# 实验四 电影评论情感分类

## 一、 实验目的

1. 进一步加深对卷积神经网络基本原理的理解。

2. 掌握卷积神经网络处理文本的各项技术。

3. 掌握文本分类模型Text-CNN的架构和原理。

## 二、 实验要求

1. 任选一个深度学习框架建立Text-CNN模型（本实验指导书以TensorFlow为例）。

2. 实现对中文电影评论的情感分类，实现测试准确率在83%以上。

3. 也可采用LSTM实现，实现测试准确率高于卷积神经网络。

4. 按规定时间在课程网站提交实验报告、代码以及PPT。

## 三、 实验原理

Text-CNN和传统的CNN结构类似，具有词嵌入层、卷积层、池化层和全连接层的四层结构，如图1所示。

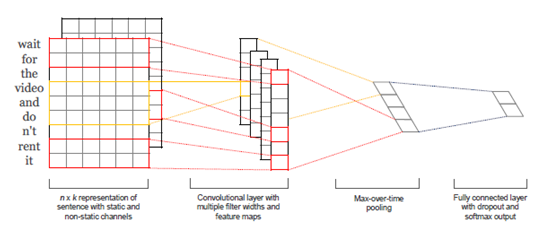


图1 Text-CNN网络结构

如图1中所示，Text-CNN的词嵌入层（Word embedding）使用二维矩阵来表示长文本。词嵌入将输入文本的每个词语通过空间映射，将独热表示（One-Hot Representation）转换成分布式表示（Distributed Representation），进而可以使用低维的词向量来表示每一个词语。经过词嵌入，每个单词具有相同长度的词向量表示。将各个词语的向量表示连起来便可以得到二维矩阵。得到词向量的方式有多种，常用的是Word2vec方法。若使用预训练好的词向量，在训练模型的时候可以选择更新或不更新词向量，分别对应嵌入层状态为Non-static和Static。

Text-CNN的卷积层是主要部分，卷积核的宽度等于词向量的维度，经卷积后可以提取文本的特征向量。与在图像领域应用类似，Text-CNN可以设置多个卷积核以提取文本的多层特征，长度为N的卷积核可以提取文本中的N-gram特征。

Text-CNN的池化层一般采取Max-over-time pooling，输出最大值，从而判断词嵌入中是否含N-gram。

Text-CNN的全连接层采用了Dropout算法防止过拟合，并使用Softmax函数输出各个类别的概率。

算法具体原理可阅读：Kim Y .2014--《Convolutional Neural Networks for Sentence Classification》一文。

## 四、 实验所用工具及数据集（以PyTorch为例）

1. 主要工具

Python 3.6.9、PyTorch 1.4.0、numpy-1.16.6、gensim-4.0.1

2. 数据集

1) 训练集。包含19998条中文电影评论，其中正负向评论各9999条。

2) 验证集。包含5629条中文电影评论，其中正负向评论各2812,2817条。

3) 测试集。包含369条中文电影评论，其中正负向评论各187,182条。

4) 预训练词向量。中文维基百科词向量word2vec：wiki\_word2vec\_50.bin。

## 五、 实验步骤与方法

1. 加载本实验所有函数库

**import** gensim

**import** torch

**import** torch**.**nn **as** nn

**import** torch**.**nn**.**functional **as** F

**import** numpy **as** np

**from** collections **import** Counter

**from** torch**.**utils**.**data **import** TensorDataset**,** DataLoader

2. 数据预处理

①　设置分类类别以及类别对应词典{pos:0, neg:1};

②　从train.txt, validation.txt构建词汇表并存储，形如{word: id};

**def** build\_word2id**(**save\_to\_path**=None):**

"""

:param save\_to\_path: path to save word2id

:return: word2id dictionary {word: id}

"""

word2id **=** **{**'\_PAD\_'**:** 0**}**

path **=** **[**'./Dataset/train.txt'**,** './Dataset/validation.txt'**]**

# write the index to word2id[word]

**for** \_path **in** path**:**

**with** **open(**\_path**,** encoding**=**'utf-8'**)** **as** f**:**

**for** line **in** f**.**readlines**():**

sp **=** line**.**strip**().**split**()**

**for** word **in** sp**[**1**:]:**

**if** word **not** **in** word2id**.**keys**():**

word2id**[**word**]** **=** **len(**word2id**)**

**if** save\_to\_path**:**

**with** **open(**save\_to\_path**,** 'w'**,** encoding**=**'utf-8'**)** **as** f**:**

**for** w **in** word2id**:**

f**.**write**(**w**+**'\t'**)**

f**.**write**(str(**word2id**[**w**]))**

f**.**write**(**'\n'**)**

**return** word2id

③　加载上述构建的词汇表;

word2id **=** build\_word2id**(**'./Dataset/word2id.txt'**)**

**print(type(**word2id**),** **len(**word2id**))**

输出：

<class 'dict'> 58954

④　基于预训练好的word2vec: wiki\_word2vec\_50.bin构建训练语料中所含词语的word2vec;

