# 6.5 卷积神经网络进行情感分类

# 导入本章所需要的模块

import matplotlib.pyplot as plt

import numpy as np

import os

import pandas as pd

import string

import torch

import torch.nn.functional as F

import torch.optim as optim

from torch import nn

from torchtext import data

from torchtext.vocab import Vectors

import re

import time

import copy

def load\_text\_data(path):

text\_data = []

label = []

for dset in ['pos', 'neg']:

path\_dset = os.path.join(path, dset) # dset : dataset

path\_list = os.listdir(path\_dset)

for fname in path\_list: # fname : filename

if fname.endswith('.txt'):

filename = os.path.join(path\_dset, fname)

with open(filename) as f:

text\_data.append(f.read())

if dset == 'pos ':

label.append(1)

else:

label.append(0)

return np.array(text\_data), np.array(label)

df1 = pd.read\_csv('train\_imdb.csv')

df2 = pd.read\_csv('test\_imdb.csv')

# 提取"text"字段的内容并转换为Numpy数组

text\_data = df1['text'].values

train\_text = np.array(text\_data)

train\_label = df1['label'].values

text\_data = df2['text'].values

test\_text = np.array(text\_data)

test\_label = df2['label'].values

print(len(train\_text), len(train\_label))

print(len(test\_text), len(test\_label))

# 25000 25000

# 25000 25000

def text\_preprocess(text\_data):

text\_pre = []

for text1 in text\_data:

text1 = re.sub('<br /><br />', ' ', text1)

text1 = text1.lower()

text1 = re.sub('\d+', '', text1)

text1 = text1.translate(str.maketrans("", "", string.punctuation.replace("'", "")))

text1 = text1.strip()

text\_pre.append(text1)

return np.array(text\_pre)

train\_text\_pre = text\_preprocess(train\_text)

test\_text\_pre = text\_preprocess(test\_text)

from nltk.tokenize import word\_tokenize

import numpy as np

import re

# nltk.download('punkt') # 下载punkt tokenizer

def stop\_stem\_word(datalist, stop\_words):

datalist\_pre = []

for text in datalist:

text\_words = word\_tokenize(text)

text\_words = [word for word in text\_words if word not in stop\_words]

text\_words = [word for word in text\_words if len(re.findall("'", word)) == 0]

datalist\_pre.append(text\_words)

return np.array(datalist\_pre)

from nltk.corpus import stopwords

# nltk.download('stopwords')

stop\_words = stopwords.words('english')

stop\_words = set(stop\_words)

train\_text\_pre2 = stop\_stem\_word(train\_text\_pre, stop\_words)

test\_text\_pre2 = stop\_stem\_word(test\_text\_pre, stop\_words)

print(train\_text\_pre[1])

# 篇章级语料

# i am curious yellow is a risible and pretentious steaming pile it doesn't matter what

print("=" \* 10)

print(train\_text\_pre2[1])

# 去除停用词，按照空格切分成一个个的单词

# ['curious', 'yellow', 'risible', 'pretentious', 'steaming', 'pile', 'matter', 'one', 'political'

# 装上NLTK的stopwords

# Traceback (most recent call last):

# File "D:\pythoncode\learn\a\deep\_learning6.5.py", line 94, in <module>

# stop\_words = stopwords.words('english')

# NameError: name 'stopwords' is not defined

texts = [" ".join(words) for words in train\_text\_pre2]

traindatasave = pd.DataFrame({"text": texts, "label": train\_label})

texts = [" ".join(words) for words in test\_text\_pre2]

testdatasave = pd.DataFrame({"text": texts, "label": test\_label})

traindatasave.to\_csv("imdb\_train.csv", index=False)

testdatasave.to\_csv("imdb\_test.csv", index=False)

traindata = pd.DataFrame({"train\_text": train\_text, "train\_word": train\_text\_pre2, "train\_label": train\_label})

train\_word\_num = [len(text) for text in train\_text\_pre2]

