

## 复旦大学计算机科学技术学院

### 2018-2019 学年第一学期期末论文课程评分表

课程名称： 自然语言处理

课程代码： COMP130141.01

开课院系： 计算机科学技术学院

学生姓名： 何林芳 学号： 16307130364 专业： 计算机科学与技术

论文名称： 《为网络聊天信息添加表情》

（以上由学生填写）

成绩： \_\_\_\_\_

论文评语（教师填写）：

任课教师签名：

日 期：

# 为网络聊天信息添加表情

【何林芳 16307130364】

## 【问题描述】

编写一个程序能够根据输入的句子自动添加一个表情。

选题受 [Ng 深度学习课程作业-emojiify](#) 启发。平常在网络聊天中，仅仅只发送文字消息会显得枯燥乏味，但每次需要自己添加表情略显麻烦，并且不同的个体所习惯使用的 emoji 各不相同，基于此我希望设计一个轻量级的中文版本的、有个体差异性的序列模型生成想要的表情，并比较不同模型的不同准确度。

## 【关键词】

Embedding, Word2Vec, Recurrent Neural Network, Long Short Term Memeory, Language model







## 【总体思路】

1. 获取并处理中文维基百科语料，采用 word2vec 得到词向量
2. 使用监督学习方式，收集常用的带 emoji 的聊天信息，分成训练集和测试集（后面会合并）
3. 把句子用词向量表达作为特征，把 emoji 作为标签进行训练
4. 进行预测，模型评估

## 【具体过程】

### 一、搜集语料，获取词向量

1. 语料来源：中文的语料收集比英文要麻烦不少，由于爬虫经验的不足，我选择了已经比较全面的[中文维基语料](#)。
2. 提取文本：用 WikiExtractor 从原始的 xml 文件中提取出标题和正文，新建[脚本](#)运行。
3. 繁简转换：由于得到的语料库是繁体中文，这与平常使用习惯不符，因此利用 [openccc](#) 工具进行繁简体转换。
4. 新建脚本对文件中的特殊符号进行替换。最后得到 6 个文件，后面 3 个是简体中文。

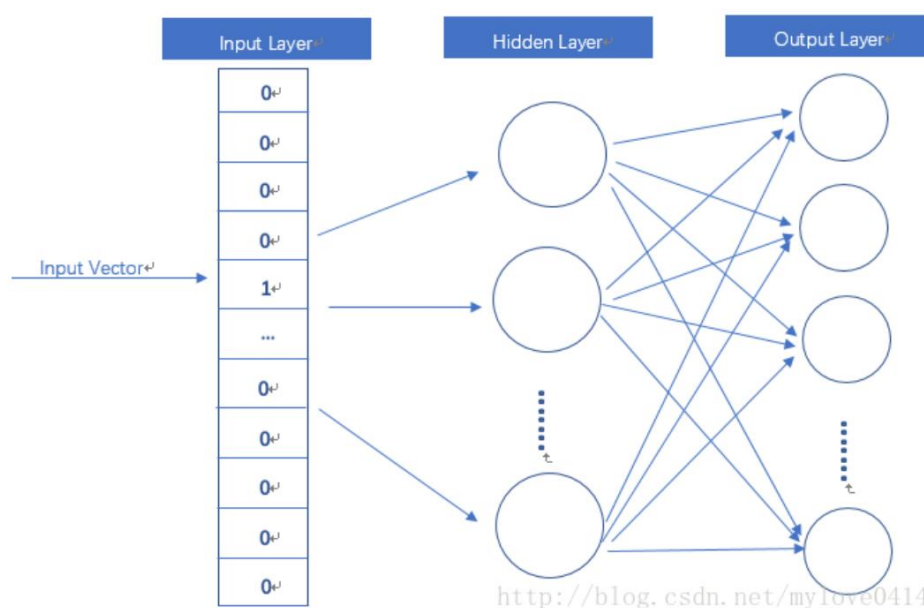
名称	类型	大小
 wiki_00	文件	512,000 KB
 wiki_01	文件	512,000 KB
 wiki_02	文件	245,458 KB
 zh_wiki_00	文件	516,628 KB
 zh_wiki_01	文件	518,817 KB
 zh_wiki_02	文件	248,336 KB

## 5. 使用 gensim 的 word2vec 训练 [参考](#)

如果用 one-hot 对词汇编码，即会十分冗长，也不能表示词与词之间的关系，此时需要进行 word embedding（词嵌入），即将高维词向量嵌入到一个低维空间，这个训练过程会引入词的上下文，从而能表达这个词的意思。

word2vec 模型其实就是简单化的神经网络。输入是 One-Hot Vector，Hidden Layer 没有激活函数，也就是线性的单元。Output Layer 维度跟 Input Layer 的维度一样，用的是 Softmax 回归。这个模型定义数据的输入和输出的方法一般分为 CBOW(Continuous Bag-of-Words) 与 Skip-Gram 两种。

