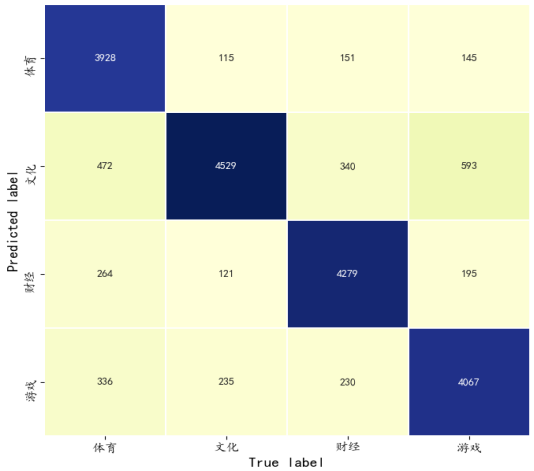
[Python人工智能] 二十八.Keras深度学习中文文本分类万字总结（CNN、TextCNN、LSTM、BiLSTM、BiLSTM+Attention）

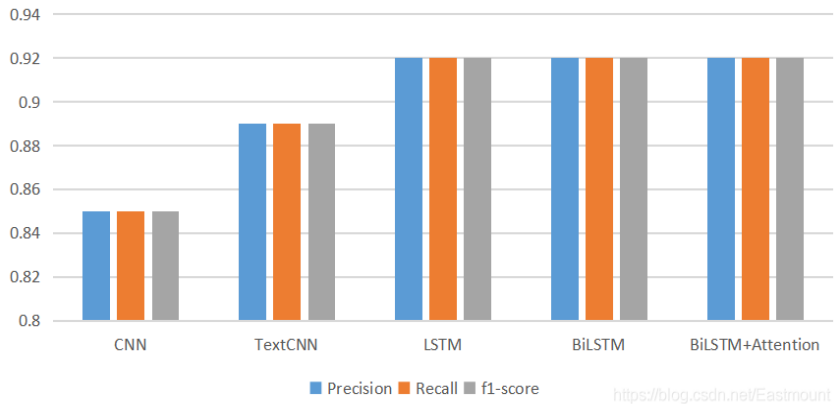
原创 Eastmount 2021-03-19 05:12:10 3397 已收藏 207 编辑 版权

分类专栏： Python+TensorFlow人工智能 文章标签： Python人工智能 BiLSTM 深度学习 Attention TextCNN 原力计划

从本专栏开始，作者正式研究Python深度学习、神经网络及人工智能相关知识。前一篇文章分享了BiLSTM-CRF模型搭建及训练、预测，最终实现医学命名实体识别实验。这篇文章将详细讲解Keras实现经典的深度学习文本分类算法，包括LSTM、BiLSTM、BiLSTM+Attention和CNN、TextCNN，这篇文章将以代码为主，让读者直观感受深度神经网络及对比实验。个人感觉还不错，基础性文章，希望对您有所帮助~

本次实验结果对比如下图所示：





本专栏主要结合作者之前的博客、AI经验和相关视频及论文介绍，后面随着深入会讲解更多的Python人工智能案例及应用。基础性文章，希望对您有所帮助，如果文章中存在错误或不足之处，还请海涵~作者作为人工智能的菜鸟，希望大家能与我在这一笔一划的博客中成长起来。写了这么多年博客，尝试第一个付费专栏，为小宝赚点奶粉钱，但更多博客尤其基础性文章，还是会继续免费分享，该专栏也会用心撰写，望对得起读者。当然如果您是在读学生或经济拮据，可以私聊我给你每篇文章开白名单，或者转发原文给你，更希望您能进步，一起加油喔！

> 注意，本文代码采用GPU+Pycharm实现，如果你的电脑是CPU实现，将相关GPU操作注释即可。这里仅做简单的对比实验，不进行参数优化、实验原因分析及详细的效果提升，后面文章会介绍优化、参数选择、实验评估等。

文章目录

- Keras下载地址：https://github.com/eastmountyxz/AI-for-Keras
- TensorFlow下载地址：https://github.com/eastmountyxz/AI-for-TensorFlow

同时推荐前面作者另外五个Python系列文章。从2014年开始，作者主要写了三个Python系列文章，分别是基础知识、网络爬虫和数据分析。2018年陆续增加了Python图像识别和Python人工智能专栏。

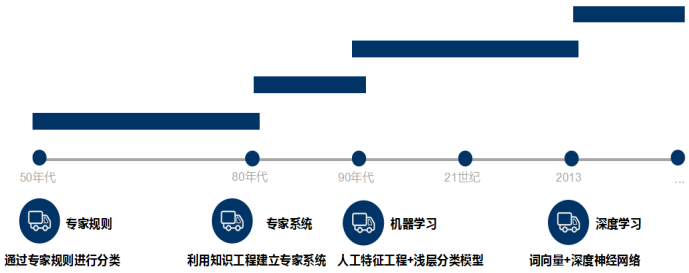
- Python基础知识系列：Python基础知识学习与提升
- Python网络爬虫系列：Python爬虫之Selenium+BeautifulSoup+Requests
- Python数据分析系列：知识图谱、web数据挖掘及NLP
- Python图像识别系列：Python图像处理及图像识别
- Python人工智能系列：Python人工智能及知识图谱实战

前文赏析：

- [Python人工智能] 一.TensorFlow2.0环境搭建及神经网络入门
- [Python人工智能] 二.TensorFlow基础及一元直线预测案例
- [Python人工智能] 三.TensorFlow基础之Session、变量、传入值和激励函数
- [Python人工智能] 四.TensorFlow创建回归神经网络及Optimizer优化器
- [Python人工智能] 五.Tensorboard可视化基本用法及绘制整个神经网络
- [Python人工智能] 六.TensorFlow实现分类学习及MNIST手写体识别案例
- [Python人工智能] 七.什么是过拟合及dropout解决神经网络中的过拟合问题
- [Python人工智能] 八.卷积神经网络CNN原理详解及TensorFlow编写CNN
- [Python人工智能] 九.gensim词向量Word2Vec安装及《庆余年》中文短文本相似度计算
- [Python人工智能] 十.Tensorflow+Opencv实现CNN自定义图像分类案例及与机器学习KNN图像分类算法对比
- [Python人工智能] 十一.Tensorflow如何保存神经网络参数
- [Python人工智能] 十二.循环神经网络RNN和LSTM原理详解及TensorFlow编写RNN分类案例
- [Python人工智能] 十三.如何评价神经网络、loss曲线图绘制、图像分类案例的F值计算
- [Python人工智能] 十四.循环神经网络LSTM RNN回归案例之sin曲线预测
- [Python人工智能] 十五.无监督学习Autoencoder原理及聚类可视化案例详解
- [Python人工智能] 十六.Keras环境搭建、入门基础及回归神经网络案例
- [Python人工智能] 十七.Keras搭建分类神经网络及MNIST数字图像案例分析
- [Python人工智能] 十八.Keras搭建卷积神经网络及CNN原理详解
  [Python人工智能] 十九.Keras搭建循环神经网络分类案例及RNN原理详解
- [Python人工智能] 二十.基于Keras+RNN的文本分类vs基于传统机器学习的文本分类
- [Python人工智能] 二十一.Word2Vec+CNN中文文本分类详解及与机器学习（RF\DTC\SVM\KNN\NB\LR）分类对比
- [Python人工智能] 二十二.基于大连理工情感词典的情感分析和情绪计算
- [Python人工智能] 二十三.基于机器学习和TFIDF的情感分类（含详细的NLP数据清洗）
- [Python人工智能] 二十四.易学智能GPU搭建Keras环境实现LSTM恶意URL请求分类
- [Python人工智能] 二十六.基于BiLSTM-CRF的医学命名实体识别研究（上）数据预处理
- [Python人工智能] 二十七.基于BiLSTM-CRF的医学命名实体识别研究（下）模型构建
- [Python人工智能] 二十八.Keras深度学习中文文本分类万字总结（CNN、TextCNN、LSTM、BiLSTM、BiLSTM+Attention）
  《人工智能狂潮》读后感——什么是人工智能？(一)

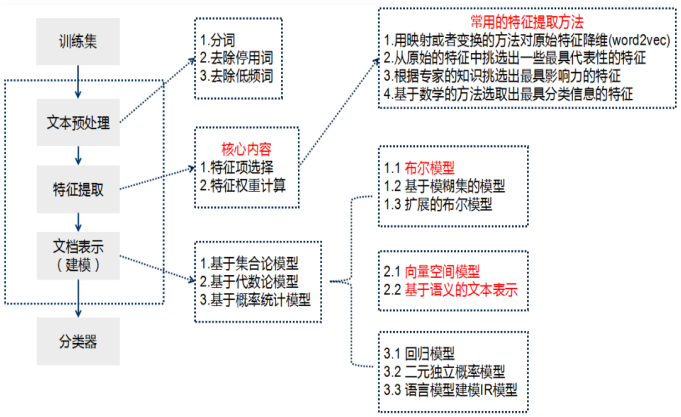
一.文本分类概述

文本分类旨在对文本集按照一定的分类体系或标准进行自动分类标记，属于一种基于分类体系的自动分类。文本分类最早可以追溯到上世纪50年代，那时主要通过专家定义规则来进行文本分类；80年代出现了利用知识工程建立的专家系统；90年代开始借助于机器学习方法，通过人工特征工程和浅层分类模型来进行文本分类。**现在多采用词向量以及深度神经网络来进行文本分类。**



牛亚峰老师将传统的文本分类流程归纳如下图所示。在传统的文本分类中，基本上大部分机器学习方法都在文本分类领域有所应用。主要包括：

- Naive Bayes
- KNN
- SVM
- 集合类方法
- 最大熵
- 神经网络



利用Keras框架进行文本分类的基本流程如下：

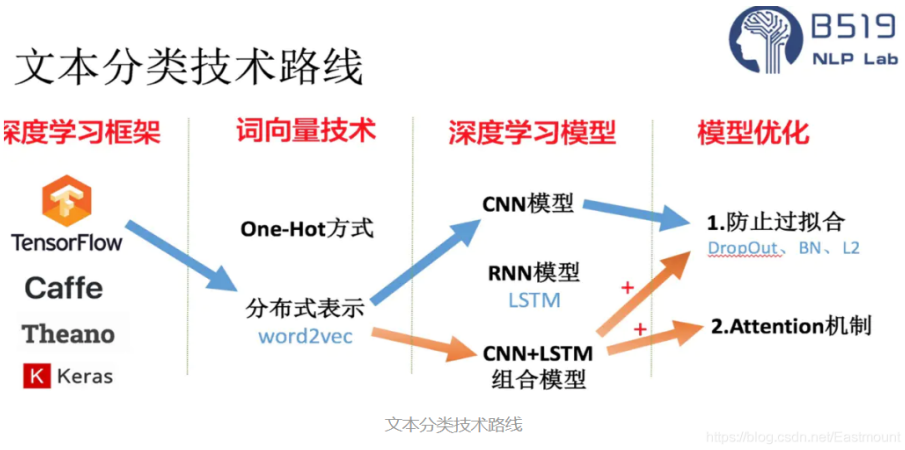
- 步骤 1：文本的预处理，分词->去除停用词->统计选择top n的词做为特征词
- 步骤 2：为每个特征词生成ID
- 步骤 3：将文本转化成ID序列，并将左侧补齐
- 步骤 4：训练集shuffle
- 步骤 5：Embedding Layer 将词转化为词向量
- 步骤 6：添加模型，构建神经网络结构
- 步骤 7：训练模型
- 步骤 8：得到准确率、召回率、F1值