traindata["train\_word\_num"] = train\_word\_num

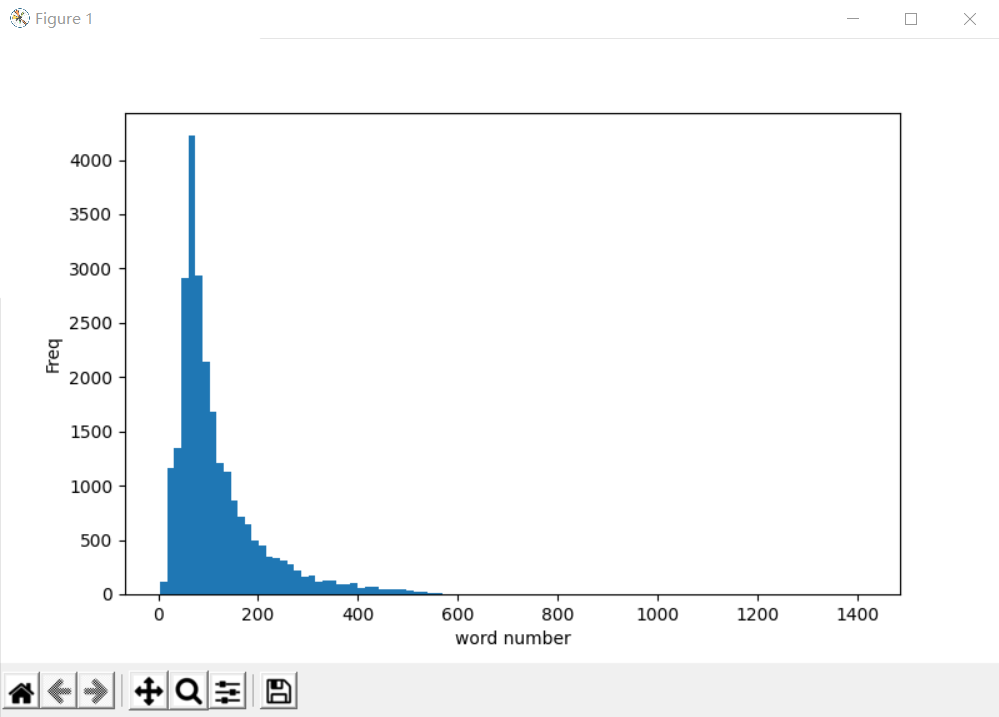
plt.figure(figsize=(8, 5))

\_ = plt.hist(train\_word\_num, bins=100)

plt.xlabel("word number")

plt.ylabel("Freq")

plt.show()



from wordcloud import WordCloud

plt.figure(figsize=(16, 10))

for ii in np.unique(train\_label):

text = np.array(traindata.train\_word[traindata.train\_label == ii])

text = ' '.join(np.concatenate(text))

plt.subplot(1, 2, ii + 1)

wordcod = WordCloud(font\_path='simhei.ttf', margin=5, width=1800, height=1000, max\_words=500, min\_font\_size=5,

background\_color='white',

max\_font\_size=250)

wordcod.generate\_from\_text(text)

plt.imshow(wordcod)

plt.axis("off")

if ii == 1:

plt.title("Positive")

else:

plt.title("Negative")

plt.subplots\_adjust(wspace=0.05)

plt.show()



# 课本代码导入模块没有给出代码导致，导入包即可

# Traceback (most recent call last):

# File "D:\pythoncode\learn\a\deep\_learning6.5.py", line 140, in <module>

# wordcod = WordCloud(margin=5, width=1800, height=1000, max\_words=500, min\_font\_size=5, max\_font\_size=250)

# NameError: name 'WordCloud' is not defined

# 字体报错 TrueType字体 TrueTypefonts .ttf文件 经过测试，字体路径是找到的，不然会报cannot open resource，

# wordcloud装了最新版1.9.2，降版本pip install wordcloud == 1.8.0 成功解决

# Traceback (most recent call last):

# File "D:\pythoncode\learn\a\deep\_learning6.5.py", line 142, in <module>

# wordcod.generate\_from\_text(text)

# File "D:\anaconda3\envs\deeplearning\lib\site-packages\wordcloud\wordcloud.py", line 621, in generate\_from\_text

# self.generate\_from\_frequencies(words)

# File "D:\anaconda3\envs\deeplearning\lib\site-packages\wordcloud\wordcloud.py", line 508, in generate\_from\_frequencies

# box\_size = draw.textbbox((0, 0), word, font=transposed\_font, anchor="lt")

# File "D:\anaconda3\envs\deeplearning\lib\site-packages\PIL\ImageDraw.py", line 651, in textbbox

# raise ValueError("Only supported for TrueType fonts")

# ValueError: Only supported for TrueType fonts

#

# 进程已结束,退出代码1

mytokenize = lambda x: x.split()

# tokenize='basic\_english'

TEXT = data.Field(sequential=True, tokenize=mytokenize,

include\_lengths=True, use\_vocab=True,

batch\_first=True, fix\_length=200)

LABEL = data.Field(sequential=False, use\_vocab=False,

pad\_token=None, unk\_token=None)

# 对所要读取的数据集的列进行处理

train\_test\_fields = [ # 踩坑，注意这里是按照顺序索引的，比如，它不会按照列名找到”label“，而是按照第一个就是TEXT

("text", TEXT),

("label", LABEL)

]

traindata, testdata = data.TabularDataset.splits(

path='./',

format='csv',

train='imdb\_train.csv', fields=train\_test\_fields,

test='imdb\_test.csv', skip\_header=True

)