**def** build\_word2vec**(**fname**,** word2id**,** save\_to\_path**=None):**

"""

:param fname: pre-trained word2vec by others

:param word2id: built word2id by us

:param save\_to\_path: path to save word2vec

:return: wordid\_vecs means wordid to wordvector dictionary {id: word2vec}

"""

n\_words **=** **max(**word2id**.**values**())** **+** 1

model **=** gensim**.**models**.**KeyedVectors**.**load\_word2vec\_format**(**fname**,** binary**=True)**

wordid\_vecs **=** np**.**array**(**np**.**random**.**uniform**(-**1.**,** 1.**,** **[**n\_words**,** model**.**vector\_size**]))**

**for** word **in** word2id**.**keys**():**

**try:**

wordid\_vecs**[**word2id**[**word**]]** **=** model**[**word**]**

**except** **KeyError:**

**pass**

**if** save\_to\_path**:**

**with** **open(**save\_to\_path**,** 'w'**,** encoding**=**'utf-8'**)** **as** f**:**

**for** vec **in** wordid\_vecs**:**

vec **=** **[str(**w**)** **for** w **in** vec**]**

f**.**write**(**' '**.**join**(**vec**))**

f**.**write**(**'\n'**)**

**return** wordid\_vecs

⑤　加载上述构建的word2ve;

word2vec **=** build\_word2vec**(**'./Dataset/wiki\_word2vec\_50.bin'**,** word2id**)**

**assert** word2vec**.**shape **==** **(**58954**,** 50**)**

**print(**word2vec**)**

输出：

**[[** 0.58683994 0.47758984 0.34983207 **...** 0.24294434 0.58708753

**-**0.24165251**]**

**[** 0.33509246 **-**0.59427804 0.17512511 **...** **-**0.03102478 0.0637431

0.07115829**]**

**[** 0.59741241 0.75101197 0.18067721 **...** **-**0.18746521 0.6136632

**-**0.12780823**]**

**...**

**[-**0.14330814 **-**0.1964599 0.35209112 **...** **-**0.69162697 **-**0.17640286

0.6477952 **]**

**[** 0.2875104 0.22257346 0.6065948 **...** **-**0.10173644 0.16712433

0.191283 **]**

**[** 0.50211847 **-**0.45644817 0.21553677 **...** **-**0.34367573 0.1960599

**-**0.03544554**]]**

⑥　加载语料库：train/dev/test;

生成批处理id序列。

经过数据预处理，数据的格式如下：

x: [1434, 5454, 2323, ..., 0, 0, 0]

y: [0, 1]

x为构成一条评论的词所对应的分类id。

y为onehot编码: pos-[1, 0], neg-[0, 1]

但是在训练计算损失函数时要求output.shape=(batch\_size, num\_classes), y.shape=(batch\_size, 1)，即y是1维的，所以one-hot在编码中用处不大。

**def** class\_to\_id**(**classes**=None):**

"""

:param classes: label to classify, default is 0:pos, 1:neg

:return: classes = ['0', '1'], {classes：id} = {'0': 0, '1': 1}

"""

**if** **not** classes**:**

classes **=** **[**'0'**,** '1'**]**

clas2id **=** **{**clas**:** idx **for** **(**idx**,** clas**)** **in** **enumerate(**classes**)}**

**return** classes**,** clas2id

**def** load\_corpus**(**path**,** word2id**,** max\_sen\_len**=**50**):**

"""

:param

path: sample corpus file

word2id: built word2id by us

:return

contents: array, text to id;

labels\_arr: array, (len,)

labels\_onehot: array, onehot format, (len, 2)

"""

