# 实验5 CNN对话情绪识别

一、 实验目的

1）掌握CNN应用于情绪识别的方法。

2）学习用CNN网络建立情绪识别训练模型，并对模型进行评估。

二、 实验内容

对话情绪识别，目标是识别智能对话场景中用户的情绪，帮助企业更全面的把握产品体验、监控客户服务质量，适用于聊天、客服等多种场景。例如在智能音箱、智能车载等场景中，识别用户的情绪，可以适当地进行情绪安抚，改善产品的用户交互体验，在智能客服场景中，可以分析客服服务质量、降低人工质检成本，也能够帮助企业更好地把握对话质量、提高用户满意度。



从上图可以看到，对于用户的对话文本（通常是语音识别后的文本），模型会判断该文本属于不同情绪类别的概率，并给出最后的情绪类别，在本案例中，对话情绪类别有三种：负向情绪(0)、中性情绪(1)和正向情绪(2)，属于短文本三分类问题。

三、实验环境

深度学习框架：

Pytorch

Python 版本：

3.8

其它环境：

Windows10专业版（64bit）

GPU版本CUDA 10.2，且仅支持单卡

pip版本：20.2.2+（64bit）

四、算法流程

from \_\_future\_\_ import absolute\_import

from \_\_future\_\_ import division

from \_\_future\_\_ import print\_function

import os

import time

import paddle

import paddle.fluid as fluid

import numpy as np

import logging

import codecs

from work import model

from work import utils

from config import train\_parameters

logger = utils.logger

vocab = {}

def load\_vocab(file\_path):

"""

加载词典

"""

global vocab

with codecs.open(file\_path, 'r', encoding='utf8') as fin:

wid = 0

for line in fin:

if line.strip() not in vocab:

vocab[line.strip()] = wid

wid += 1

vocab["<unk>"] = len(vocab)

return vocab

def custom\_reader(file\_list, data\_dir):

"""

自定义用户数据读取器

:param file\_list:

:param data\_dir:

:param mode:

:return:

"""

unk\_id = len(vocab) - 1

file\_path = os.path.join(data\_dir, file\_list)

with codecs.open(file\_path, encoding='utf8') as flist:

lines = [line.strip() for line in flist]

def reader():

np.random.shuffle(lines)

for line in lines:

if line.startswith("label"):

continue

cols = line.strip().split("\t")

if len(cols) != 2:

logger.warning("error format {}".format(line))

continue

label = int(cols[0])

wids = [vocab[x] if x in vocab else unk\_id

for x in cols[1].split(" ")]

# logger.info("label:{} wids:{}".format(label, wids))

yield wids, label

return reader

def optimizer\_rms\_setting():

"""

阶梯型的学习率适合比较大规模的训练数据

"""

batch\_size = train\_parameters["train\_batch\_size"]

iters = train\_parameters["sample\_count"] // batch\_size

learning\_strategy = train\_parameters['rsm\_strategy']

lr = learning\_strategy['learning\_rate']

boundaries = [i \* iters for i in learning\_strategy["lr\_epochs"]]

values = [i \* lr for i in learning\_strategy["lr\_decay"]]

optimizer = fluid.optimizer.RMSProp(

learning\_rate=fluid.layers.piecewise\_decay(boundaries, values))

return optimizer

def optimizer\_sgd\_setting():

"""

loss下降相对较慢，但是最终效果不错，阶梯型的学习率适合比较大规模的训练数据

"""

learning\_strategy = train\_parameters['sgd\_strategy']

batch\_size = train\_parameters["train\_batch\_size"]

iters = train\_parameters["sample\_count"] // batch\_size

lr = learning\_strategy['learning\_rate']

boundaries = [i \* iters for i in learning\_strategy["lr\_epochs"]]

values = [i \* lr for i in learning\_strategy["lr\_decay"]]

learning\_rate = fluid.layers.piecewise\_decay(boundaries, values)

optimizer = fluid.optimizer.SGD(learning\_rate=learning\_rate)

return optimizer

def optimizer\_adam\_setting():

"""