注意，如果使用TFIDF而非词向量进行文档表示，则直接分词去停后生成TFIDF矩阵后输入模型。

深度学习文本分类方法包括：

- 卷积神经网络(TextCNN)
- 循环神经网络(TextRNN)
- TextRNN+Attention

- TextRCNN(TextRNN+CNN)
- BiLSTM+Attention
- 迁移学习



文本分类技术路线

推荐牛亚峰老师的文章：

- 基于 word2vec 和 CNN 的文本分类： 综述 & 实践

二.数据预处理及分词

这篇文章主要以代码为主，算法原理知识前面的文章和后续文章再继续补充。数据集如下图所示：

- **训练集**：news_dataset_train.csv
  游戏主题（10000）、体育主题（10000）、文化主题（10000）、财经主题（10000）
- **测试集**：news_dataset_test.csv
  游戏主题（5000）、体育主题（5000）、文化主题（5000）、财经主题（5000）
- **验证集**：news_dataset_val.csv
  游戏主题（5000）、体育主题（5000）、文化主题（5000）、财经主题（5000）

label	content
体育	做引体向上时，使不出劲怎么办？
体育	做引体向上时，使不出劲怎么办？
文化	做艺术家难，难在艺术家要不断的否定自己，对于这样的观点大家是怎么理解的？
财经	做一个苗木农场，有资源没有资金，怎样才能融资？
游戏	做为游戏玩家，怎样通过人工智能的辅助提高成绩？
财经	做期货的人会考虑反跟单吗？
文化	做梦都想住的中式大宅，只是饱饱眼福，欣赏欣赏就觉得好幸福了
体育	做梦都想得到“詹姆斯”
财经	做股票不需要太多的聪明，愚蠢的人却都在找捷径
财经	做电子商务选择甘肃扬帆起航的八大理由
财经	坐在自行车后座哭泣
文化	作文写作的方法技巧都在这里了！
游戏	作为游戏主播，是精通一个游戏好，还是精通多个游戏好呢？
文化	作为一个普通开封人来到北京故宫，他不看文物只关注建筑色彩线条
体育	作为央视体育体育频道，CCTV5一到周末就直播马拉松你怎么看？
体育	作为喜欢足球的上海人，你是如何在上港、申花、申鑫之间作出选择的？
文化	作为上海人或新上海人，你能写出多少上海闲话？
游戏	作为电子竞技的战争：玩家未知战场的政治
财经	作为曾经的全国工业明星城市，常州未来走向何方？
文化	作为infp型人，你对于慎独能够做到什么程度？

首先需要进行中文分词预处理，调用Jieba库实现。代码如下：

- **data_preprocess.py**

```
# -*- coding:utf-8 -*-
# By:Eastmount CSDN 2021-03-19
import csv
import pandas as pd
import numpy as np
import jieba
import jieba.analyse

#添加自定义词典和停用词典
jieba.load_userdict("user_dict.txt")
stop_list = pd.read_csv('stop_words.txt',
                        engine='python',
                        encoding='utf-8',
                        delimiter="\n",
                        names=['t'])['t'].tolist()

#------------------------------------------------------------------------
#Jieba分词函数
def txt_cut(juzi):
    return [w for w in jieba.lcut(juzi) if w not in stop_list]

#------------------------------------------------------------------------
#中文分词读取文件
def fenci(filename,result):
    #写入分词结果
    fw = open(result, "w", newline = '',encoding = 'gb18030')
    writer = csv.writer(fw)
    writer.writerow(['label','cutword'])

    #使用csv.DictReader读取文件中的信息
    labels = []
    contents = []
    with open(filename, "r", encoding="UTF-8") as f:
        reader = csv.DictReader(f)
        for row in reader:
            #数据元素获取
            labels.append(row['label'])
            content = row['content']
            #中文分词
            seglist = txt_cut(content)
            #空格拼接
            output = ' '.join(list(seglist))
            contents.append(output)

            #文件写入
            tlist = []
            tlist.append(row['label'])
            tlist.append(output)
            writer.writerow(tlist)
    print(labels[:5])
    print(contents[:5])
    fw.close()

#------------------------------------------------------------------------
#主函数
if __name__ == '__main__':
    fenci("news_dataset_train.csv", "news_dataset_train_fc.csv")
    fenci("news_dataset_test.csv", "news_dataset_test_fc.csv")
    fenci("news_dataset_val.csv", "news_dataset_val_fc.csv")
```

运行结果如下图所示：

```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\xiuzhang\AppData\Local\Temp\jieba.cache
Loading model cost 0.883 seconds.
Prefix dict has been built succesfully.
['文化', '文化', '游戏', '游戏', '游戏']
['上句 浊水 三江 一夜 雨 接句', '看待 曹德旺 说 一百对 夫妻 中 一对 幸福 这句 话', '绝地 求生 蓝洞 反 外挂 措施 效果显著  外挂 制作者 跑 路  游戏 体验 提升', 'uzi 多久 职业赛', '炉石 听说 3 重点 玩家 15 分钟 战胜 哈加 莎']
['游戏', '游戏', '游戏', '游戏', '游戏']
['无双 大蛇 3 世界观 登场 角色 新 系统 图文 介绍  游戏', '30 分钟 死亡 LMS 战区 首次 击败 LPL', '王者 荣耀  接吻 赛后 诸葛亮 采访 赵云 关系 周瑜 笔下 小乔', '英雄 联盟 新 英雄 血港 开膛手 登场 厄 加特 终于 最丑', 'adc 满血 开大 剑圣']
['体育', '体育', '财经', '财经', '财经']
['做 引体向上 时 胸口 触杆 训练 动作', '做 引体向上 时 使不出 劲', '做 投资 概率 确定性', '做声 屏障 企业 那家 求 推荐', '做 期货 钱']
>>>
```

By: Eastmount CSDN

label	cutword
游戏	无双 大蛇 3 世界观 登场 角色 新 系统 图文 介绍 游戏
游戏	30 分钟 死亡 LMS 战区 首次 击败 LPL
游戏	王者 荣耀 接吻 赛后 诸葛亮 采访 赵云 关系 周瑜 笔下 小乔
游戏	英雄 联盟 新 英雄 血港 开膛手 登场 厄 加特 终于 最丑
游戏	adc 满血 开大 剑圣
游戏	刺激 战场 仓库 楼顶 教学 98K 三级 包 四倍 镜全
游戏	王者 荣耀 五黑 排位 中 感到 绝望 组合
游戏	DNF 肝 新 职业 平衡性 改版 全 职业 预约 活动 来袭
游戏	球球 作战 狂神 莫欺 小球 穷 活着 慢慢 反杀 第一
游戏	M4 最强 满配 步枪 冲锋枪 狙击枪 武器
游戏	王者 荣耀 上单 最强 5 英雄 盘点 宫本 武藏 垫底 前 2 名竟 全是 坦克
游戏	CF 手游 单挑 想 赤龙 火炮 恶心 死 姜 老的辣
游戏	中国 玩家 贡献 2 亿美元 救 不活 魔兽 电影 制作方 拍卖 电影 道具
游戏	先来场 比赛 暖暖身 全名 QSL 锦标赛 大乱斗
游戏	王者 荣耀 里 貂蝉 克星
游戏	5 月 12 日 RNGM 五五开 队 YTG 终结 24 连胜
游戏	海贼王 尾田 早就 暗示 霸王 色变 鸡肋 方法

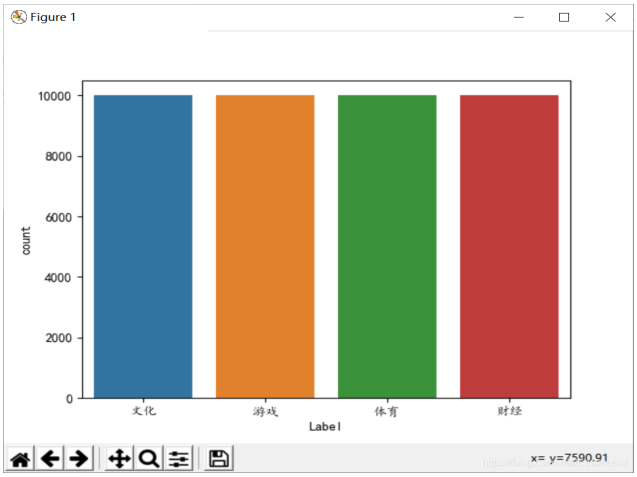
接着我们尝试简单查看数据的长度分布情况及标签可视化。

- **data_show.py**

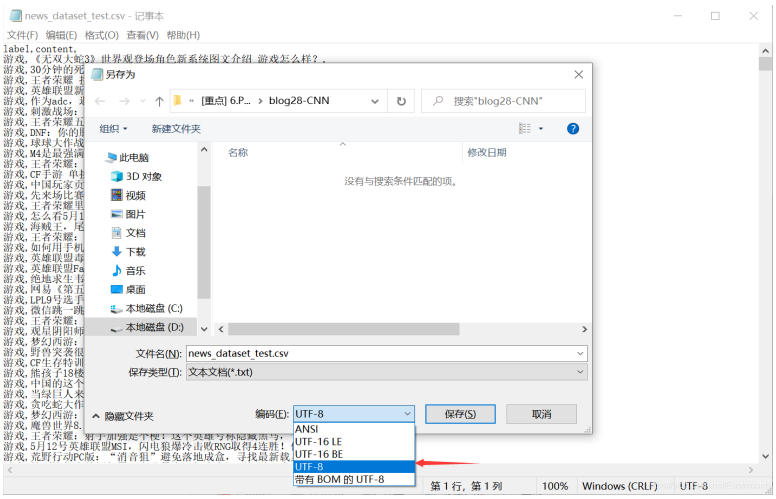
```
# -*- coding: utf-8 -*-
"""
Created on 2021-03-19
@author: xiuzhang Eastmount CSDN
"""
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns

#---------------------------------------第一步 数据读取-----------------------------------
## 读取测数据集
train_df = pd.read_csv("news_dataset_train_fc.csv")
val_df = pd.read_csv("news_dataset_val_fc.csv")
test_df = pd.read_csv("news_dataset_test_fc.csv")
print(train_df.head())

## 解决中文显示问题
plt.rcParams['font.sans-serif'] = ['KaiTi']  #指定默认字体 SimHei黑体
plt.rcParams['axes.unicode_minus'] = False   #解决保存图像是负号'

## 查看训练集都有哪些标签
plt.figure()
sns.countplot(train_df.label)
```

```
plt.xlabel('Label',size = 10)
plt.xticks(size = 10)
plt.show()

## 分析训练集中词组数量的分布
print(train_df.cutwordnum.describe())
plt.figure()
plt.hist(train_df.cutwordnum,bins=100)
plt.xlabel("词组长度", size = 12)
plt.ylabel("频数", size = 12)
plt.title("训练数据集")
plt.show()
```

输出结果如下图所示，后面的文章我们会介绍论文如何绘制好看的图表。



> 注意，如果报错“UnicodeDecodeError: ‘utf-8’ codec can’t decode byte 0xce in position 17: invalid continuation byte”，需要将CSV文件保存为UTF-8格式，如下图所示。