\_**,** clas2id **=** class\_to\_id**()**

contents**,** labels **=** **[],** **[]**

**with** **open(**path**,** encoding**=**'utf-8'**)** **as** f**:**

**for** line **in** f**.**readlines**():**

sp **=** line**.**strip**().**split**()**

# print(sp)

label **=** sp**[**0**]**

content **=** **[**word2id**.**get**(**w**,** 0**)** **for** w **in** sp**[**1**:]]**

content **=** content**[:**max\_sen\_len**]**

**if** **len(**content**)** **<** max\_sen\_len**:**

content **+=** **[**word2id**[**'\_PAD\_'**]]** **\*** **(**max\_sen\_len **-** **len(**content**))**

labels**.**append**(**label**)**

contents**.**append**(**content**)**

counter **=** Counter**(**labels**)**

**print(**'总样本数为：%d' **%** **(len(**labels**)))**

**print(**'各个类别样本数如下：'**)**

**for** w **in** counter**:**

**print(**w**,** counter**[**w**])**

contents **=** np**.**asarray**(**contents**)**

labels\_arr **=** np**.**array**([**clas2id**[**l**]** **for** l **in** labels**])**

labels\_onehot **=** np**.**array**([[**0**,**0**]]** **\*** **len(**labels**))**

**for** idx**,** val **in** **enumerate(**labels**):**

**if** val **==** '0'**:**

labels\_onehot**[**idx**][**0**]=**1

**else:**

labels\_onehot**[**idx**][**1**]=**1

**return** contents**,** labels\_arr**,** labels\_onehot

**print(**'train corpus load: '**)**

train\_contents**,** train\_labels**,** \_ **=** load\_corpus**(**'./Dataset/train.txt'**,** word2id**,** max\_sen\_len**=**50**)**

**print(**'\nvalidation corpus load: '**)**

val\_contents**,** val\_labels**,** \_ **=** load\_corpus**(**'./Dataset/validation.txt'**,** word2id**,** max\_sen\_len**=**50**)**

**print(**'\ntest corpus load: '**)**

test\_contents**,** test\_labels**,** \_ **=** load\_corpus**(**'./Dataset/test.txt'**,** word2id**,** max\_sen\_len**=**50**)**

输出：

train corpus load**:**

总样本数为：19998

各个类别样本数如下：

1 9999

0 9999

validation corpus load**:**

总样本数为：5629

各个类别样本数如下：

1 2812

0 2817

test corpus load**:**

总样本数为：369

各个类别样本数如下：

1 187

0 182

3. 建立Text-CNN模型

①　配置模型相关参数，在COINFIG类中完成

**class** **CONFIG():**

update\_w2v **=** **True** # update w2v during training or not

vocab\_size **=** 58954 # vocabulary size same as the # of word2id

n\_class **=** 2 # the # of classes

embedding\_dim **=** 50 # dimension of wordvector

drop\_keep\_prob **=** 0.5 # dropout layer, the rate of parameter 'keep'

num\_filters **=** 256 # the # of filter in convolution layer

kernel\_size **=** 3 # the size of kernel in convolution layer

pretrained\_embed **=** word2vec # pretrained word embedding model

②　使用Pytorch框架完成Text-CNN模型的建立

**class** **TextCNN(**nn**.**Module**):**

**def** \_\_init\_\_**(**self**,** config**):**

**super(**TextCNN**,** self**).**\_\_init\_\_**()**

update\_w2v **=** config**.**update\_w2v

vocab\_size **=** config**.**vocab\_size

n\_class **=** config**.**n\_class

embedding\_dim **=** config**.**embedding\_dim

num\_filters **=** config**.**num\_filters

kernel\_size **=** config**.**kernel\_size

drop\_keep\_prob **=** config**.**drop\_keep\_prob

pretrained\_embed **=** config**.**pretrained\_embed

# Use the pre-trained wordvector

self**.**embedding **=** nn**.**Embedding**(**vocab\_size**,** embedding\_dim**)**

self**.**embedding**.**weight**.**data**.**copy\_**(**torch**.**from\_numpy**(**pretrained\_embed**))**

self**.**embedding**.**weight**.**requires\_grad **=** update\_w2v

# Convolution layer

self**.**conv **=** nn**.**Conv2d**(**1**,**num\_filters**,(**kernel\_size**,**embedding\_dim**))**

# Dropout

self**.**dropout **=** nn**.**Dropout**(**drop\_keep\_prob**)**

# Full connection layer

self**.**fc **=** nn**.**Linear**(**num\_filters**,** n\_class**)**

**def** forward**(**self**,** x**):**

x **=** x**.**to**(**torch**.**int64**)**

x **=** self**.**embedding**(**x**)**

x **=** x**.**unsqueeze**(**1**)**

x **=** F**.**relu**(**self**.**conv**(**x**)).**squeeze**(**3**)**

x **=** F**.**max\_pool1d**(**x**,** x**.**size**(**2**)).**squeeze**(**2**)**

x **=** self**.**dropout**(**x**)**

x **=** self**.**fc**(**x**)**

**return** x

4. 模型训练与验证

先设置一些参数，再使用训练集和验证集完成模型训练、验证。返回训练、验证损失和准确率。

config **=** CONFIG**()** # config the parameters of model

learning\_rate **=** 0.001 # learn rate

batch\_size **=** 32 # batch size

epochs **=** 4 # epoches

model\_path **=** **None** # path of pre-trained model

verbose **=** **True** # print the training process

device **=** torch**.**device**(**'cuda:0' **if** torch**.**cuda**.**is\_available**()** **else** 'cpu'**)**