能够比较快速的降低 loss，但是相对后期乏力

"""

learning\_strategy = train\_parameters['adam\_strategy']

learning\_rate = learning\_strategy['learning\_rate']

optimizer = fluid.optimizer.Adam(learning\_rate=learning\_rate)

return optimizer

def load\_params(exe, program):

if train\_parameters['continue\_train'] and os.path.exists(train\_parameters['save\_model\_dir']):

logger.info('load param from retrain model')

fluid.io.load\_persistables(executor=exe,

dirname=train\_parameters['save\_model\_dir'],

main\_program=program)

elif train\_parameters['pretrained'] and os.path.exists(train\_parameters['pretrained\_model\_dir']):

logger.info('load param from pretrained model')

def if\_exist(var):

return os.path.exists(os.path.join(train\_parameters['pretrained\_model\_dir'], var.name))

fluid.io.load\_vars(exe, train\_parameters['pretrained\_model\_dir'], main\_program=program,

predicate=if\_exist)

def train():

logger.info("start train text classification, train params: {}".format(train\_parameters))

logger.info("create place, use gpu: {}".format(train\_parameters['use\_gpu']))

load\_vocab(os.path.join(train\_parameters['data\_dir'], train\_parameters["vocabulary"]))

place = fluid.CUDAPlace(0) if train\_parameters['use\_gpu'] else fluid.CPUPlace()

logger.info("define input data tensor")

text = fluid.layers.data(name='text', shape=[1], dtype='int64', lod\_level=1)

label = fluid.layers.data(name='label', shape=[1], dtype='int64')

feeder = fluid.DataFeeder(feed\_list=[text, label], place=place)

logger.info("build custom reader")

reader = paddle.batch(custom\_reader(train\_parameters['train\_file\_list'], train\_parameters['data\_dir']),

batch\_size=train\_parameters['train\_batch\_size'])

logger.info("build network {}".format(train\_parameters['model\_type']))

if train\_parameters['model\_type'] == "cnn\_net":

network = model.cnn\_net

elif train\_parameters['model\_type'] == "bow\_net":

network = model.bow\_net

elif train\_parameters['model\_type'] == "lstm\_net":

network = model.lstm\_net

elif train\_parameters['model\_type'] == "bilstm\_net":

network = model.bilstm\_net

elif train\_parameters['model\_type'] == "gru\_net":

network = model.gru\_net

elif train\_parameters['model\_type'] == "textcnn\_net":

network = model.textcnn\_net

else:

raise ValueError("Unknown model type!")

avg\_cost, prediction = network(text, label, len(vocab), class\_dim=train\_parameters['class\_dim'])

# 选取不同的优化器

logger.info("build optimizer")

optimizer = optimizer\_adam\_setting()

optimizer.minimize(avg\_cost)

exe = fluid.Executor(place)

train\_program = fluid.default\_main\_program()

exe.run(fluid.default\_startup\_program())

load\_params(exe, train\_program)

acc\_top1 = fluid.layers.accuracy(input=prediction, label=label, k=1)

train\_fetch\_list = [avg\_cost.name, acc\_top1.name, prediction.name]

sample\_freq = train\_parameters['sample\_frequency']

current\_best\_acc = 0.0

for pass\_id in range(train\_parameters["num\_epochs"]):

logger.info("current pass: %d, start read text", pass\_id)

pass\_mean\_acc = 0.0

pass\_mean\_loss = 0.0

pass\_batch\_count = 0

for batch\_id, data in enumerate(reader()):

try:

t1 = time.time()

loss, acc\_top, prediction = exe.run(train\_program,

feed=feeder.feed(data),

fetch\_list=train\_fetch\_list)

period = time.time() - t1

prediction = np.array(prediction)

acc\_top = np.mean(np.array(acc\_top))

loss = np.mean(np.array(loss))

pass\_batch\_count += 1

pass\_mean\_acc += acc\_top

pass\_mean\_loss += loss

if batch\_id % sample\_freq == 0:

logger.info("pass {}, batch {}, accuracy:{} loss:{} time:{}"

.format(pass\_id, batch\_id, acc\_top, loss, "%2.2f sec" % period))

except Exception as e:

logger.error("train exception: ", e)