三.CNN中文文本分类

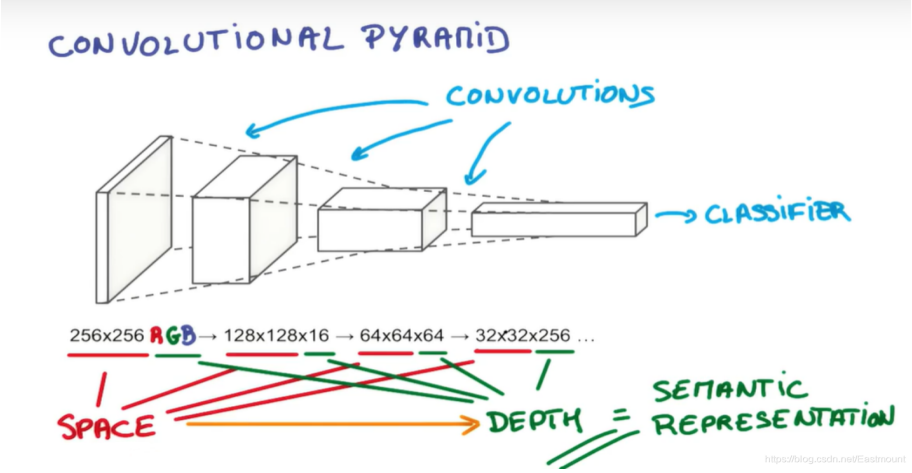
1.原理介绍

卷积神经网络（Convolutional Neural Networks, CNN）是一类包含卷积计算且具有深度结构的前馈神经网络（Feedforward Neural Networks），是深度学习（deep learning）的代表算法之一。它通常应用于图像识别和语音识等领域，并能给出更优秀的结果，也可以应用于视频分析、机器翻译、自然语言处理、药物发现等领域。著名的阿尔法狗让计算机看懂围棋就是基于卷积神经网络的。

- 卷积是指不在对每个像素做处理，而是对图片区域进行处理，这种做法加强了图片的连续性，看到的是一个图形而不是一个点，也加深了神经网络对图片的理解。
- **通常卷积神经网络会依次经历“图片->卷积->持化->卷积->持化->结果传入两层全连接神经层->分类器”的过程，最终实现一个CNN的分类处理。**

推荐作者前文：

- [Python人工智能] 十八.Keras搭建卷积神经网络及CNN原理详解



2.代码实现

Keras实现文本分类的CNN代码如下：

- **Keras_CNN_cnews.py**

```
# -*- coding: utf-8 -*-
"""
Created on 2021-03-19
@author: xiuzhang Eastmount CSDN
CNN Model
"""
import os
import time
import pickle
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding
from keras.layers import Convolution1D, MaxPool1D, Flatten
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.callbacks import EarlyStopping
```

```
from keras.models import load_model
from keras.models import Sequential

## GPU处理 读者如果是CPU注释该部分代码即可
## 指定每个GPU进程中使用显存的上限 0.9表示可以使用GPU 90%的资源进行训练
os.environ["CUDA_DEVICES_ORDER"] = "PCI_BUS_IS"
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.8)
sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))

start = time.clock()

#----------------------------第一步 数据读取----------------------------
## 读取测数据集
train_df = pd.read_csv("news_dataset_train_fc.csv")
val_df = pd.read_csv("news_dataset_val_fc.csv")
test_df = pd.read_csv("news_dataset_test_fc.csv")
print(train_df.head())

## 解决中文显示问题
plt.rcParams['font.sans-serif'] = ['KaiTi']  #指定默认字体 SimHei黑体
plt.rcParams['axes.unicode_minus'] = False   #解决保存图像是负号'

#--------------------------第二步 OneHotEncoder()编码--------------------
## 对数据集的标签数据进行编码
train_y = train_df.label
val_y = val_df.label
test_y = test_df.label
print("Label:")
print(train_y[:10])

le = LabelEncoder()
train_y = le.fit_transform(train_y).reshape(-1,1)
val_y = le.transform(val_y).reshape(-1,1)
test_y = le.transform(test_y).reshape(-1,1)
print("LabelEncoder")
print(train_y[:10])
print(len(train_y))

## 对数据集的标签数据进行one-hot编码
ohe = OneHotEncoder()
train_y = ohe.fit_transform(train_y).toarray()
val_y = ohe.transform(val_y).toarray()
test_y = ohe.transform(test_y).toarray()
print("OneHotEncoder:")
print(train_y[:10])

#-----------------------第三步 使用Tokenizer对词组进行编码--------------------
max_words = 6000
max_len = 600
tok = Tokenizer(num_words=max_words)  #最大词语数为6000
print(train_df.cutword[:5])
print(type(train_df.cutword))

## 防止语料中存在数字str处理
train_content = [str(a) for a in train_df.cutword.tolist()]
val_content = [str(a) for a in val_df.cutword.tolist()]
test_content = [str(a) for a in test_df.cutword.tolist()]
tok.fit_on_texts(train_content)
print(tok)

#当创建Tokenizer对象后 使用fit_on_texts()函数识别每个词
```

```
#tok.fit_on_texts(train_df.cutword)

## 保存训练好的Tokenizer和导入
with open('tok.pickle', 'wb') as handle: #saving
    pickle.dump(tok, handle, protocol=pickle.HIGHEST_PROTOCOL)
with open('tok.pickle', 'rb') as handle: #loading
    tok = pickle.load(handle)

## 使用word_index属性查看每个词对应的编码
## 使用word_counts属性查看每个词对应的频数
for ii,iterm in enumerate(tok.word_index.items()):
    if ii < 10:
        print(iterm)
    else:
        break
print("====================")
for ii,iterm in enumerate(tok.word_counts.items()):
    if ii < 10:
        print(iterm)
    else:
        break

#--------------------------第四步 数据转化为序列-----------------------------
## 使用sequence.pad_sequences()将每个序列调整为相同的长度
## 对每个词编码之后，每句新闻中的每个词就可以用对应的编码表示，即每条新闻可以转变成一个向量了
train_seq = tok.texts_to_sequences(train_content)
val_seq = tok.texts_to_sequences(val_content)
test_seq = tok.texts_to_sequences(test_content)

## 将每个序列调整为相同的长度
train_seq_mat = sequence.pad_sequences(train_seq,maxlen=max_len)
val_seq_mat = sequence.pad_sequences(val_seq,maxlen=max_len)
test_seq_mat = sequence.pad_sequences(test_seq,maxlen=max_len)
print("数据转换序列")
print(train_seq_mat.shape)
print(val_seq_mat.shape)
print(test_seq_mat.shape)
print(train_seq_mat[:2])

#-------------------------------第五步 建立CNN模型-------------------------
## 类别为4个
num_labels = 4
inputs = Input(name='inputs',shape=[max_len], dtype='float64')
## 词嵌入使用预训练的词向量
layer = Embedding(max_words+1, 128, input_length=max_len, trainable=False)(inputs)
## 卷积层和池化层(词窗大小为3 128核)
cnn = Convolution1D(128, 3, padding='same', strides = 1, activation='relu')(layer)
cnn = MaxPool1D(pool_size=4)(cnn)
## Dropout防止过拟合
flat = Flatten()(cnn)
drop = Dropout(0.3)(flat)
## 全连接层
main_output = Dense(num_labels, activation='softmax')(drop)
model = Model(inputs=inputs, outputs=main_output)

## 优化函数 评价指标
model.summary()
model.compile(loss="categorical_crossentropy",
              optimizer='adam',      # RMSprop()
              metrics=["accuracy"])

#-------------------------------第六步 模型训练和预测-------------------------
```

```
## 先设置为train训练 再设置为test测试
flag = "train"
if flag == "train":
    print("模型训练")
    ## 模型训练 当val-loss不再提升时停止训练 0.0001
    model_fit = model.fit(train_seq_mat, train_y, batch_size=128, epochs=10,
                          validation_data=(val_seq_mat,val_y),
                          callbacks=[EarlyStopping(monitor='val_loss',min_delta=0.0001)]
                         )
    ## 保存模型
    model.save('my_model.h5')
    del model  # deletes the existing model
    ## 计算时间
    elapsed = (time.clock() - start)
    print("Time used:", elapsed)
    print(model_fit.history)

else:
    print("模型预测")
    ## 导入已经训练好的模型
    model = load_model('my_model.h5')
    ## 对测试集进行预测
    test_pre = model.predict(test_seq_mat)
    ## 评价预测效果，计算混淆矩阵
    confm = metrics.confusion_matrix(np.argmax(test_y,axis=1),np.argmax(test_pre,axis=1))
    print(confm)

    ## 混淆矩阵可视化
    Labname = ["体育", "文化", "财经", "游戏"]
    print(metrics.classification_report(np.argmax(test_y,axis=1),np.argmax(test_pre,axis=1)))
    plt.figure(figsize=(8,8))
    sns.heatmap(confm.T, square=True, annot=True,
                fmt='d', cbar=False, linewidths=.6,
                cmap="YlGnBu")
    plt.xlabel('True label',size = 14)
    plt.ylabel('Predicted label', size = 14)
    plt.xticks(np.arange(4)+0.5, Labname, size = 12)
    plt.yticks(np.arange(4)+0.5, Labname, size = 12)
    plt.savefig('result.png')
    plt.show()

    #----------------------------------第七 验证算法-------------------------
    ## 使用tok对验证数据集重新预处理，并使用训练好的模型进行预测
    val_seq = tok.texts_to_sequences(val_df.cutword)
    ## 将每个序列调整为相同的长度
    val_seq_mat = sequence.pad_sequences(val_seq,maxlen=max_len)
    ## 对验证集进行预测
    val_pre = model.predict(val_seq_mat)
    print(metrics.classification_report(np.argmax(val_y,axis=1),np.argmax(val_pre,axis=1)))

    ## 计算时间
    elapsed = (time.clock() - start)
    print("Time used:", elapsed)
```

GPU运行如下图所示。注意，如果您的电脑是CPU版本，只需要将上述代码第一部分注释掉即可，后面LSTM部分使用GPU对应的库函数。

```
C:\Anaconda3\envs\py36h\python.exe M:/blog28-CNN/Keras_CNN_cnews.py
Using TensorFlow backend.
2021-03-19 03:31:14.190286: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow
2021-03-19 03:31:14.957573: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 0 with properties:
name: GeForce GTX 1080 Ti major: 6 minor: 1 memoryClockRate(GHz): 1.683
pciBusID: 0000:00:09.0
totalMemory: 11.00GiB freeMemory: 9.11GiB
2021-03-19 03:31:14.964738: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu devices: 0
```
https://blog.csdn.net/Eastmount

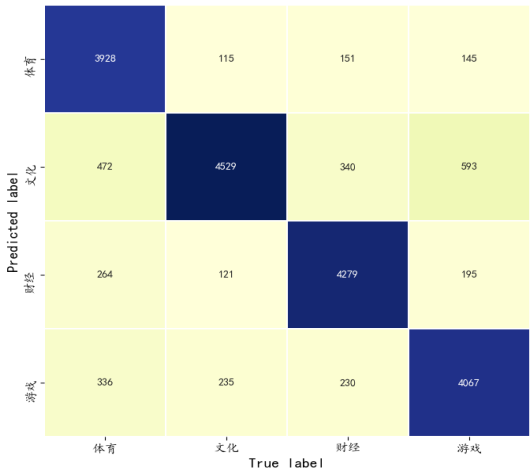
训练输出模型如下图所示：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inputs (InputLayer)          (None, 600)               0
_________________________________________________________________
embedding_1 (Embedding)      (None, 600, 128)          768128
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 600, 128)          49280
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 150, 128)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 19200)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 19200)             0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 76804
=================================================================
Total params: 894,212
Trainable params: 126,084
Non-trainable params: 768,128
```
https://blog.csdn.net/Eastmount

训练输出结果如下：