CBOW 模型的训练输入是某一个特征词的上下文相关的词对应的词向量，而输出就是这特定的一个词的词向量。Skip-Gram 模型和 CBOW 的思路是反着来的，即输入是特定的一个词的词向量，而输出是特定词对应的上下文词向量。根据查询：CBOW 对小型数据库比较合适，而 Skip-Gram 在大型语料中表现更好。



将简体文字文件转化为 txt 格式，对其进行分词，选用 word 分词算法处理专有名词比如人名等，再模型使用之前还需删掉一些词向量维数不是 200 维的错误情况。（见 utils.py 中的 read\_vecs 函数的 if 判断语句。）

由于数据量大，训练时间很长，本想做进一步改进去除 stopwords，处于时间考虑没有作此改进。

## 6. 训练结果考察

从下图中可以看到，词汇的关联性符合我们的日常经验，可以接受用这个 .vec 做为后续使用的词向量。

```
#输入gensim, 看词汇之间的相关性
```

```
from gensim.models.keyedvectors import KeyedVectors
word_vectors = KeyedVectors.load_word2vec_format("data/newsblogbbs.vec", binary = False)
```

```
word_vectors.most_similar('小猫', topn = 10)
```

```
[('猴子', 0.5022560358047485),
 ('兔子', 0.49907466769218445),
 ('狗', 0.494366317987442),
 ('小兔子', 0.49164506793022156),
 ('小猴', 0.4901922643184662),
 ('狐狸', 0.478057324886322),
 ('小老鼠', 0.46818697452545166),
 ('大狗', 0.46066048741340637),
 ('小狗', 0.4522992968559265),
 ('狗狗', 0.4488312900066376)]
```

```
word_vectors.similarity("电视剧", "电影")
```

```
0.67289269
```

## 二、数据集处理

以上的语料训练只是最基础的一步：得到词汇的向量化表示。由于我希望这个轻量级能够个性化，因此我只使用了自己的信息作为训练集/测试集。

1. 导出我的微信聊天记录，导出过程参考[这里](#)
2. 选取可行的最常用的几种表情

由于微信自带的聊天表情有些不在 emoji 模块中，比如[捂脸]表情，因此需要排除这些表情或者用相似的表情替换。

最常用的表情使用频率如图

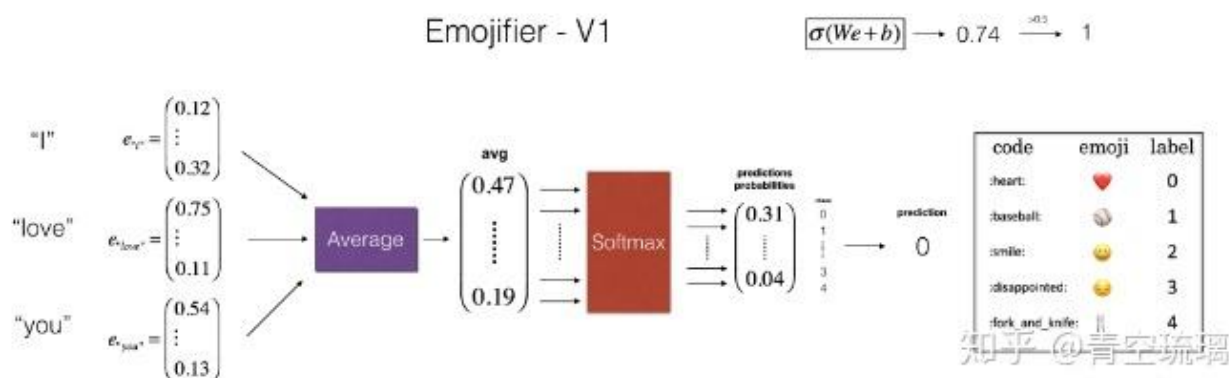
表情	编号	次数
😞	7	9
🙄	6	4
🤔	5	9
🙊	4	23
😏	3	54
😁	2	53
🙄	1	20
❤️	0	28

3. 选择最常用的 8 种表情，用数字代表，制作成 csv 文件，第一列是文字消息，第二列是用数字表示的表情符号。
4. 在训练过程中，利用 jieba 对中文进行分词，把数字表示的 label 转换为 one-hot 向量

