# 机器翻译报告

## 一、任务描述

给定一个对话数据集，其中每段对话包含多个句子，每个句子具有情绪标注。要去根据对话历史实现对最后一句话的情绪预测。

训练集样例：

Text：“所以在盼望结束么 \_\_eou\_\_ 是期待奖品的那天嘻嘻 \_\_eou\_\_ 那要是抽不到你你不得哭 \_\_eou\_\_ 抽不到口红我可能会哭”

Label：6163

共有六种情绪标注：（幸福，喜爱，痛苦，恐惧，厌恶，无），对应标签数字1~6。

要求根据对话历史（给定每句话的情绪标注），预测最后一句话的情绪。

可以将这个任务视为一个中文文本多分类问题来解决。

## 二、数据预处理

本文采用如下方式处理训练集数据：对于一条对话D={C1,C2…Cn|Label=l1,l2…ln}，将其转换为一组输入，形式为[C1,C1+l1+C2,…,C1+l1+C2+l2+…Cn|Label=l1,l2…ln]。其中‘+’操作为字符串拼接。对于一个由n个句子组成的对话，经过处理后则可以得到n条训练数据，每条数据的标签即为改部分对话末尾句的标签。这样做是为了利用数据集是对话数据这个特点，并且增加了训练集样本。将句子的情感标签加入到训练集中也是为了提高“对话”与“情绪”的作用。转换之前，清除文本中非汉字、英文、数字、‘！’的符号，并将长度超过2的相同相邻字符串缩减到2个字符。将文本中的标签转为Int型，并减1（否则后面训练模型时会报错）。对于测试集中数据，每条对话转换为C1+l1+C2+l2+…Cn，标签未知。相当于只是将一个对话整合为一句话。

处理后的训练集样例：

原文：

我就奇怪了  为啥你能拍得这么美呢 \_\_eou\_\_ 因为我做什么都认真，都诚心诚意！ \_\_eou\_\_ 好你诚心诚意！我谦虚低调！咱都是优秀品格的人再赞一个  干杯 \_\_eou\_\_ 嗯嗯，咱俩都是最可爱的人！

处理后：

我就奇怪了为啥你能拍得这么美呢

我就奇怪了为啥你能拍得这么美呢2因为我做什么都认真，都诚心诚意！

我就奇怪了为啥你能拍得这么美呢2因为我做什么都认真，都诚心诚意！2好你诚心诚意！我谦虚低调！咱都是优秀品格的人再赞一个干杯

我就奇怪了为啥你能拍得这么美呢2因为我做什么都认真，都诚心诚意！2好你诚心诚意！我谦虚低调！咱都是优秀品格的人再赞一个干杯2嗯嗯，咱俩都是最可爱的人！

处理后的测试集样例：

原文：

您这技术高超啊 \_\_eou\_\_ 高超啥啊，那美女我还没拍她呢，她就说不要拍她哈哈。下次拍你。hoho \_\_eou\_\_ 敢问那美女是羊吗？ \_\_eou\_\_ 不是，哈哈！你好像见过一次呢，很高那个。

处理后：

您这技术高超啊2高超啥啊，那美女我还没拍她呢，她就说不要拍她哈哈。下次拍你。hoho1敢问那美女是羊吗？6不是，哈哈！你好像见过一次呢，很高那个。

## 三、模型描述

鉴于训练集中数据量不是很大（8000多条），所以在模型的选择上，使用了双向LSTM模型。其中词嵌入部分使用了预训练的中文词向量来构建词表。

本文采用的LSTM（长短时记忆网络）是一种特殊的RNN（循环神经网络）。RNN指的是网络的隐含层输出又作为其自身的输入。当使用RNN处理一个序列输入时，需要将RNN按输入时刻展开，然后将序列中的每个输入依次对应到不同网络时刻的输入上，将当前网络隐含层的输出也作为下一时刻的输入。令RNN在最后时刻输出结果，则可将其用于文本分类任务。

在原始的RNN中，信息通过多个隐含层逐层传输到输出层，在这个过程中会导致信息损失，这会使得网络参数难以优化。LSTM的关键是细胞状态。LSTM采用门控机制实现对细胞状态中信息增加或减少的控制。原理是：门的输出是 0到1 之间的实数向量，当门输出为 0 时，任何向量与之相乘都会得到 0 向量，这就相当于什么都不能通过；输出为 1 时，任何向量与之相乘都不会有任何改变，这就相当于什么都可以通过。LSTM用三个门来实现对信息的控制。LSTM的遗忘门决定上一时刻的细胞状态有多少保留到当前时刻，借此过滤掉贡献小的旧状态；输入门决定了当前时刻网络的输入有多少保存到细胞状态，借此为状态更新信息；输出门控制细胞状态有多少输出到LSTM的当前输出值。

在传统的RNN与LSTM中，信息都是单向流动，这样的模型无法捕捉到一个单词与其后面单词的关系，但在有些问题中，当前时刻的输出不仅和之前的状态有关，还可能和未来的状态有关系。为解决这个问题，可以采取双向LSTM，其思想是将同一个输入序列分别接入向前和向后两个LSTM，将两个LSTM的隐含层拼接在一起，共同接入输出层进行预测。