**print(**device**)**

# Mix the contents & labels of train & validation dataset as train\_dataloader

contents **=** np**.**vstack**([**train\_contents**,** val\_contents**])**

labels **=** np**.**concatenate**([**train\_labels**,** val\_labels**])**

train\_dataset **=** TensorDataset**(**torch**.**from\_numpy**(**contents**).type(**torch**.float),**

torch**.**from\_numpy**(**labels**).type(**torch**.**long**))**

train\_dataloader **=** DataLoader**(**dataset **=** train\_dataset**,** batch\_size **=** batch\_size**,**

shuffle **=** **True,** num\_workers **=** 2**)**

**def** train**(**dataloader**):**

# config the model, load the pretrained model if model\_path

model **=** TextCNN**(**config**)**

**if** model\_path**:**

model**.**load\_state\_dict**(**torch**.**load**(**model\_path**))**

model**.**to**(**device**)**

# set optimizer & loss

optimizer **=** torch**.**optim**.**Adam**(**model**.**parameters**(),** lr **=** learning\_rate**)**

criterion **=** nn**.**CrossEntropyLoss**()**

# circuit train

**for** epoch **in** **range(**epochs**):**

**for** batch\_idx**,** **(**batch\_x**,** batch\_y**)** **in** **enumerate(**dataloader**):**

batch\_x**,** batch\_y **=** batch\_x**.**to**(**device**),** batch\_y**.**to**(**device**)**

output **=** model**(**batch\_x**)**

loss **=** criterion**(**output**,** batch\_y**)**

**if** batch\_idx **%** 200 **==** 0 **&** verbose**:**

**print(**'Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'**.format(**

epoch**+**1**,** batch\_idx **\*** **len(**batch\_x**),** **len(**dataloader**.**dataset**),**

100. **\*** batch\_idx **/** **len(**dataloader**),** loss**.**item**()))**

optimizer**.**zero\_grad**()**

loss**.**backward**()**

optimizer**.**step**()**

# save model

torch**.**save**(**model**.**state\_dict**(),** 'model.pth'**)**

train(train\_dataloader)

输出：

Train Epoch**:** 1 **[**0**/**25627 **(**0**%)]** Loss**:** 0.737682

Train Epoch**:** 1 **[**6400**/**25627 **(**25**%)]** Loss**:** 0.458602

Train Epoch**:** 1 **[**12800**/**25627 **(**50**%)]** Loss**:** 0.551417

Train Epoch**:** 1 **[**19200**/**25627 **(**75**%)]** Loss**:** 0.410252

Train Epoch**:** 1 **[**21600**/**25627 **(**100**%)]** Loss**:** 0.323111

Train Epoch**:** 2 **[**0**/**25627 **(**0**%)]** Loss**:** 0.282093

Train Epoch**:** 2 **[**6400**/**25627 **(**25**%)]** Loss**:** 0.233074

Train Epoch**:** 2 **[**12800**/**25627 **(**50**%)]** Loss**:** 0.520438

Train Epoch**:** 2 **[**19200**/**25627 **(**75**%)]** Loss**:** 0.351832

Train Epoch**:** 2 **[**21600**/**25627 **(**100**%)]** Loss**:** 0.277885

Train Epoch**:** 3 **[**0**/**25627 **(**0**%)]** Loss**:** 0.142906

Train Epoch**:** 3 **[**6400**/**25627 **(**25**%)]** Loss**:** 0.247886

Train Epoch**:** 3 **[**12800**/**25627 **(**50**%)]** Loss**:** 0.293193

Train Epoch**:** 3 **[**19200**/**25627 **(**75**%)]** Loss**:** 0.287232

Train Epoch**:** 3 **[**21600**/**25627 **(**100**%)]** Loss**:** 0.117848

Train Epoch**:** 4 **[**0**/**25627 **(**0**%)]** Loss**:** 0.229581

Train Epoch**:** 4 **[**6400**/**25627 **(**25**%)]** Loss**:** 0.180549

Train Epoch**:** 4 **[**12800**/**25627 **(**50**%)]** Loss**:** 0.033484

Train Epoch**:** 4 **[**19200**/**25627 **(**75**%)]** Loss**:** 0.088794

Train Epoch**:** 4 **[**21600**/**25627 **(**100**%)]** Loss**:** 0.322564

5. 模型测试

使用测试集完成模型的测试。通过准确率、精度、召回率、F1-分数、混淆矩阵指标来评估模型的性能。

# set test parameters

model\_path **=** 'model.pth'