```
模型训练
Train on 40000 samples, validate on 20000 samples
Epoch 1/10
40000/40000 [==============================] - 15s 371us/step - loss: 1.1798 - acc: 0.4772 - val_loss: 0.
9878 - val_acc: 0.5977
Epoch 2/10
40000/40000 [==============================] - 4s 93us/step - loss: 0.8681 - acc: 0.6612 - val_loss: 0.81
67 - val_acc: 0.6746
Epoch 3/10
40000/40000 [==============================] - 4s 92us/step - loss: 0.7268 - acc: 0.7245 - val_loss: 0.70
84 - val_acc: 0.7330
Epoch 4/10
40000/40000 [==============================] - 4s 93us/step - loss: 0.6369 - acc: 0.7643 - val_loss: 0.64
62 - val_acc: 0.7617
Epoch 5/10
40000/40000 [==============================] - 4s 96us/step - loss: 0.5670 - acc: 0.7957 - val_loss: 0.58
95 - val_acc: 0.7867
Epoch 6/10
40000/40000 [==============================] - 4s 92us/step - loss: 0.5074 - acc: 0.8226 - val_loss: 0.55
30 - val_acc: 0.8018
Epoch 7/10
40000/40000 [==============================] - 4s 93us/step - loss: 0.4638 - acc: 0.8388 - val_loss: 0.51
05 - val_acc: 0.8185
Epoch 8/10
40000/40000 [==============================] - 4s 93us/step - loss: 0.4241 - acc: 0.8545 - val_loss: 0.48
36 - val_acc: 0.8304
Epoch 9/10
40000/40000 [==============================] - 4s 92us/step - loss: 0.3900 - acc: 0.8692 - val_loss: 0.45
99 - val_acc: 0.8403
Epoch 10/10
40000/40000 [==============================] - 4s 93us/step - loss: 0.3657 - acc: 0.8761 - val_loss: 0.44
72 - val_acc: 0.8457
Time used: 52.203992899999996
```

预测及验证结果如下：




```
[[3928  472  264  336]
 [ 115 4529  121  235]
 [ 151  340 4279  230]
 [ 145  593  195 4067]]
             precision    recall  f1-score   support

          0       0.91      0.79      0.84      5000
          1       0.76      0.91      0.83      5000
          2       0.88      0.86      0.87      5000
          3       0.84      0.81      0.82      5000

avg / total       0.85      0.84      0.84     20000

             precision    recall  f1-score   support

          0       0.90      0.77      0.83      5000
          1       0.78      0.92      0.84      5000
          2       0.88      0.85      0.86      5000
          3       0.84      0.85      0.85      5000

avg / total       0.85      0.85      0.85     20000
```

四.TextCNN中文文本分类

1.原理介绍

TextCNN 是利用卷积神经网络对文本进行分类的算法，由 Yoon Kim 于2014年在 “Convolutional Neural Networks for Sentence Classification” 一文中提出的算法。

卷积神经网络的核心思想是捕捉局部特征，对于文本来说，局部特征就是由若干单词组成的滑动窗口，类似于N-gram。卷积神经网络的优势在于能够自动地对N-gram特征进行组合和筛选，获得不同抽象层次的语义信息。下图是该论文中用于文本分类的卷积神经网络模型架构。

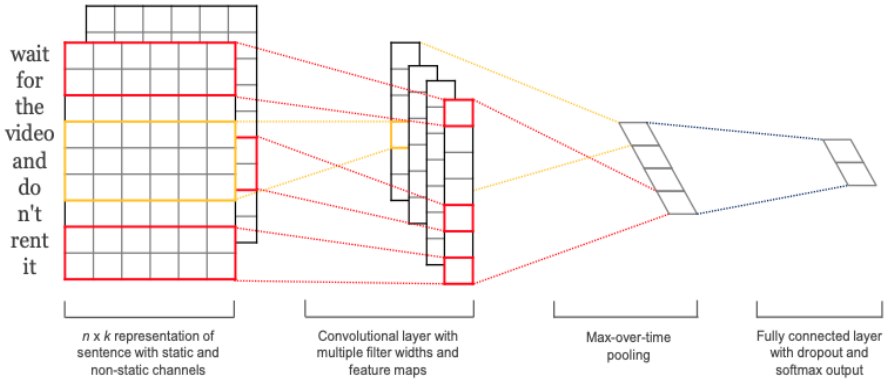
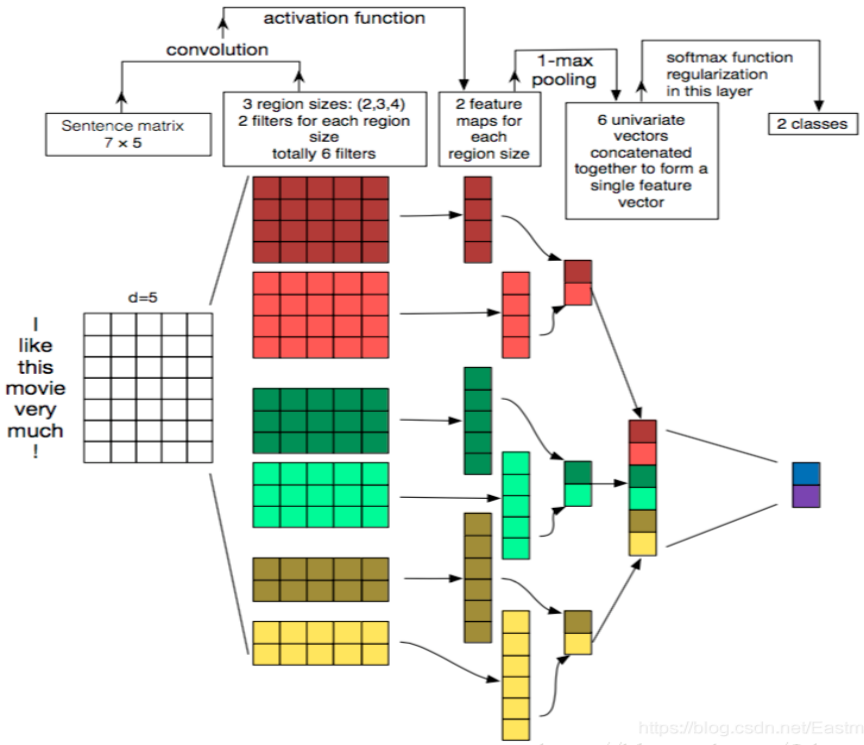


Figure 1: Model architecture with two channels for an example sentence.

另一篇TextCNN比较经典的论文是《A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification》，其模型结果如下图所示。主要用于文本分类任务的TextCNN结构描述，详细解释了TextCNN架构及词向量矩阵是如何做卷积的。



假设我们有一些句子需要对其进行分类。句子中每个词是由n维词向量组成的，也就是说输入矩阵大小为m*n，其中m为句子长度。CNN需要对输入样本进行卷积操作，对于文本数据，filter不再横向滑动，仅仅是向下移动，有点类似于N-gram在提取词与词间的局部相关性。

图中共有三种步长策略，分别是2、3、4，每个步长都有两个filter（实际训练时filter数量会很多）。在不同词窗上应用不同filter，最终得到6个卷积后的向量。然后对每一个向量进行最大化池化操作并拼接各个池化值，最终得到这个句子的特征表示，将这个句子向量丢给分类器进行分类，最终完成整个文本分类流程。

> 最后真心推荐下面这些大佬关于TextCNN的介绍，尤其是CSDN的Asia-Lee大佬，很喜欢他的文章，真心棒！
>
> - 《Convolutional Neural Networks for Sentence Classification2014》
> - 《A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification》
> - https://blog.csdn.net/asialee_bird/article/details/88813385
> - https://zhuanlan.zhihu.com/p/77634533

2.代码实现

Keras实现文本分类的TextCNN代码如下：

- **Keras_TextCNN_cnews.py**

```
# -*- coding: utf-8 -*-
"""
Created on 2021-03-19
@author: xiuzhang Eastmount CSDN
TextCNN Model
"""
import os
import time
import pickle
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding
from keras.layers import Convolution1D, MaxPool1D, Flatten
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.callbacks import EarlyStopping
from keras.models import load_model
from keras.models import Sequential
from keras.layers.merge import concatenate

## GPU处理 读者如果是CPU注释该部分代码即可
## 指定每个GPU进程中使用显存的上限 0.9表示可以使用GPU 90%的资源进行训练
os.environ["CUDA_DEVICES_ORDER"] = "PCI_BUS_IS"
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.8)
sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))

start = time.clock()

#----------------------------第一步 数据读取----------------------------
## 读取测数据集
train_df = pd.read_csv("news_dataset_train_fc.csv")
val_df = pd.read_csv("news_dataset_val_fc.csv")
test_df = pd.read_csv("news_dataset_test_fc.csv")

## 解决中文显示问题
plt.rcParams['font.sans-serif'] = ['KaiTi']  #指定默认字体 SimHei黑体
plt.rcParams['axes.unicode_minus'] = False   #解决保存图像是负号'
```

```
#-------------------------第二步 OneHotEncoder()编码-------------------
## 对数据集的标签数据进行编码
train_y = train_df.label
val_y = val_df.label
test_y = test_df.label
print("Label:")
print(train_y[:10])

le = LabelEncoder()
train_y = le.fit_transform(train_y).reshape(-1,1)
val_y = le.transform(val_y).reshape(-1,1)
test_y = le.transform(test_y).reshape(-1,1)
print("LabelEncoder")
print(train_y[:10])
print(len(train_y))

## 对数据集的标签数据进行one-hot编码
ohe = OneHotEncoder()
train_y = ohe.fit_transform(train_y).toarray()
val_y = ohe.transform(val_y).toarray()
test_y = ohe.transform(test_y).toarray()
print("OneHotEncoder:")
print(train_y[:10])

#----------------------第三步 使用Tokenizer对词组进行编码--------------------
max_words = 6000
max_len = 600
tok = Tokenizer(num_words=max_words)  #最大词语数为6000
print(train_df.cutword[:5])
print(type(train_df.cutword))

## 防止语料中存在数字str处理
train_content = [str(a) for a in train_df.cutword.tolist()]
val_content = [str(a) for a in val_df.cutword.tolist()]
test_content = [str(a) for a in test_df.cutword.tolist()]
tok.fit_on_texts(train_content)
print(tok)

## 保存训练好的Tokenizer和导入
with open('tok.pickle', 'wb') as handle: #saving
    pickle.dump(tok, handle, protocol=pickle.HIGHEST_PROTOCOL)
with open('tok.pickle', 'rb') as handle: #loading
    tok = pickle.load(handle)

#--------------------------第四步 数据转化为序列-----------------------------
train_seq = tok.texts_to_sequences(train_content)
val_seq = tok.texts_to_sequences(val_content)
test_seq = tok.texts_to_sequences(test_content)

## 将每个序列调整为相同的长度
train_seq_mat = sequence.pad_sequences(train_seq,maxlen=max_len)
val_seq_mat = sequence.pad_sequences(val_seq,maxlen=max_len)
test_seq_mat = sequence.pad_sequences(test_seq,maxlen=max_len)
print("数据转换序列")
print(train_seq_mat.shape)
print(val_seq_mat.shape)
print(test_seq_mat.shape)
print(train_seq_mat[:2])

#------------------------------第五步 建立TextCNN模型-------------------------
## 类别为4个
num_labels = 4
```

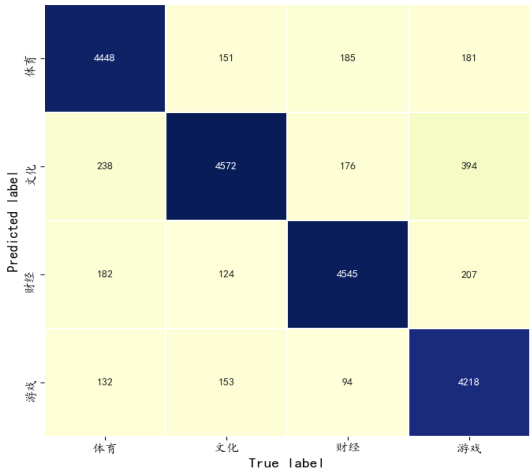
```
inputs = Input(name='inputs',shape=[max_len], dtype='float64')
## 词嵌入使用预训练的词向量
layer = Embedding(max_words+1, 256, input_length=max_len, trainable=False)(inputs)
## 词窗大小分别为3,4,5
cnn1 = Convolution1D(256, 3, padding='same', strides = 1, activation='relu')(layer)
cnn1 = MaxPool1D(pool_size=4)(cnn1)
cnn2 = Convolution1D(256, 4, padding='same', strides = 1, activation='relu')(layer)
cnn2 = MaxPool1D(pool_size=4)(cnn2)
cnn3 = Convolution1D(256, 5, padding='same', strides = 1, activation='relu')(layer)
cnn3 = MaxPool1D(pool_size=4)(cnn3)