## 三、语言模型

本次实验探究了两种模型，一个是循环神经网络 RNN (LSTM)，一个是仅使用一个 softmax 层的网络。

### 1. Baseline 模型



模型的输入是对应于句子的分词（我用 jieba 对中文进行分词），输出是形状（1,8）的概率向量，然后在 `argmax` 层中传递以提取最可能的表情符号输出的索引。为了使我们的标签成为适合训练 `softmax` 分类器的格式，我们将 `label` 从其当前形状当前形状（`m, 1`）转换为“one-hot”（`m, 8`），其中每一行都是一个 one-hot 矢量。

第一步是将输入的句子进行分词，并且去掉‘ ’的影响，然后将其平均在一起(average pooling)，得到这个句子的表示，使用之前训练好的词向量来表示这个句子。（200 维）

由于我们的标签是 one-hot 向量表示，因此需要模型的构造是：

$$Z^{(i)} = W * avg^{(i)} + b$$

$$a^{(i)} = softmax(z^{(i)})$$

$$L^{(i)} = - \sum_{k=0}^{n-1} Yoh_k^{(i)} * \log(a_k^{(i)})$$

训练结果

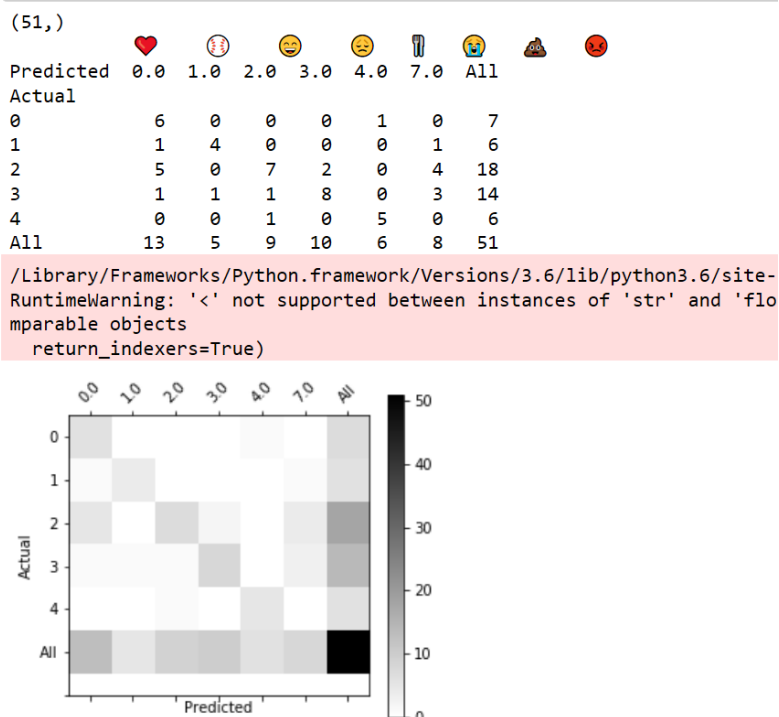
Epoch	Cost	Accuracy
0	2.00143172853	0.311258278146
100	0.817315720679	0.721854304636
200	0.594640808365	0.834437086093
300	0.493292517258	0.887417218543
400	0.428624999241	0.927152317881
500	0.377853540802	0.940397350993
600	0.334819241469	0.94701986755
700	0.297968634949	0.953642384106
800	0.266642597598	0.966887417219
900	0.240182678998	0.973509933775

```
print("Training set:")
pred_train = predict(X_train, Y_train, W, b, word_to_vec_map)
print('Test set:')
pred_test = predict(X_test, Y_test, W, b, word_to_vec_map)
```

Training set:  
Accuracy: 0.973509933775  
Test set:  
Accuracy: 0.764705882353

模型训练后的准确率达到 97%，但若是测试集测试，精度只有 76%，说明这个模型的鲁棒性不强。

查看混淆矩阵



这里对句子的处理是 average pooling，我还尝试了 max pooling，但是

## 2. LSTM 模型

前面说到，最基础的 baseline model 鲁棒性低，因此我们针对现有的特点进行改进。

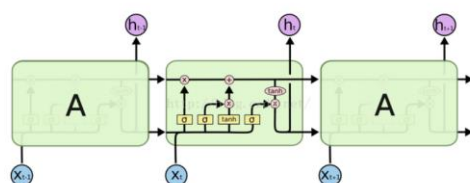
数据集小：由于数据量不大，再将其拆分为训练集和测试集可能会影响精度，因此我将两个 csv 文件整合起来，形成一个文件，增强模型的泛化能力。