batch\_size **=** 32

# load test dataset as test\_dataloader

test\_dataset **=** TensorDataset**(**torch**.**from\_numpy**(**test\_contents**).type(**torch**.float),**

torch**.**from\_numpy**(**test\_labels**).type(**torch**.**long**))**

test\_dataloader **=** DataLoader**(**dataset **=** test\_dataset**,** batch\_size **=** batch\_size**,**

shuffle **=** **False,** num\_workers **=** 2**)**

**def** predict**(**dataloader**):**

# load trained model

model **=** TextCNN**(**config**)**

model**.**load\_state\_dict**(**torch**.**load**(**model\_path**))**

model**.eval()**

model**.**to**(**device**)**

# circuit test

count**,** correct**,** real\_predict\_00**,** real\_predict\_01**,** real\_predict\_10**,** real\_predict\_11 **=** 0**,** 0**,** 0**,** 0**,** 0**,** 0

**for** \_**,** **(**batch\_x**,** batch\_y**)** **in** **enumerate(**dataloader**):**

batch\_x**,** batch\_y **=** batch\_x**.**to**(**device**),** batch\_y**.**to**(**device**)**

output **=** model**(**batch\_x**)**

count **+=** **len(**batch\_x**)**

correct **+=** **(**output**.**argmax**(**1**)** **==** batch\_y**).float().sum().**item**()**

# print(np.array([(output.argmax(1)[idx] == 0 and batch\_y[idx] == 0).float().cpu().numpy() for (idx, \_) in enumerate(batch\_y)]))

# print([(output.argmax(1)[idx] == 0 and batch\_y[idx] == 0).float().cpu().numpy() for (idx, \_) in enumerate(batch\_y)])

real\_predict\_00 **+=** np**.**array**([(**output**.**argmax**(**1**)[**idx**]** **==** 0 **and** batch\_y**[**idx**]** **==** 0**).float().**cpu**().**numpy**()** **for** **(**idx**,** \_**)** **in** **enumerate(**batch\_y**)]).sum().**item**()**

real\_predict\_01 **+=** np**.**array**([(**output**.**argmax**(**1**)[**idx**]** **==** 0 **and** batch\_y**[**idx**]** **==** 1**).float().**cpu**().**numpy**()** **for** **(**idx**,** \_**)** **in** **enumerate(**batch\_y**)]).sum().**item**()**

real\_predict\_10 **+=** np**.**array**([(**output**.**argmax**(**1**)[**idx**]** **==** 1 **and** batch\_y**[**idx**]** **==** 0**).float().**cpu**().**numpy**()** **for** **(**idx**,** \_**)** **in** **enumerate(**batch\_y**)]).sum().**item**()**

real\_predict\_11 **+=** np**.**array**([(**output**.**argmax**(**1**)[**idx**]** **==** 1 **and** batch\_y**[**idx**]** **==** 1**).float().**cpu**().**numpy**()** **for** **(**idx**,** \_**)** **in** **enumerate(**batch\_y**)]).sum().**item**()**

# calculate accuracy, precision, recall, F1\_score, confusion\_matrix

accuracy **=** correct**/**count

precision **=** real\_predict\_00 **/** **(**real\_predict\_00 **+** real\_predict\_10**)**

recall **=** real\_predict\_00 **/** **(**real\_predict\_00 **+** real\_predict\_01**)**

F1\_score **=** 2**\***precision**\***recall**/(**precision**+**recall**)**

confusion\_matrix **=** **[[**real\_predict\_00**,** real\_predict\_01**],** **[**real\_predict\_10**,** real\_predict\_11**]]**

**print(**'The accuracy, precision, recall, F1\_score, confusion\_matrix of test is\n{:.2f}% \n{} \n{} \n{} \n{}.'**.format(**100**\***accuracy**,** precision**,** recall**,** F1\_score**,** confusion\_matrix**))**

6. 预测

使用predict函数完成电影评论的情感分类。

predict**(**test\_dataloader**)**

输出：

The accuracy, precision, recall, F1\_score, confusion\_matrix of test is

86.18%

0.8626373626373627

0.8579234972677595

0.8602739726027399

[[157.0, 26.0], [25.0, 161.0]].

基本达到了83%以上分类正确率的要求。