## 四、实验步骤

模型的搭建，使用windows环境下GPU版本的Pytorch。

实验流程如下：

1. 读取训练集数据与测试集数据。本实验使用torchtext中legacy.data模块中的TabularDataset来加载数据集，其中训练集的Fields包括文本与标签两部分，而测试集只有文本。对文本进行分词，实验中使用jieba.cut实现中文分词。将训练数据按0.95：0.05随机地分出一个验证集用来在后续对模型地迭代训练中进行对模型效果的评估。
2. 利用下载的基于微博数据预训练的词向量构建词表。随后利用torchtext中legacy.data中的Iterator构建训练数据迭代器、验证数据迭代器、测试数据迭代器，batch\_size设为128，按照句子的长度来排序batch，即让长度相近的句子放在一个batch中。
3. 使用torch.nn.LSTM搭建LSTM网络，embedding部分使用前面基于预训练词向量构建好的词表，设置参数bidirectional=True，在LSTM网络后加入一个线性层用来得到分类的结果。
4. 设置优化器为Adam优化器，损失函数为交叉熵损失。
5. 迭代训练模型，每一轮epoch包括两个部分（模型训练与评估）。模型的训练流程为：加载训练数据和标签-模型输入输出-计算损失函数-损失反向传播-梯度截断-Adam优化器更新梯度参数。模型的评估采用准确率、召回率与F1值这三项指标。动态地选取F1值最高的那一轮epoch的模型保存，用来后续的测试。
6. 用保存的模型对测试集做出预测，输出类标签。

## 五、参数调试

截取几组不同超参数下的模型评估结果

epoch:5 - acc: 0.6813 recall:0.6423 f1:0.6395

epoch:6 - acc: 0.7623 recall:0.6252 f1:0.6638

epoch:7 - acc: 0.7093 recall:0.6915 f1:0.6996

epoch:8 - acc: 0.7223 recall:0.6597 f1:0.6867

lr=0.0008 hidden size=128 baidu（词向量）

epoch:5 - acc: 0.7213 recall:0.6684 f1:0.6837

epoch:6 - acc: 0.7262 recall:0.6762 f1:0.6971

epoch:7 - acc: 0.6971 recall:0.6951 f1:0.6949

epoch:8 - acc: 0.7050 recall:0.6757 f1:0.6885

lr=0.0015 hidden size=128 baidu（词向量）

epoch:5 - acc: 0.6875 recall:0.6846 f1:0.6834

epoch:6 - acc: 0.7330 recall:0.6491 f1:0.6822

epoch:7 - acc: 0.6928 recall:0.6826 f1:0.6866

epoch:8 - acc: 0.7364 recall:0.6780 f1:0.7037

lr=0.001 hidden size=256 baidu（词向量）

epoch:1 - acc: 0.6122 recall:0.5192 f1:0.5399

epoch:2 - acc: 0.8038 recall:0.5708 f1:0.6003

epoch:3 - acc: 0.7531 recall:0.6009 f1:0.6509

epoch:4 - acc: 0.7440 recall:0.6237 f1:0.6651

epoch:5 - acc: 0.7484 recall:0.6589 f1:0.6855

epoch:6 - acc: 0.7290 recall:0.6690 f1:0.6943

epoch:7 - acc: 0.7087 recall:0.6767 f1:0.6908

epoch:8 - acc: 0.7169 recall:0.6822 f1:0.6978

epoch:9 - acc: 0.7425 recall:0.6603 f1:0.6816

epoch:10 - acc: 0.6950 recall:0.6764 f1:0.6853

lr=0.001 hidden size=256 weibo（词向量）

根据实验结果可以看出，一般在训练到epoch5到8时（模型层数为3时），模型的F1值可以达到最高，能达到0.7左右，模型的准确率基本在0.7以上，召回率略低，在0.65以上。

## 六、实验结果

令学习率lr=0.0015，隐藏层维度hidden size=256，词向量选用微博数据训练的词向量，模型层数设为3，达到最好效果：

epoch:5 - acc: 0.7523 recall:0.6929 f1:0.7141

epoch:6 - acc: 0.7459 recall:0.6885 f1:0.7122

epoch:7 - acc: 0.7095 recall:0.6707 f1:0.6861

epoch:8 - acc: 0.7317 recall:0.6902 f1:0.7087

测试集结果为acc: 0.7512 recall:0.6813 f1:0.7145

## 七、心得体会

这次实验是我第一次动手实现一个nlp相关任务（也是第一个深度学习任务）。一开始先从多层感知机学起，之后学习并实现了简单的LSTM网络，也实现了一个transformer。一开始是准备使用transformer完成实验的，但是实验过程中发现效果并不好，不管怎么调参数准确率基本就在50%多一点，这个效果就和普通的LSTM差不多，我以为是写的transformer有问题，于是后面就在LSTM上调试，也尝试使用了Bi-LSTM。后面经同学指点，发现是词表构建的问题。

一开始，在词表构建的时候，我使用训练集数据构建词表。在对文本分词后,利用一个Vocab类实现词到id的映射，依此构建词表。后面发现这种构建词表的方式有问题，于是尝试使用预训练的词向量，发现效果显著提升。

关于数据的预处理。一开始把训练数据进行简单逐句的切分，相当与把情感预测任务作为一个单纯的文本分类任务来做，这样就没有用到对话这个性质。于是后面改为了前面描述的数据处理方式，即处理无用符号后结合标签进行拼接。为了扩大训练集，把每个字句也作为训练数据，而不是仅使用一条对话。

关于模型训练，本实验中模型一般在epoch5-8时达到估计的F1最高，继续训练则效果会下降。模型的层数方面，layers=3会比2好，但增加到4就效果不明显了，所以最后使用layers=3.

后续改进思路：训练集中，不同标签的分布很不均衡。后续的改进思路是，对占比较高的标签采取欠采样，或者对占比较少的标签过采样。可以参考EasyEnsemble方法，对占比较高的标签欠采样，之后用集成的思想训练多个分类器，然后通过硬投票表决得到最终预测结果。