# 合并三个模型的输出向量
cnn = concatenate([cnn1,cnn2,cnn3], axis=-1)
flat = Flatten()(cnn)
drop = Dropout(0.2)(flat)
main_output = Dense(num_labels, activation='softmax')(drop)
model = Model(inputs=inputs, outputs=main_output)
model.summary()
model.compile(loss="categorical_crossentropy",
              optimizer='adam',      # RMSprop()
              metrics=["accuracy"])

#------------------------------第六步 模型训练和预测-------------------------
## 先设置为train训练 再设置为test测试
flag = "train"
if flag == "train":
    print("模型训练")
    ## 模型训练 当val-loss不再提升时停止训练 0.0001
    model_fit = model.fit(train_seq_mat, train_y, batch_size=128, epochs=10,
                          validation_data=(val_seq_mat,val_y),
                          callbacks=[EarlyStopping(monitor='val_loss',min_delta=0.0001)]
                         )
    model.save('my_model.h5')
    del model
    elapsed = (time.clock() - start)
    print("Time used:", elapsed)
    print(model_fit.history)

else:
    print("模型预测")
    ## 导入已经训练好的模型
    model = load_model('my_model.h5')
    ## 对测试集进行预测
    test_pre = model.predict(test_seq_mat)
    ## 评价预测效果，计算混淆矩阵
    confm = metrics.confusion_matrix(np.argmax(test_y,axis=1),np.argmax(test_pre,axis=1))
    print(confm)

    ## 混淆矩阵可视化
    Labname = ["体育", "文化", "财经", "游戏"]
    print(metrics.classification_report(np.argmax(test_y,axis=1),np.argmax(test_pre,axis=1)))
    plt.figure(figsize=(8,8))
    sns.heatmap(confm.T, square=True, annot=True,
                fmt='d', cbar=False, linewidths=.6,
                cmap="YlGnBu")
    plt.xlabel('True label',size = 14)
    plt.ylabel('Predicted label', size = 14)
    plt.xticks(np.arange(4)+0.5, Labname, size = 12)
    plt.yticks(np.arange(4)+0.5, Labname, size = 12)
    plt.savefig('result.png')
    plt.show()
```

```
#---------------------------------第七 验证算法-------------------------
## 使用tok对验证数据集重新预处理，并使用训练好的模型进行预测
val_seq = tok.texts_to_sequences(val_df.cutword)
## 将每个序列调整为相同的长度
val_seq_mat = sequence.pad_sequences(val_seq,maxlen=max_len)
## 对验证集进行预测
val_pre = model.predict(val_seq_mat)
print(metrics.classification_report(np.argmax(val_y,axis=1),np.argmax(val_pre,axis=1)))
elapsed = (time.clock() - start)
print("Time used:", elapsed)
```

训练模型如下所示：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inputs (InputLayer)          (None, 600)               0
_________________________________________________________________
embedding_1 (Embedding)      (None, 600, 256)          1536256
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 600, 256)          196864
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 600, 256)          262400
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 600, 256)          327936
_________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 150, 256)       0
_________________________________________________________________
max_pooling1d_2 (MaxPooling1D)  (None, 150, 256)       0
_________________________________________________________________
max_pooling1d_3 (MaxPooling1D)  (None, 150, 256)       0
_________________________________________________________________
concatenate_1 (Concatenate)  (None, 150, 768)          0
                                                       max_pooling1d_1[0][0]
                                                       max_pooling1d_2[0][0]
                                                       max_pooling1d_3[0][0]
_________________________________________________________________
flatten_1 (Flatten)          (None, 115200)            0
_________________________________________________________________
dropout_1 (Dropout)          (None, 115200)            0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 460804
=================================================================
Total params: 2,784,260
Trainable params: 1,248,004
Non-trainable params: 1,536,256
_________________________________________________________________
```

预测结果如下：



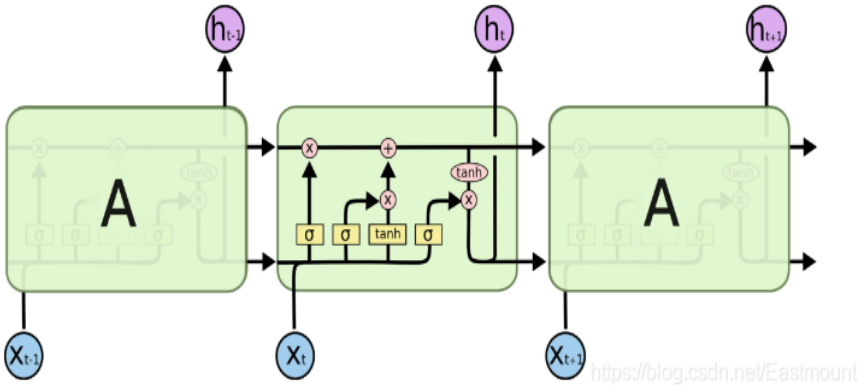

```
[[4448  238  182  132]
 [ 151 4572  124  153]
 [ 185  176 4545   94]
 [ 181  394  207 4218]]
             precision    recall  f1-score   support

          0       0.90      0.89      0.89      5000
          1       0.85      0.91      0.88      5000
          2       0.90      0.91      0.90      5000
          3       0.92      0.84      0.88      5000

avg / total       0.89      0.89      0.89     20000

             precision    recall  f1-score   support

          0       0.90      0.88      0.89      5000
          1       0.86      0.93      0.89      5000
          2       0.91      0.89      0.90      5000
          3       0.92      0.88      0.90      5000

avg / total       0.90      0.90      0.90     20000
```

五.LSTM中文文本分类

1.原理介绍

Long Short Term 网络（LSTM） 是一种RNN（Recurrent Neural Network）特殊的类型，可以学习长期依赖信息。LSTM 由Hochreiter & Schmidhuber (1997)提出，并在近期被Alex Graves进行了改良和推广。在很多问题，LSTM都取得相当巨大的成功，并得到了广泛的使用。

由于RNN存在梯度消失的问题，人们对于序列索引位置t的隐藏结构做了改进，通过一些技巧让隐藏结构复杂起来，来避免梯度消失的问题，这样的特殊RNN就是我们的LSTM。LSTM的全称是Long Short-Term Memory。LSTM由于其设计的特点，非常适合用于对时序数据的建模，如文本数据。LSTM的结构如下图：



LSTM 通过刻意的设计来避免长期依赖问题。记住长期的信息在实践中是 LSTM 的默认行为，而非需要付出很大代价才能获得的能力。LSTM是在普通的RNN上面做了一些改进，LSTM RNN多了三个控制器，即：

- 输入控制器
- 输出控制器
- 忘记控制器

左边多了个条主线，例如电影的主线剧情，而原本的RNN体系变成了分线剧情，并且三个控制器都在分线上。

- **输入控制器（write gate）**：在输入input时设置一个gate，gate的作用是判断要不要写入这个input到我们的内存Memory中，它相当于一个参数，也是可以被训练的，这个参数就是用来控制要不要记住当下这个点。
- **输出控制器（read gate）**：在输出位置的gate，判断要不要读取现在的Memory。
- **忘记控制器（forget gate）**：处理位置的忘记控制器，判断要不要忘记之前的Memory。

LSTM工作原理为：如果分支内容对于最终结果十分重要，输入控制器会将这个分支内容按重要程度写入主线内容，再进行分析；如果分线内容改变了我们之前的想法，那么忘记控制器会将某些主线内容忘记，然后按比例替换新内容，所以主线内容的更新就取决于输入和忘记控制；最后的输出会基于主线内容和分线内容。通过这三个gate能够很好地控制我们的RNN，基于这些控制机制，LSTM是延缓记忆的良药，从而带来更好的结果。

2.代码实现

Keras实现文本分类的LSTM代码如下：

- **Keras_LSTM_cnews.py**

```
"""
Created on 2021-03-19
@author: xiuzhang Eastmount CSDN
LSTM Model
"""
import os
import time
import pickle
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding
from keras.layers import Convolution1D, MaxPool1D, Flatten
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.callbacks import EarlyStopping
from keras.models import load_model
from keras.models import Sequential

#GPU加速 CuDNNLSTM比LSTM快
from keras.layers import CuDNNLSTM, CuDNNGRU

## GPU处理 读者如果是CPU注释该部分代码即可
## 指定每个GPU进程中使用显存的上限 0.9表示可以使用GPU 90%的资源进行训练
os.environ["CUDA_DEVICES_ORDER"] = "PCI_BUS_IS"
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.8)
sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))

start = time.clock()

#----------------------------第一步 数据读取----------------------------
## 读取测数据集
train_df = pd.read_csv("news_dataset_train_fc.csv")
val_df = pd.read_csv("news_dataset_val_fc.csv")
test_df = pd.read_csv("news_dataset_test_fc.csv")
print(train_df.head())

## 解决中文显示问题
plt.rcParams['font.sans-serif'] = ['KaiTi']  #指定默认字体 SimHei黑体
plt.rcParams['axes.unicode_minus'] = False   #解决保存图像是负号'

#--------------------------第二步 OneHotEncoder()编码--------------------
## 对数据集的标签数据进行编码
train_y = train_df.label
val_y = val_df.label
test_y = test_df.label
print("Label:")
print(train_y[:10])

le = LabelEncoder()
train_y = le.fit_transform(train_y).reshape(-1,1)
val_y = le.transform(val_y).reshape(-1,1)
test_y = le.transform(test_y).reshape(-1,1)
print("LabelEncoder")
print(train_y[:10])
print(len(train_y))
```

```
## 对数据集的标签数据进行one-hot编码
ohe = OneHotEncoder()
train_y = ohe.fit_transform(train_y).toarray()
val_y = ohe.transform(val_y).toarray()
test_y = ohe.transform(test_y).toarray()
print("OneHotEncoder:")
print(train_y[:10])

#----------------------第三步 使用Tokenizer对词组进行编码--------------------
max_words = 6000
max_len = 600
tok = Tokenizer(num_words=max_words)  #最大词语数为6000
print(train_df.cutword[:5])
print(type(train_df.cutword))

## 防止语料中存在数字str处理
train_content = [str(a) for a in train_df.cutword.tolist()]
val_content = [str(a) for a in val_df.cutword.tolist()]
test_content = [str(a) for a in test_df.cutword.tolist()]
tok.fit_on_texts(train_content)
print(tok)

## 保存训练好的Tokenizer和导入
with open('tok.pickle', 'wb') as handle: #saving
    pickle.dump(tok, handle, protocol=pickle.HIGHEST_PROTOCOL)
with open('tok.pickle', 'rb') as handle: #loading
    tok = pickle.load(handle)

#--------------------------第四步 数据转化为序列-----------------------------
train_seq = tok.texts_to_sequences(train_content)
val_seq = tok.texts_to_sequences(val_content)
test_seq = tok.texts_to_sequences(test_content)

## 将每个序列调整为相同的长度
train_seq_mat = sequence.pad_sequences(train_seq,maxlen=max_len)
val_seq_mat = sequence.pad_sequences(val_seq,maxlen=max_len)
test_seq_mat = sequence.pad_sequences(test_seq,maxlen=max_len)
print("数据转换序列")
print(train_seq_mat.shape)
print(val_seq_mat.shape)
print(test_seq_mat.shape)
print(train_seq_mat[:2])

#-------------------------------第五步 建立LSTM模型-------------------------
## 定义LSTM模型
inputs = Input(name='inputs',shape=[max_len],dtype='float64')
## Embedding(词汇表大小,batch大小,每个新闻的词长)
layer = Embedding(max_words+1, 128, input_length=max_len)(inputs)
#layer = LSTM(128)(layer)
layer = CuDNNLSTM(128)(layer)

layer = Dense(128, activation="relu", name="FC1")(layer)
layer = Dropout(0.1)(layer)
layer = Dense(4, activation="softmax", name="FC2")(layer)
model = Model(inputs=inputs, outputs=layer)
model.summary()
model.compile(loss="categorical_crossentropy",
              optimizer='adam',  # RMSprop()
              metrics=["accuracy"])

#-------------------------------第六步 模型训练和预测-------------------------
```

```
## 先设置为train训练 再设置为test测试
flag = "train"
if flag == "train":
    print("模型训练")
    ## 模型训练 当val-loss不再提升时停止训练 0.0001
    model_fit = model.fit(train_seq_mat, train_y, batch_size=128, epochs=10,
                          validation_data=(val_seq_mat,val_y),
                          callbacks=[EarlyStopping(monitor='val_loss',min_delta=0.0001)]
                         )
    model.save('my_model.h5')
    del model
    elapsed = (time.clock() - start)
    print("Time used:", elapsed)
    print(model_fit.history)

else:
    print("模型预测")
    ## 导入已经训练好的模型
    model = load_model('my_model.h5')
    ## 对测试集进行预测
    test_pre = model.predict(test_seq_mat)
    ## 评价预测效果，计算混淆矩阵
    confm = metrics.confusion_matrix(np.argmax(test_y,axis=1),np.argmax(test_pre,axis=1))
    print(confm)

    ## 混淆矩阵可视化
    Labname = ["体育", "文化", "财经", "游戏"]
    print(metrics.classification_report(np.argmax(test_y,axis=1),np.argmax(test_pre,axis=1)))
    plt.figure(figsize=(8,8))
    sns.heatmap(confm.T, square=True, annot=True,
                fmt='d', cbar=False, linewidths=.6,
                cmap="YlGnBu")
    plt.xlabel('True label',size = 14)
    plt.ylabel('Predicted label', size = 14)
    plt.xticks(np.arange(4)+0.8, Labname, size = 12)
    plt.yticks(np.arange(4)+0.4, Labname, size = 12)
    plt.savefig('result.png')
    plt.show()

    #---------------------------------第七 验证算法-------------------------
    ## 使用tok对验证数据集重新预处理，并使用训练好的模型进行预测
    val_seq = tok.texts_to_sequences(val_df.cutword)
    ## 将每个序列调整为相同的长度
    val_seq_mat = sequence.pad_sequences(val_seq,maxlen=max_len)
    ## 对验证集进行预测
    val_pre = model.predict(val_seq_mat)
    print(metrics.classification_report(np.argmax(val_y,axis=1),np.argmax(val_pre,axis=1)))
    elapsed = (time.clock() - start)
    print("Time used:", elapsed)
```