未关注词序：baseline model 应用在这样一个类似情感分析的问题中，虽然看起来精度是可以接受的，但是数据量一大，可能效果就不会很好，因此，考虑到语言独有的 sequence 特征，利用 lstm 将会是一个不错的选择。同时也是因为数据不多，为了防止过拟合，需要添加 dropout。

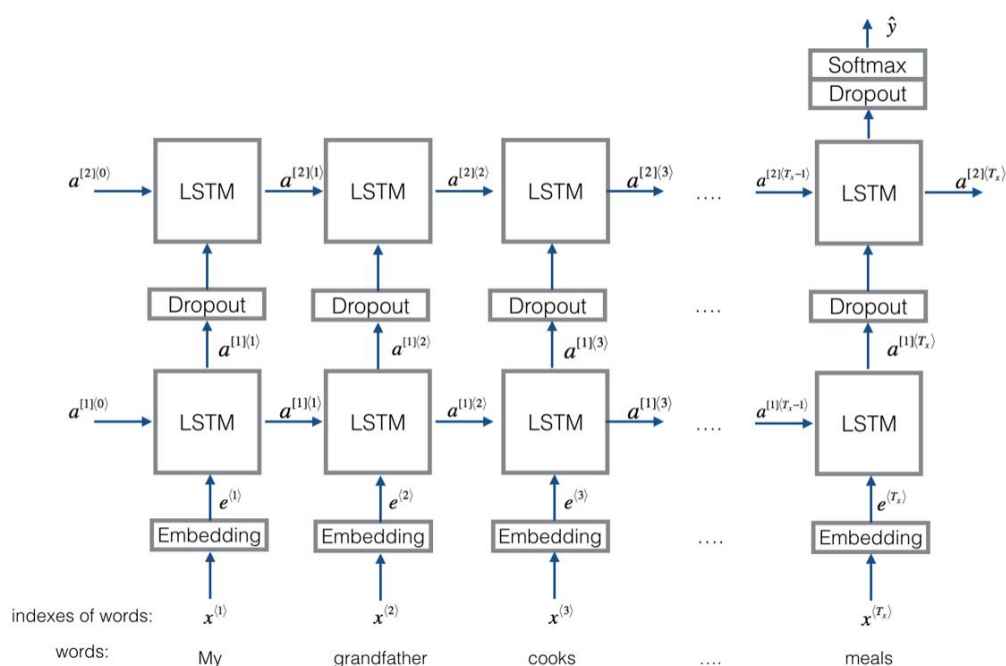
长短期记忆模型(long-short term memory)：一种特殊的 RNN 模型，是为了解决 RNN 模型梯度弥散的问题而提出的；在传统的 RNN 中，训练算法使用的是 BPTT，当时间比较长时，需要回传的残差会指数下降，导致网络权重更新缓慢，无法体现出 RNN 的长期记忆的效果，因此需要一个存储单元来存储记忆，因此 LSTM 模型被提出。

我并没有自己实现 lstm，而是利用 pytorch 已有的模型来构建。

使用交叉熵作为损失函数，我认为这样一个 many-to-one 的问题就是分类问题，而分类问题一般都是使用 cross entropy，同样利用 pytorch 中已实现的部分。



所以构建这样一个模型:



对应的模型构建代码:

```
def __init__(self, vocab_size, embedding_dim, pretrained_weight):
    super(myModel, self).__init__()
    self.word_embeddings = nn.Embedding(vocab_size, embedding_dim)
    pretrained_weight = np.array(pretrained_weight)
    self.word_embeddings.weight.data.copy_(torch.from_numpy(pretrained_weight))
    self.rnn = nn.LSTM(embedding_dim, 128, 2, batch_first=True, dropout=0.5)
    self.linear = nn.Linear(128, 8)
    self.out = nn.Softmax()
```

其中 batch\_size 是 20.

#### 四、预测 demo

##### 1. Simple model

```
X_my_sentences = np.array(["我喜欢你", "我憎恨你", "他太傻了", "想吃薯条", "气死我了", "这部电影真好看"])
Y_my_labels = np.array([[0], [7], [3], [1], [7], [2]])

pred = predict(X_my_sentences, Y_my_labels, W, b, word_to_vec_map)
print_res(X_my_sentences, pred)
```