# 每训练完成一个批次，看下准确率，然后决定是否保存

pass\_loss = np.sum(pass\_mean\_loss) / pass\_batch\_count

pass\_acc = np.sum(pass\_mean\_acc) / pass\_batch\_count

logger.info("pass {} train result, current pass mean accuracy:{} loss:{}".format(pass\_id, pass\_acc, pass\_loss))

if pass\_acc > current\_best\_acc:

logger.info("temp save pass {} train result, current best pass mean accuracy:{}".format(pass\_id, pass\_acc))

fluid.io.save\_persistables(dirname=train\_parameters['save\_model\_dir'], main\_program=train\_program,

executor=exe)

current\_best\_acc = pass\_acc

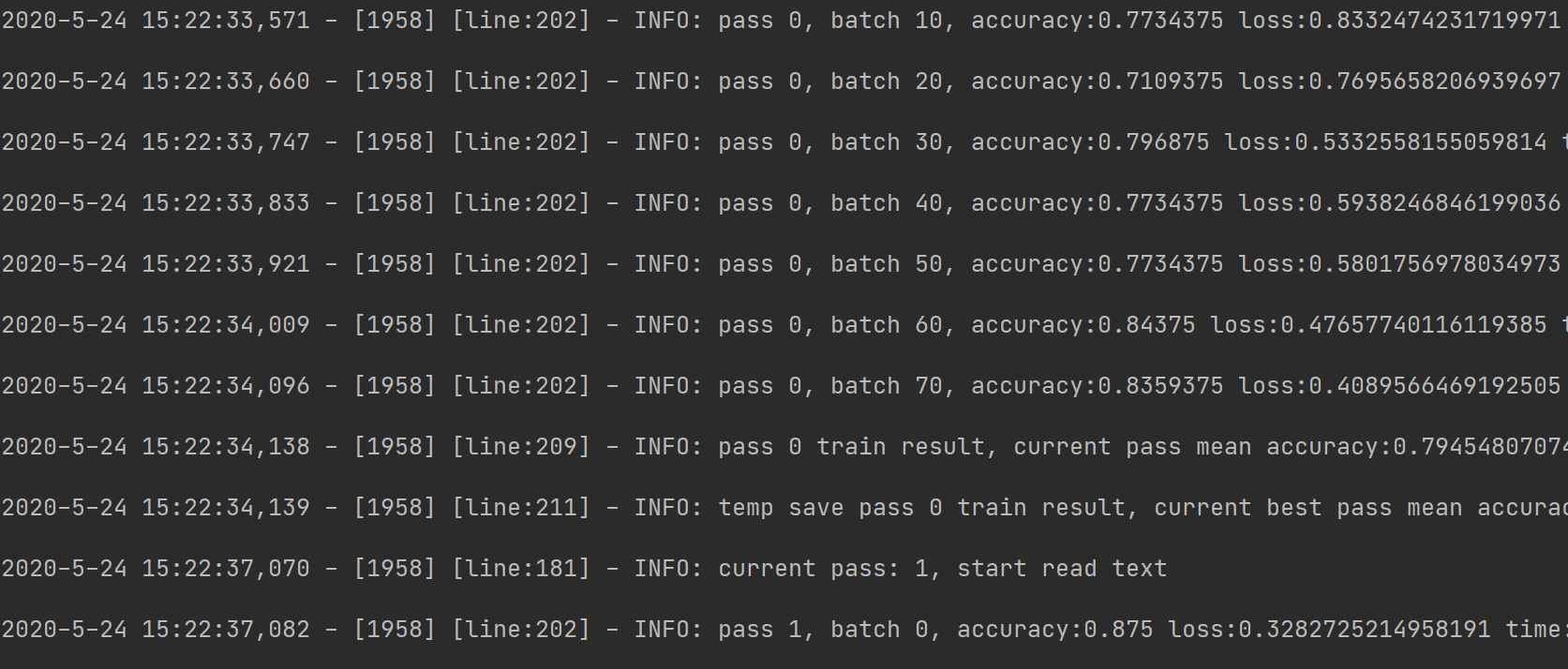
logger.info("training till last epcho, end training")

if \_\_name\_\_ == '\_\_main\_\_':

train()

**五、实验结果与分析**

运行结果截图



模型训练效果的好坏不仅与模型迭代的次数有关，而且与训练集的大小有关。分词的精确程度以及数据量的大小会直接影响的最终模型的预测效果。