训练输出模型如下所示：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inputs (InputLayer)          (None, 600)               0
_________________________________________________________________
embedding_1 (Embedding)      (None, 600, 128)          768128
_________________________________________________________________
cu_dnnlstm_1 (CuDNNLSTM)     (None, 128)               132096
_________________________________________________________________
```

```
FC1 (Dense)                  (None, 128)               16512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
FC2 (Dense)                  (None, 4)                 516
=================================================================
Total params: 917,252
Trainable params: 917,252
Non-trainable params: 0
```

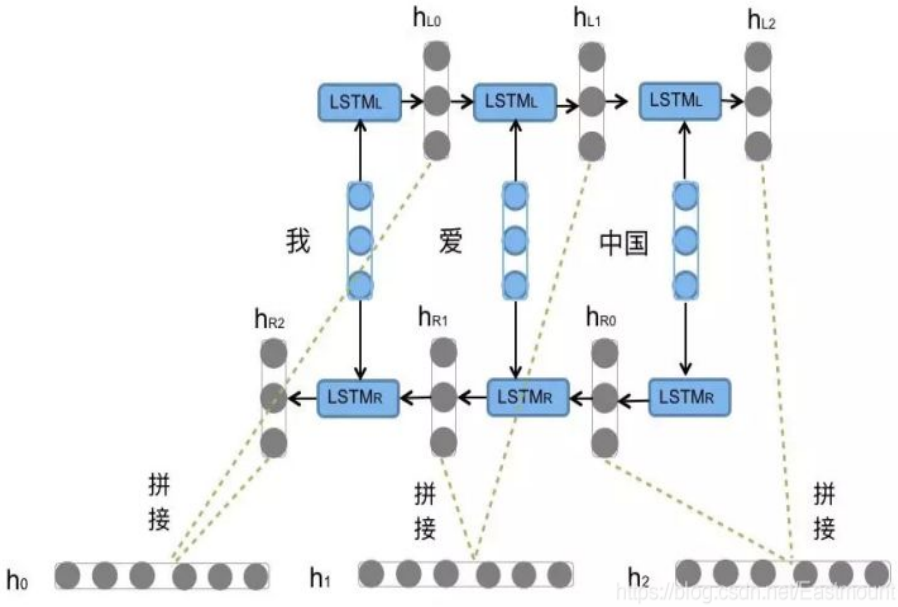
预测结果如下所示：

```
[[4539  153  188  120]
 [  47 4628  181  144]
 [ 113  133 4697   57]
 [ 101  292  157 4450]]
              precision    recall  f1-score   support

           0       0.95      0.91      0.93      5000
           1       0.89      0.93      0.91      5000
           2       0.90      0.94      0.92      5000
           3       0.93      0.89      0.91      5000

avg / total       0.92      0.92      0.92     20000

              precision    recall  f1-score   support

           0       0.96      0.89      0.92      5000
           1       0.89      0.94      0.92      5000
           2       0.90      0.93      0.92      5000
           3       0.94      0.92      0.93      5000

avg / total       0.92      0.92      0.92     20000
```

六.BiLSTM中文文本分类

1.原理介绍

BiLSTM是Bi-directional Long Short-Term Memory的缩写，是由前向LSTM与后向LSTM组合而成。它和LSTM在自然语言处理任务中都常被用来建模上下文信息。前向的LSTM与后向的LSTM结合成BiLSTM。比如，我们对“我爱中国”这句话进行编码，模型如图所示。



由于利用LSTM对句子进行建模还存在一个问题：无法编码从后到前的信息。在更细粒度的分类时，如对于强程度的褒义、弱程度的褒义、中性、弱程度的贬义、强程度的贬义的五分类任务需要注意情感词、程度词、否定词之间的交互。举一个例子，“这个餐厅脏得不行，没有隔壁好”，这里的“不行”是对“脏”的程度的一种修饰，通过BiLSTM可以更好的捕捉双向的语义依赖。

- 参考文章：https://zhuanlan.zhihu.com/p/47802053

2.代码实现

Keras实现文本分类的BiLSTM代码如下：

- **Keras_BiLSTM_cnews.py**

```
"""
Created on 2021-03-19
@author: xiuzhang Eastmount CSDN
BiLSTM Model
"""
import os
import time
import pickle
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding
from keras.layers import Convolution1D, MaxPool1D, Flatten
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.callbacks import EarlyStopping
from keras.models import load_model
from keras.models import Sequential

#GPU加速 CuDNNLSTM比LSTM快
```

```
from keras.layers import CuDNNLSTM, CuDNNGRU
from keras.layers import Bidirectional

## GPU处理 读者如果是CPU注释该部分代码即可
## 指定每个GPU进程中使用显存的上限 0.9表示可以使用GPU 90%的资源进行训练
os.environ["CUDA_DEVICES_ORDER"] = "PCI_BUS_IS"
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.8)
sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))

start = time.clock()

#----------------------------第一步 数据读取----------------------------
## 读取测数据集
train_df = pd.read_csv("news_dataset_train_fc.csv")
val_df = pd.read_csv("news_dataset_val_fc.csv")
test_df = pd.read_csv("news_dataset_test_fc.csv")
print(train_df.head())

## 解决中文显示问题
plt.rcParams['font.sans-serif'] = ['KaiTi']  #指定默认字体 SimHei黑体
plt.rcParams['axes.unicode_minus'] = False   #解决保存图像是负号'

#--------------------------第二步 OneHotEncoder()编码--------------------
## 对数据集的标签数据进行编码
train_y = train_df.label
val_y = val_df.label
test_y = test_df.label
print("Label:")
print(train_y[:10])

le = LabelEncoder()
train_y = le.fit_transform(train_y).reshape(-1,1)
val_y = le.transform(val_y).reshape(-1,1)
test_y = le.transform(test_y).reshape(-1,1)
print("LabelEncoder")
print(train_y[:10])
print(len(train_y))

## 对数据集的标签数据进行one-hot编码
ohe = OneHotEncoder()
train_y = ohe.fit_transform(train_y).toarray()
val_y = ohe.transform(val_y).toarray()
test_y = ohe.transform(test_y).toarray()
print("OneHotEncoder:")
print(train_y[:10])

#-----------------------第三步 使用Tokenizer对词组进行编码--------------------
max_words = 6000
max_len = 600
tok = Tokenizer(num_words=max_words)  #最大词语数为6000
print(train_df.cutword[:5])
print(type(train_df.cutword))

## 防止语料中存在数字str处理
train_content = [str(a) for a in train_df.cutword.tolist()]
val_content = [str(a) for a in val_df.cutword.tolist()]
test_content = [str(a) for a in test_df.cutword.tolist()]
tok.fit_on_texts(train_content)
print(tok)

## 保存训练好的Tokenizer和导入
```

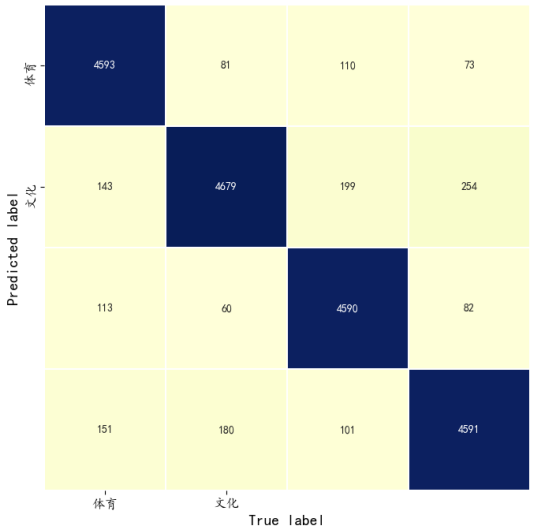
```
with open('tok.pickle', 'wb') as handle: #saving
    pickle.dump(tok, handle, protocol=pickle.HIGHEST_PROTOCOL)
with open('tok.pickle', 'rb') as handle: #loading
    tok = pickle.load(handle)

#--------------------------第四步 数据转化为序列-----------------------------
train_seq = tok.texts_to_sequences(train_content)
val_seq = tok.texts_to_sequences(val_content)
test_seq = tok.texts_to_sequences(test_content)

## 将每个序列调整为相同的长度
train_seq_mat = sequence.pad_sequences(train_seq,maxlen=max_len)
val_seq_mat = sequence.pad_sequences(val_seq,maxlen=max_len)
test_seq_mat = sequence.pad_sequences(test_seq,maxlen=max_len)
print("数据转换序列")
print(train_seq_mat.shape)
print(val_seq_mat.shape)
print(test_seq_mat.shape)
print(train_seq_mat[:2])