# 打印数据集大小

print(len(traindata), len(testdata))

# 25000 25000

# ex0 = traindata.examples[0]

# print('ex0.label', ex0.label)

# print('ex0.text', ex0.text)

train\_data, val\_data = traindata.split(split\_ratio=0.7)

print(len(train\_data), len(val\_data))

# 17500 7500

vec = Vectors("glove.6B.100d.txt")

# 使用训练集构建单词表，导入预先训练的词嵌入

TEXT.build\_vocab(train\_data, max\_size=20000, vectors=vec)

LABEL.build\_vocab(train\_data)

# 训练集、验证集和测试集定义为迭代器

print(TEXT.vocab.freqs.most\_common(n=10))

# [('movie', 30061), ('film', 27361), ('one', 18244), ('like', 13721), ('good', 10390), ('would', 9348), ('even', 8852), ('time', 8475), ('really', 8274), ('story', 8239)]

print("词典的词数:", len(TEXT.vocab.itos))

# 词典的词数: 20002

print("前10个单词：", TEXT.vocab.itos[0:10])

# 前10个单词： ['<unk>', '<pad>', 'movie', 'film', 'one', 'like', 'good', 'would', 'even', 'time']

print("类别标签情况：", LABEL.vocab.freqs)

# 类别标签情况： Counter({'0': 8795, '1': 8705})

BATCH\_SIZE = 32

train\_iter = data.BucketIterator(train\_data, batch\_size=BATCH\_SIZE)

val\_iter = data.BucketIterator(val\_data, batch\_size=BATCH\_SIZE)

test\_iter = data.BucketIterator(testdata, batch\_size=BATCH\_SIZE)

for step, batch in enumerate(train\_iter):

if step > 0:

break

print("数据的类别标签", batch.label)

print("数据的尺寸", batch.text[0].shape)

print("数据样本数", batch.text[1])

# 数据的类别标签 tensor([1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,

# 1, 0, 1, 1, 1, 0, 0, 0])

# 数据的尺寸 torch.Size([32, 200])

# 数据样本数 tensor([ 55, 30, 96, 68, 129, 100, 33, 200, 156, 49, 39, 200, 200, 21,

# 165, 152, 62, 101, 33, 110, 68, 105, 31, 111, 107, 58, 109, 76,

# 101, 200, 68, 64])

class CNN\_Text(nn.Module):

def \_\_init\_\_(self, vocab\_size, embedding\_dim, n\_filters, filter\_sizes, output\_dim, dropout, pad\_idx):

super().\_\_init\_\_()

self.embedding = nn.Embedding(vocab\_size, embedding\_dim, padding\_idx=pad\_idx)

self.convs = nn.ModuleList([

nn.Conv2d(in\_channels=1, out\_channels=n\_filters,

kernel\_size=(fs, embedding\_dim)) for fs in filter\_sizes])

self.fc = nn.Linear(len(filter\_sizes) \* n\_filters, output\_dim)

self.dropout = nn.Dropout(dropout)

def forward(self, text):

embedded = self.embedding(text)

embedded = embedded.unsqueeze(1)

conved = [F.relu(conv(embedded)).squeeze(3) for conv in self.convs]

pooled = [F.max\_pool1d(conv, conv.shape[2]).squeeze(2) for conv in conved]

cat = self.dropout(torch.cat(pooled, dim=1))

return self.fc(cat)

INPUT\_DIM = len(TEXT.vocab)

EMBEDDING\_DIM = 100

N\_FILTERS = 100

FILTER\_SIZES = [3, 4, 5]

OUTPUT\_DIM = 1

DROPOUT = 0.5

PAD\_IDX = TEXT.vocab.stoi[TEXT.pad\_token]

model = CNN\_Text(INPUT\_DIM, EMBEDDING\_DIM, N\_FILTERS, FILTER\_SIZES, OUTPUT\_DIM, DROPOUT, PAD\_IDX)

print(model)

pretrained\_embeddings = TEXT.vocab.vectors

model.embedding.weight.data.copy\_(pretrained\_embeddings)

UNK\_IDX = TEXT.vocab.stoi[TEXT.unk\_token]

model.embedding.weight.data[UNK\_IDX] = torch.zeros(EMBEDDING\_DIM)

model.embedding.weight.data[PAD\_IDX] = torch.zeros(EMBEDDING\_DIM)

optimizer = optim.Adam(model.parameters())

criterion = nn.BCEWithLogitsLoss()