Accuracy: 0.666666666667

我喜欢你 ❤️  
 我憎恨你 😞  
 他太傻了 😞  
 想吃薯条 🍟  
 气死我了 😡  
 这部电影真好看 😊

## 2. Lstm model

```
x_test = np.array(['我喜欢你', '今天天气真好', '你真好看', '我们打球去'])
X_test_indices = feature_matrix(x_test, word_to_index, 12)
X_test_indices = torch.from_numpy(X_test_indices)
X_test_indices = X_test_indices.long()
pred = model(X_test_indices)
for i in range(len(x_test)):
    num = np.argmax(pred.data[i])
    print('prediction: ' + x_test[i] + label_to_emoji(num.item()).strip())

torch.save(model.state_dict(), "emojify.pkl")
```

D:\Programming\Anaconda3\lib\site-packages\ipykernel\_launcher.py:16: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include dim=X as an argument.

```
app.launch_new_instance()
```

< >

```
prediction: 我喜欢你❤️
prediction: 今天天气真好😊
prediction: 你真好看😁
prediction: 我们打球去🏀
```

### 【总结与改进】

现有的模型有如下不足：

1. 不善于处理否定类表达，比如“不开心”添加了[开心]的表情，“不喜欢”添加了[爱心]表情，而是只能简单处理“喜欢”，“讨厌”这一类感情特征十分明显的词汇。
2. 句子一长，感情表达复杂，就容易出现混乱。

对于这个模型，未来应用时可以做如下改进：

#### 1. 个性化

个性化体现在数据集上，由于这里仅使用了我个人的聊天信息，所以训练出来的模型只符合我的习惯，而换一个人需要换一个训练数据集。

#### 2. 增强鲁棒性

由于我是 CPU 训练数据集需要大量时间，如果时间充足的话我会在词向量里，或者是之后对句子的向量化表示中，增强情感词汇的表达（增加表情本质上是情感分类问题），而降低诸如“今天”，“她”这样的对感情意义不大的词的表达。

#### 3. 增加表情

为了结果的可视化，我并没有真正按照微信聊天记录的表情频率来训练，而是根据 emoji 模块中已有的表情来进行选择，但实际上只要把表情用独一无二的数字表示出来就可以了，并不一定非要对应某个编码。

#### 4. 表情包功能

每个人都有自己习惯使用的表情包（图片/动图等），我希望以后能针对图片也进行编码，用 cnn 识别图片信息，将表情包也纳入可增添范围。

### 【感想】

由于缺乏经验，一开始我走了很多弯路，尤其是在找中文语料并处理的时候，花了很多时间却总是结果不理想，在网上搜资料的时候也是鱼龙混杂，所以刚开始时很痛苦。

数据部分处理完毕后，剩下的模型构建工作，我得到的经验是一定要多看源码，有些问题不一定能搜到，但是分析源代码总能得到意想不到的收获，再依葫芦画瓢即可。

平常在课程上学到的理论知识，比如分词等，只有经过自己动手实践才有深刻的认识，比如我一开始对分词认识不够，而在使用了 jieba 后才明白分词的真正意义是为了便于对整



个句子的总体把握。本次实验的模型虽然简单，但是我认为非常有趣，甚至可以用到日常生活中。

#### 【参考资料、工具与文件】

1. 中文维基语料 <https://dumps.wikimedia.org/zhwiki/latest/zhwiki-latest-pages-articles.xml.bz2>
2. 语料提取方法 <https://blog.csdn.net/u013421941/article/details/68947622>
3. Andrew Ng deeplearning.ai 编程作业 <https://zhuanlan.zhihu.com/p/47762573>
4. 用 wiki 语料训练 word2vec 模型 <https://blog.csdn.net/hereiskxm/article/details/49664845>
5. 通俗理解 word2vec <https://www.jianshu.com/p/471d9bfb72f>
6. Keras 实现 <https://blog.csdn.net/kurumi233/article/details/79349080>
7. 邱锡鹏《神经网络与深度学习》 <https://nndl.github.io/>

#### 【附件清单】

Data 文件夹:

Train\_emooji\_ch.csv 训练数据集

Tesss\_ch.csv 测试数据集

Combined\_data.csv 合并数据集

myWords.vec 词向量（可能由于过大无法传至 elearning）

.ipynb\_checkpoints、\_\_pycache\_\_ 文件夹：与 ipynb 文件相关联

模型的实现过程:

simple\_model.ipynb

lstm\_model.ipynb

lstm\_model2.ipynb 和前一个是一样的，但是由于一些误操作可能 checkpoints 不同

对应的 py 文件:

Utils.py 对数据的一些统一操作

Lstm\_model.py 和 simple\_modes 对应上述 ipynb 文件的代码

Preprocess 文件夹：预处理数据中用到

Process\_wiki.py

WikiExtractor.py

Emojify.pkl 是 torch 保存的 lstm 模型参数