#-------------------------------第五步 建立BiLSTM模型-------------------------
num_labels = 4
model = Sequential()
model.add(Embedding(max_words+1, 128, input_length=max_len))
model.add(Bidirectional(CuDNNLSTM(128)))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(num_labels, activation='softmax'))
model.summary()
model.compile(loss="categorical_crossentropy",
              optimizer='adam',  # RMSprop()
              metrics=["accuracy"])

#-------------------------------第六步 模型训练和预测-------------------------
## 先设置为train训练 再设置为test测试
flag = "train"
if flag == "train":
    print("模型训练")
    ## 模型训练 当val-loss不再提升时停止训练 0.0001
    model_fit = model.fit(train_seq_mat, train_y, batch_size=128, epochs=10,
                          validation_data=(val_seq_mat,val_y),
                          callbacks=[EarlyStopping(monitor='val_loss',min_delta=0.0001)]
                         )
    model.save('my_model.h5')
    del model
    elapsed = (time.clock() - start)
    print("Time used:", elapsed)
    print(model_fit.history)

else:
    print("模型预测")
    ## 导入已经训练好的模型
    model = load_model('my_model.h5')
    ## 对测试集进行预测
    test_pre = model.predict(test_seq_mat)
    ## 评价预测效果，计算混淆矩阵
    confm = metrics.confusion_matrix(np.argmax(test_y,axis=1),np.argmax(test_pre,axis=1))
    print(confm)

    ## 混淆矩阵可视化
    Labname = ["体育", "文化", "财经", "游戏"]
    print(metrics.classification_report(np.argmax(test_y,axis=1),np.argmax(test_pre,axis=1)))
```

```
    plt.figure(figsize=(8,8))
    sns.heatmap(confm.T, square=True, annot=True,
                fmt='d', cbar=False, linewidths=.6,
                cmap="YlGnBu")
    plt.xlabel('True label',size = 14)
    plt.ylabel('Predicted label', size = 14)
    plt.xticks(np.arange(4)+0.5, Labname, size = 12)
    plt.yticks(np.arange(4)+0.5, Labname, size = 12)
    plt.savefig('result.png')
    plt.show()

    #---------------------------------第七 验证算法-------------------------
    ## 使用tok对验证数据集重新预处理，并使用训练好的模型进行预测
    val_seq = tok.texts_to_sequences(val_df.cutword)
    ## 将每个序列调整为相同的长度
    val_seq_mat = sequence.pad_sequences(val_seq,maxlen=max_len)
    ## 对验证集进行预测
    val_pre = model.predict(val_seq_mat)
    print(metrics.classification_report(np.argmax(val_y,axis=1),np.argmax(val_pre,axis=1)))
    elapsed = (time.clock() - start)
    print("Time used:", elapsed)
```

训练输出模型如下所示，GPU时间还是非常快。

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 600, 128)          768128
_________________________________________________________________
bidirectional_1 (Bidirection (None, 256)               264192
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 516
=================================================================
Total params: 1,065,732
Trainable params: 1,065,732
Non-trainable params: 0

Train on 40000 samples, validate on 20000 samples
Epoch 1/10
40000/40000 [==============================] - 23s 587us/step - loss: 0.5825 - acc: 0.8038 - val_loss: 0.
2321 - val_acc: 0.9246
Epoch 2/10
40000/40000 [==============================] - 21s 521us/step - loss: 0.1433 - acc: 0.9542 - val_loss: 0.
2422 - val_acc: 0.9228
Time used: 52.763230400000005
```

预测结果如下图所示：




```
[[4593  143  113  151]
 [  81 4679   60  180]
 [ 110  199 4590  101]
 [  73  254   82 4591]]
             precision    recall  f1-score   support

          0       0.95      0.92      0.93      5000
          1       0.89      0.94      0.91      5000
          2       0.95      0.92      0.93      5000
          3       0.91      0.92      0.92      5000

avg / total       0.92      0.92      0.92     20000

             precision    recall  f1-score   support

          0       0.94      0.90      0.92      5000
          1       0.89      0.95      0.92      5000
          2       0.95      0.90      0.93      5000
          3       0.91      0.94      0.93      5000

avg / total       0.92      0.92      0.92     20000
```

七.BiLSTM+Attention中文文本分类

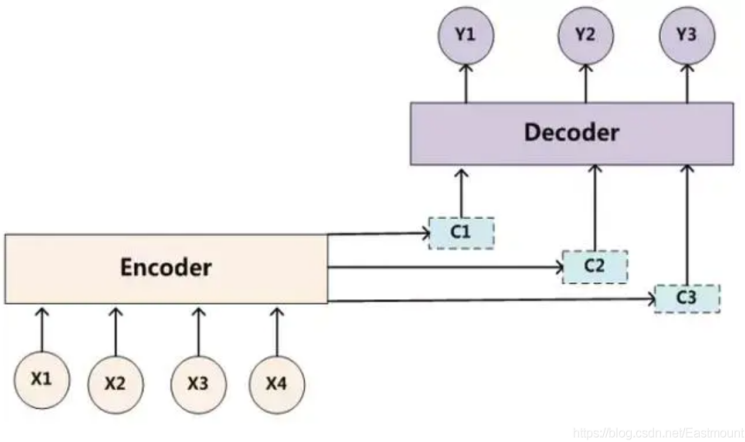
1.原理介绍

Attention机制是模仿人类注意力而提出的一种解决问题的办法，简单地说就是从大量信息中快速筛选出高价值信息。主要用于解决LSTM/RNN模型输入序列较长的时候很难获得最终合理的向量表示问题，做法是保留LSTM的中间结果，用新的模型对其进行学习，并将其与输出进行关联，从而达到信息筛选的目的。

What is attention?

先简单描述一下attention机制是什么。相信做NLP的同学对这个机制不会很陌生，它在论文 《**Attention is all you need**》 中可以说是大放异彩，在machine translation任务中，帮助深度模型在性能上有了很大的提升，输出了当时最好的state-of-art model。当然该模型除了attention机制外，还用了很多有用的trick，以帮助提升模型性能。但是不能否认的时，这个模型的核心就是attention。

attention机制又称为注意力机制，顾名思义，是一种能让模型对重要信息重点关注并充分学习吸收的技术，它不算是一个完整的模型，应当是一种技术，能够作用于任何序列模型中。



Why attention?

为什么要引入attention机制。比如在seq2seq模型中，对于一段文本序列，我们通常要使用某种机制对该序列进行编码，通过降维等方式将其encode成一个固定长度的向量，用于输入到后面的全连接层。一般我们会使用CNN或者RNN（包括GRU或者LSTM）等模型来对序列数据进行编码，然后采用各种pooling或者对RNN直接取最后一个t时刻的hidden state作为句子的向量输出。

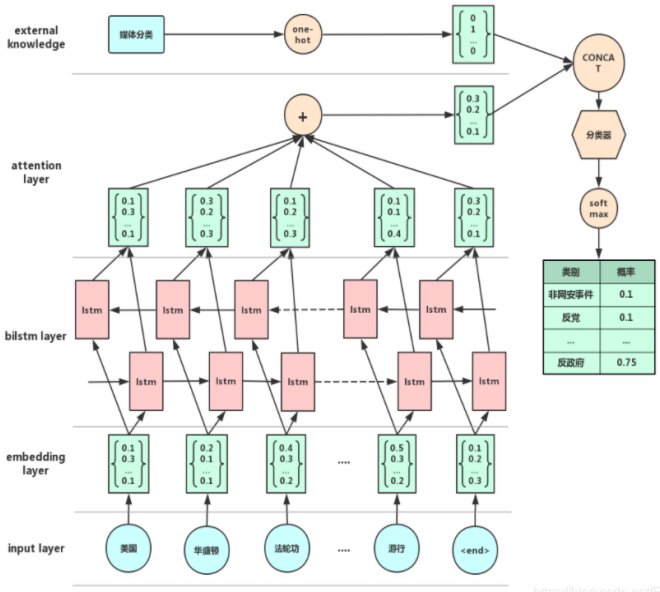
但这里会有一个问题： 常规的编码方法，无法体现对一个句子序列中不同语素的关注程度，在自然语言中，一个句子中的不同部分是有不同含义和重要性的，比如上面的例子中：I hate this movie.如果做情感分析，明显对hate这个词语应当关注更多。当然是用CNN和RNN能够编码这种信息，但这种编码能力也是有上限的，对于较长的文本，模型效果不会再提升太多。

- 参考及推荐文章：https://zhuanlan.zhihu.com/p/46313756

Attention的应用领域非常广泛，文本、图片等都有应用。

- 文本：应用于seq2seq模型，最常见的应用是翻译
- 图片：应用于卷积神经网络的图片提取
- 语音

下图是一个比较经典的BiLSTM+Attention模型，也是我们接下来需要建立的模型。



2.代码实现

Keras实现文本分类的BiLSTM+Attention代码如下：

- **Keras_Attention_BiLSTM_cnews.py**

```
"""
Created on 2021-03-19
@author: xiuzhang Eastmount CSDN
BiLSTM+Attention Model
"""
import os
import time
import pickle
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding
from keras.layers import Convolution1D, MaxPool1D, Flatten
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.callbacks import EarlyStopping
from keras.models import load_model
from keras.models import Sequential

#GPU加速 CuDNNLSTM比LSTM快
from keras.layers import CuDNNLSTM, CuDNNGRU
from keras.layers import Bidirectional

## GPU处理 读者如果是CPU注释该部分代码即可
## 指定每个GPU进程中使用显存的上限 0.9表示可以使用GPU 90%的资源进行训练
os.environ["CUDA_DEVICES_ORDER"] = "PCI_BUS_IS"
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
```

```
gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.8)
sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))

start = time.clock()

#----------------------------第一步 数据读取----------------------------
## 读取测数据集
train_df = pd.read_csv("news_dataset_train_fc.csv")
val_df = pd.read_csv("news_dataset_val_fc.csv")
test_df = pd.read_csv("news_dataset_test_fc.csv")
print(train_df.head())

## 解决中文显示问题
plt.rcParams['font.sans-serif'] = ['KaiTi']  #指定默认字体 SimHei黑体
plt.rcParams['axes.unicode_minus'] = False   #解决保存图像是负号'

#--------------------------第二步 OneHotEncoder()编码--------------------
## 对数据集的标签数据进行编码
train_y = train_df.label
val_y = val_df.label
test_y = test_df.label
print("Label:")
print(train_y[:10])

le = LabelEncoder()
train_y = le.fit_transform(train_y).reshape(-1,1)
val_y = le.transform(val_y).reshape(-1,1)
test_y = le.transform(test_y).reshape(-1,1)
print("LabelEncoder")
print(train_y[:10])
print(len(train_y))

## 对数据集的标签数据进行one-hot编码
ohe = OneHotEncoder()
train_y = ohe.fit_transform(train_y).toarray()
val_y = ohe.transform(val_y).toarray()
test_y = ohe.transform(test_y).toarray()
print("OneHotEncoder:")
print(train_y[:10])

#-----------------------第三步 使用Tokenizer对词组进行编码--------------------
max_words = 6000
max_len = 600
tok = Tokenizer(num_words=max_words)  #最大词语数为6000
print(train_df.cutword[:5])
print(type(train_df.cutword))

## 防止语料中存在数字str处理
train_content = [str(a) for a in train_df.cutword.tolist()]
val_content = [str(a) for a in val_df.cutword.tolist()]
test_content = [str(a) for a in test_df.cutword.tolist()]
tok.fit_on_texts(train_content)
print(tok)

## 保存训练好的Tokenizer和导入
with open('tok.pickle', 'wb') as handle: #saving
    pickle.dump(tok, handle, protocol=pickle.HIGHEST_PROTOCOL)
with open('tok.pickle', 'rb') as handle: #loading
    tok = pickle.load(handle)

#----------------------------第四步 数据转化为序列-----------------------------
train_seq = tok.texts_to_sequences(train_content)
```

```
val_seq = tok.texts_to_sequences(val_content)
test_seq = tok.texts_to_sequences(test_content)

## 将每个序列调整为相同的长度
train_seq_mat = sequence.pad_sequences(train_seq,maxlen=max_len)
val_seq_mat = sequence.pad_sequences(val_seq,maxlen=max_len)
test_seq_mat = sequence.pad_sequences(test_seq,maxlen=max_len)
print("数据转换序列")
print(train_seq_mat.shape)
print(val_seq_mat.shape)
print(test_seq_mat.shape)
print(train_seq_mat[:2])

#---------------------------第五步 建立Attention机制----------------------
"""
由于Keras目前还没有现成的Attention层可以直接使用，我们需要自己来构建一个新的层函数。
Keras自定义的函数主要分为四个部分，分别是：
init：初始化一些需要的参数
bulid：具体来定义权重是怎么样的
call：核心部分，定义向量是如何进行运算的
compute_output_shape：定义该层输出的大小
推荐文章：
    https://blog.csdn.net/huanghaocs/article/details/95752379
    https://zhuanlan.zhihu.com/p/29201491
"""

# Hierarchical Model with Attention
from keras import initializers
from keras import constraints
from keras import activations
from keras import regularizers
from keras import backend as K
from keras.engine.topology import Layer

K.clear_session()

class AttentionLayer(Layer):
    def __init__(self, attention_size=None, **kwargs):
        self.attention_size = attention_size
        super(AttentionLayer, self).__init__(**kwargs)

    def get_config(self):
        config = super().get_config()
        config['attention_size'] = self.attention_size
        return config

    def build(self, input_shape):
        assert len(input_shape) == 3

        self.time_steps = input_shape[1]
        hidden_size = input_shape[2]
        if self.attention_size is None:
            self.attention_size = hidden_size

        self.W = self.add_weight(name='att_weight', shape=(hidden_size, self.attention_size),
                                 initializer='uniform', trainable=True)
        self.b = self.add_weight(name='att_bias', shape=(self.attention_size,),
                                 initializer='uniform', trainable=True)
        self.V = self.add_weight(name='att_var', shape=(self.attention_size,),
                                 initializer='uniform', trainable=True)
        super(AttentionLayer, self).build(input_shape)
```