#

def train\_epoch(model, iterator, optimizer, criterion):

epoch\_loss = 0

epoch\_acc = 0

train\_corrects = 0

train\_num = 0

model.train()

for batch in iterator:

optimizer.zero\_grad()

pre = model(batch.text[0]).squeeze(1)

loss = criterion(pre, batch.label.type(torch.FloatTensor))

pre\_lab = torch.round(torch.sigmoid(pre))

train\_corrects += torch.sum(torch.tensor(pre\_lab.long() == batch.label)) # 括号内转为torch.tensor

train\_num += len(batch.label)

loss.backward()

optimizer.step()

epoch\_loss += loss.item()

epoch\_loss = epoch\_loss / train\_num

epoch\_acc = train\_corrects.double().item() / train\_num

return epoch\_loss, epoch\_acc

def evaluate(model, iterator, criterion):

epoch\_loss = 0

epoch\_acc = 0

train\_corrects = 0

train\_num = 0

model.eval()

with torch.no\_grad():

for batch in iterator:

pre = model(batch.text[0]).squeeze(1)

loss = criterion(pre, batch.label.type(torch.FloatTensor))

pre\_lab = torch.round(torch.sigmoid(pre))

train\_corrects += torch.sum(torch.tensor(pre\_lab.long() == batch.label))

train\_num += len(batch.label)

epoch\_loss += loss.item()

epoch\_loss = epoch\_loss / train\_num

epoch\_acc = train\_corrects.double().item() / train\_num

return epoch\_loss, epoch\_acc

EPOCHS = 10

best\_val\_loss = float('inf')

best\_acc = float(0)

for epoch in range(EPOCHS):

start\_time = time.time()

train\_loss, train\_acc = train\_epoch(model, train\_iter, optimizer, criterion)

val\_loss, val\_acc = evaluate(model, val\_iter, criterion)

end\_time = time.time()

print("Epoch:", epoch + 1, "|", "Epoch time", end\_time - start\_time, "s")

print("Train Loss:", train\_loss, "|", "Train Acc:", train\_acc)

print("Val. Loss:", val\_loss, "|", "Val. Acc:", val\_acc)

if (val\_loss < best\_val\_loss) & (val\_acc > best\_acc):

best\_model\_wts = copy.deepcopy(model.state\_dict())

best\_val\_loss = val\_loss

best\_acc = val\_acc

model.load\_state\_dict(best\_model\_wts)

test\_loss, test\_acc = evaluate(model, test\_iter, criterion)

print("在测试集上的预测精度", test\_acc)

# Epoch: 1 | Epoch time 62.18106269836426 s

# Train Loss: 0.014305785978691919 | Train Acc: 0.7802857142857142

# Val. Loss: 0.010499856820702553 | Val. Acc: 0.8602666666666666

# Epoch: 2 | Epoch time 64.33356475830078 s

# Train Loss: 0.008702067593591553 | Train Acc: 0.8852

# Val. Loss: 0.0092372254550457 | Val. Acc: 0.878

# Epoch: 3 | Epoch time 63.778724908828735 s

# Train Loss: 0.005293139319973332 | Train Acc: 0.9350857142857143

# Val. Loss: 0.009738493733604749 | Val. Acc: 0.8790666666666667

# Epoch: 4 | Epoch time 64.7396559715271 s

# Train Loss: 0.0028063043298731955 | Train Acc: 0.9686857142857143

# Val. Loss: 0.011253245748331149 | Val. Acc: 0.8785333333333334

# Epoch: 5 | Epoch time 66.96816277503967 s

# Train Loss: 0.0014695384308296654 | Train Acc: 0.9862857142857143

# Val. Loss: 0.013079128769785165 | Val. Acc: 0.8745333333333334

# Epoch: 6 | Epoch time 66.8894910812378 s

# Train Loss: 0.0007409520095946001 | Train Acc: 0.9937142857142857

# Val. Loss: 0.015568103588372469 | Val. Acc: 0.8732

# Epoch: 7 | Epoch time 64.34104418754578 s

# Train Loss: 0.0005200898583279923 | Train Acc: 0.9951428571428571

# Val. Loss: 0.017476340961021682 | Val. Acc: 0.8736

# Epoch: 8 | Epoch time 67.11497616767883 s

# Train Loss: 0.0003756954844608637 | Train Acc: 0.9958857142857143

# Val. Loss: 0.02083487620341281 | Val. Acc: 0.8597333333333333

# Epoch: 9 | Epoch time 65.55319833755493 s

# Train Loss: 0.00022720735681110195 | Train Acc: 0.9985142857142857

# Val. Loss: 0.021132646035403012 | Val. Acc: 0.8742666666666666

# Epoch: 10 | Epoch time 78.16548037528992 s

# Train Loss: 0.00020813046585618786 | Train Acc: 0.9982857142857143

# Val. Loss: 0.023343770688772202 | Val. Acc: 0.8709333333333333

# 在测试集上的预测精度 0.85108

#

# 进程已结束,退出代码0