$\text{loss}(x, class) = weight[class]\left(-x[class] + \log\left(\sum_j \exp(x[j])\right)\right) \tag{3.2}$$

(2) 考虑到深度学习的难易程度，在过采样和降采样这两个对称的方案中，我选择了降采样的方案。具体地，假设正负样例的数目比是 $12:1$，则将正样例按 $1/12$ 的概率进行降采样，保证训练时正负样例的数目比是 $1:1$，这样做同时也减少了训练数据的数据量，可以一定程度上降低模型训练的难度。在实际的使用过程中，这种方案的效果更加显著。

4 优化方案

4.1 速度优化

从数据集分析可以看出语句的长度具有较大差异，为了节省显存的开销和降低训练的难度，通常会选择一个最大长度进行语句的截断和对齐（例如将最大长度设置为 100）。但是在一个 mini-batch 中的数据的最大语句长度通常远小于这个最大长度，所以这种方式依然会造成显存的浪费和不必要的训练时间开销。在训练速度的优化方面，我修改了 pyTorch 提供的 DataLoader 类中的 collate_fn 方法，动态地根据当前 mini-batch 中的数据来设置合适的截断长度。这一优化的效果如表 2所示，所使用的显卡型号是 TITAN Xp。

表 2: 速度优化结果

方案	运行速度
固定最大截断长度为 100	5.32 min/epoch
动态设置最大截断长度	2.73 min/epoch

4.2 精度优化

在精度的优化方面，我主要尝试了以下几个方面的工作：

(1) 手动清除训练集中存在的噪声，例如数据标记的错误。

(2) 设置交叉熵损失函数的类别权重。

(3) 进行降采样操作。

(4) 使用额外的训练集。

这些优化的效果如表 3所示。

表 3: 精度优化结果

方案	B 榜线上 F1 值
无优化操作	0.6973
手动清除噪声	0.7182
设置交叉熵损失函数的类别权重	0.7763
进行降采样操作	0.7935
使用额外的训练集	0.8092
调整降采样的比例	0.8167

5 项目结构

整个项目的结构如下所示：

```
/
├── data
│   ├── test.content.real.txt
│   ├── test.content.txt
│   ├── test.label.real.txt
│   ├── test_v3.txt
│   ├── train.txt
│   └── train_downsample.txt
├── preprocessed_data
│   ├── dev.json
│   ├── test.json
│   └── train.json
├── saved_model
├── data_loader.py
├── downsample.py
├── main.py
├── metrics.py
├── model.py
├── preprocess.py
├── preprocess_embedding.py
├── statistics.py
└── train.py
```

其中各个模块的内容或功能如下：

- data/*.txt: 未经过预处理的训练集、额外训练集和测试集；
- preprocessed_data/*.json: 进过预处理的训练集，验证集和测试集；
- data_loader.py: 数据读入和整理为神经网络输入的形式；
- main.py: 主运行脚本，提供本次试验所需的操作；
- metrics.py: 原始提供的评测模块；
- model.py: 本次实验的词向量模型和分类模型；
- preprocess.py: 数据预处理操作；
- preprocess_embedding.py: 数据预处理操作；
- statistics.py: 进行基本的统计操作；
- train.py: 模型的训练，验证和测试；

6 运行方式

本工程的运行方式较为复杂，首先需要配置安装以下的软硬件环境：

- python >= 3.6;
- pytorch 0.4.1;
- pytorch-pretrained-bert 0.3.0 ;
- 8000MB 显存以上的 GPU 一张，本次试验我使用的硬件是 TITAN Xp GPU 服务器；

具体的运行方式如下：

- 进入项目目录，执行 python downsample.py，生成降采样后的训练数据；
- 执行 python preprocess.py，将语句进行 BERT 模型所需的下标化；
- 执行 python main.py –gpu-devices 0，进行模型的训练（参数已经和最终 Leaderboard 提交版本一致，无需调整），这里 gpu 参数请根据实际的情况进行修改。训练结束后，最优的模型将被保存至 saved_model/1 文件夹下；
- 执行 python main.py –resume True –load-model saved_model/1/ckpt-epoch-2 –save-dir saved_model/1 –gpu-devices 0，进行模型的测试，这里 gpu 参数请根据实际的情况进行修改。测试结束后，测试文件将被保存至 saved_model/1 文件夹下，即为 Leaderboard 的最终提交版本；

7 运行实例及线上结果

模型训练过程中的运行实例如图 2所示，可以看到在训练的过程中，模型在训练集上的 Loss 在持续下降，而在验证集上的 F1 值先上升后下降。我将在验证集上取得最优表现的模型进行保存，并用于测试和线上提交。

```
INFO:root:Number of parameters: 102269186
INFO:root:Beginning training...
INFO:root:Epoch: 1 | Loss: 0.01649 | Precision: 0.69600 | Recall: 0.87000 | F1: 0.77333 | Time: 6.115
INFO:root:Saving checkpoint...
INFO:root:Checkpoint saved to saved_model/1/ckpt-epoch-1.
INFO:root:Epoch: 2 | Loss: 0.01281 | Precision: 0.76098 | Recall: 0.78000 | F1: 0.77037 | Time: 7.332
INFO:root:Saving checkpoint...
INFO:root:Checkpoint saved to saved_model/1/ckpt-epoch-2.
INFO:root:Epoch: 3 | Loss: 0.00842 | Precision: 0.79293 | Recall: 0.78500 | F1: 0.78894 | Time: 8.198
INFO:root:Saving checkpoint...
INFO:root:Checkpoint saved to saved_model/1/ckpt-epoch-3.
INFO:root:Epoch: 4 | Loss: 0.00502 | Precision: 0.79798 | Recall: 0.79000 | F1: 0.79397 | Time: 8.169
INFO:root:Saving checkpoint...
INFO:root:Checkpoint saved to saved_model/1/ckpt-epoch-4.
INFO:root:Epoch: 5 | Loss: 0.00300 | Precision: 0.79275 | Recall: 0.76500 | F1: 0.77863 | Time: 8.180
INFO:root:Saving checkpoint...
INFO:root:Checkpoint saved to saved_model/1/ckpt-epoch-5.
INFO:root:Epoch: 6 | Loss: 0.00194 | Precision: 0.79894 | Recall: 0.75500 | F1: 0.77635 | Time: 8.087
INFO:root:Saving checkpoint...
INFO:root:Checkpoint saved to saved_model/1/ckpt-epoch-6.
```

图 2: 线下表现情况

最终线上的结果如图 3所示，我最终的 F1 值为 0.816743，排名第 9 名。