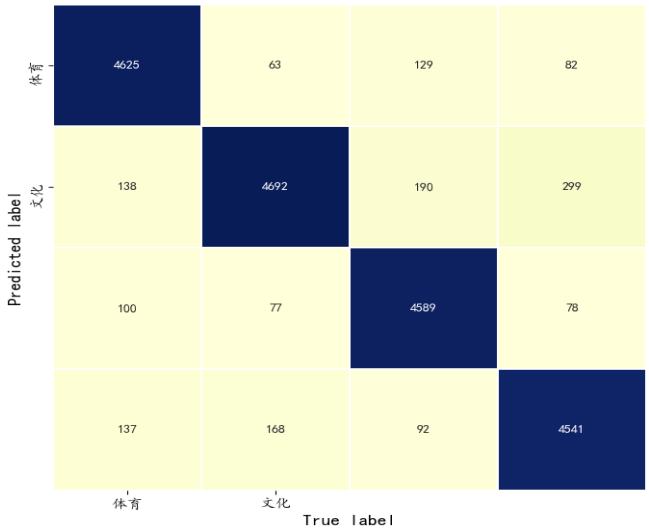
```
    def call(self, inputs):
        self.V = K.reshape(self.V, (-1, 1))
        H = K.tanh(K.dot(inputs, self.W) + self.b)
        score = K.softmax(K.dot(H, self.V), axis=1)
        outputs = K.sum(score * inputs, axis=1)
        return outputs

    def compute_output_shape(self, input_shape):
        return input_shape[0], input_shape[2]

#------------------------------第六步 建立BiLSTM模型-------------------------
## 定义BiLSTM模型
## BiLSTM+Attention
num_labels = 4
inputs = Input(name='inputs',shape=[max_len],dtype='float64')
layer = Embedding(max_words+1, 256, input_length=max_len)(inputs)
#lstm = Bidirectional(LSTM(100, dropout=0.2, recurrent_dropout=0.1, return_sequences=True))(layer)
bilstm = Bidirectional(CuDNNLSTM(128, return_sequences=True))(layer)  #参数保持维度3
layer = Dense(128, activation='relu')(bilstm)
layer = Dropout(0.2)(layer)
## 注意力机制
attention = AttentionLayer(attention_size=50)(layer)
output = Dense(num_labels, activation='softmax')(attention)
model = Model(inputs=inputs, outputs=output)
model.summary()
model.compile(loss="categorical_crossentropy",
              optimizer='adam',      # RMSprop()
              metrics=["accuracy"])

#------------------------------第七步 模型训练和预测-------------------------
## 先设置为train训练 再设置为test测试
flag = "test"
if flag == "train":
    print("模型训练")
    ## 模型训练 当val-loss不再提升时停止训练 0.0001
    model_fit = model.fit(train_seq_mat, train_y, batch_size=128, epochs=10,
                          validation_data=(val_seq_mat,val_y),
                          callbacks=[EarlyStopping(monitor='val_loss',min_delta=0.0001)]
                         )
    ## 保存模型
    model.save('my_model.h5')
    del model  # deletes the existing model
    ## 计算时间
    elapsed = (time.clock() - start)
    print("Time used:", elapsed)
    print(model_fit.history)

else:
    print("模型预测")
    ## 导入已经训练好的模型
    model = load_model('my_model.h5', custom_objects={'AttentionLayer': AttentionLayer(50)}, compile=Fals
e)
    ## 对测试集进行预测
    test_pre = model.predict(test_seq_mat)
    ## 评价预测效果，计算混淆矩阵
    confm = metrics.confusion_matrix(np.argmax(test_y,axis=1),np.argmax(test_pre,axis=1))
    print(confm)

    ## 混淆矩阵可视化
    Labname = ["体育", "文化", "财经", "游戏"]
    print(metrics.classification_report(np.argmax(test_y,axis=1),np.argmax(test_pre,axis=1)))
    plt.figure(figsize=(8,8))
```

```
    sns.heatmap(confm.T, square=True, annot=True,
                fmt='d', cbar=False, linewidths=.6,
                cmap="YlGnBu")
    plt.xlabel('True label',size = 14)
    plt.ylabel('Predicted label', size = 14)
    plt.xticks(np.arange(4)+0.5, Labname, size = 12)
    plt.yticks(np.arange(4)+0.5, Labname, size = 12)
    plt.savefig('result.png')
    plt.show()

    #---------------------------------第七 验证算法-------------------------
    ## 使用tok对验证数据集重新预处理，并使用训练好的模型进行预测
    val_seq = tok.texts_to_sequences(val_df.cutword)
    ## 将每个序列调整为相同的长度
    val_seq_mat = sequence.pad_sequences(val_seq,maxlen=max_len)
    ## 对验证集进行预测
    val_pre = model.predict(val_seq_mat)
    print(metrics.classification_report(np.argmax(val_y,axis=1),np.argmax(val_pre,axis=1)))

    ## 计算时间
    elapsed = (time.clock() - start)
    print("Time used:", elapsed)
```

训练输出模型如下所示：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inputs (InputLayer)          (None, 600)               0
_________________________________________________________________
embedding_1 (Embedding)      (None, 600, 256)          1536256
_________________________________________________________________
bidirectional_1 (Bidirection (None, 600, 256)          395264
_________________________________________________________________
dense_1 (Dense)              (None, 600, 128)          32896
_________________________________________________________________
dropout_1 (Dropout)          (None, 600, 128)          0
_________________________________________________________________
attention_layer_1 (Attention (None, 128)               6500
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 516
=================================================================
Total params: 1,971,432
Trainable params: 1,971,432
Non-trainable params: 0
```

预测结果如下图所示：




```
[[4625  138  100  137]
 [  63 4692   77  168]
 [ 129  190 4589   92]
 [  82  299   78 4541]]
             precision    recall  f1-score   support

          0       0.94      0.93      0.93      5000
          1       0.88      0.94      0.91      5000
          2       0.95      0.92      0.93      5000
          3       0.92      0.91      0.91      5000

avg / total       0.92      0.92      0.92     20000

             precision    recall  f1-score   support

          0       0.95      0.91      0.93      5000
          1       0.88      0.95      0.91      5000
          2       0.95      0.90      0.92      5000
          3       0.92      0.93      0.93      5000

avg / total       0.92      0.92      0.92     20000
```

八.总结

写到这里，这篇文章就介绍结束了，希望对您有所帮助。文章虽然很冗余，但还是能学到知识，尤其是代码部分，后续随着作者深入，会分享更简洁的案例，继续加油~

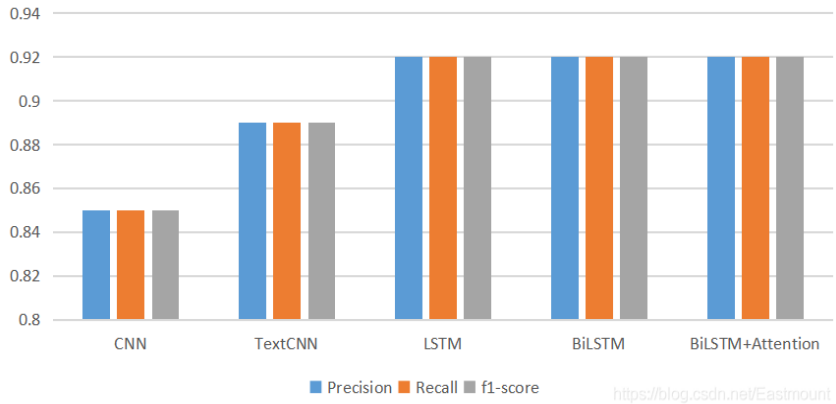
- **一.文本分类概述**
- **二.数据预处理及分词**
- **三.CNN中文文本分类**
- **四.TextCNN中文文本分类**
- **五.LSTM中文文本分类**

- **六.BiLSTM中文文本分类**
- **七.BiLSTM+Attention中文文本分类**

对比实验结果如下图所示，效果非常不理想，大家可以思考几个问题，我们后面的文章继续介绍。

- 实验结果怎么自定义函数评价，否则系统自带保留两位小时
- 实验结果怎么进行可视化分析
- 实验结果怎么进行参数选择和优化
- 实验过程的误差曲线、准确率曲线、AUC曲线怎么绘制

作者更希望提供一种可行的方法和思路，然后你针对自己的问题和数据集进行场景设计，模型优化及实验完善，最终做出自己需要的效果，一起加油吧！我也是在一步步学习总结种。



希望您喜欢这篇文章，从看视频到撰写代码，我真的写了一周时间，再次感谢参考文献的老师们。真心希望这篇文章对您有所帮助，加油~

- https://github.com/eastmountyxz/AI-for-Keras

2020年8月18新开的“娜璋AI安全之家”，主要围绕Python大数据分析、网络空间安全、人工智能、Web渗透及攻防技术进行讲解，同时分享CCF、SCI、南核北核论文的算法实现。娜璋之家会更加系统，并重构作者的所有文章，从零讲解Python和安全，写了近十年文章，真心想把自己所学所感所做分享出来，还请各位多多指教，真诚邀请您的关注！谢谢。

(By:Eastmount 2021-03-19 周五夜于武汉 http://blog.csdn.net/eastmount/)