基于Doc2vec训练句子向量

原创 荔枝boy 磐创AI 2018-05-15

收录于话题
#ML与DL基础

47个

编辑 | 磐石

出品 | 磐创AI技术团队

【磐创AI导读】：本文详细介绍了基于Doc2vec训练句子向量的原理及其python实现。欢迎大家点击上方蓝字关注我们的公众号：**磐创AI**。

目录

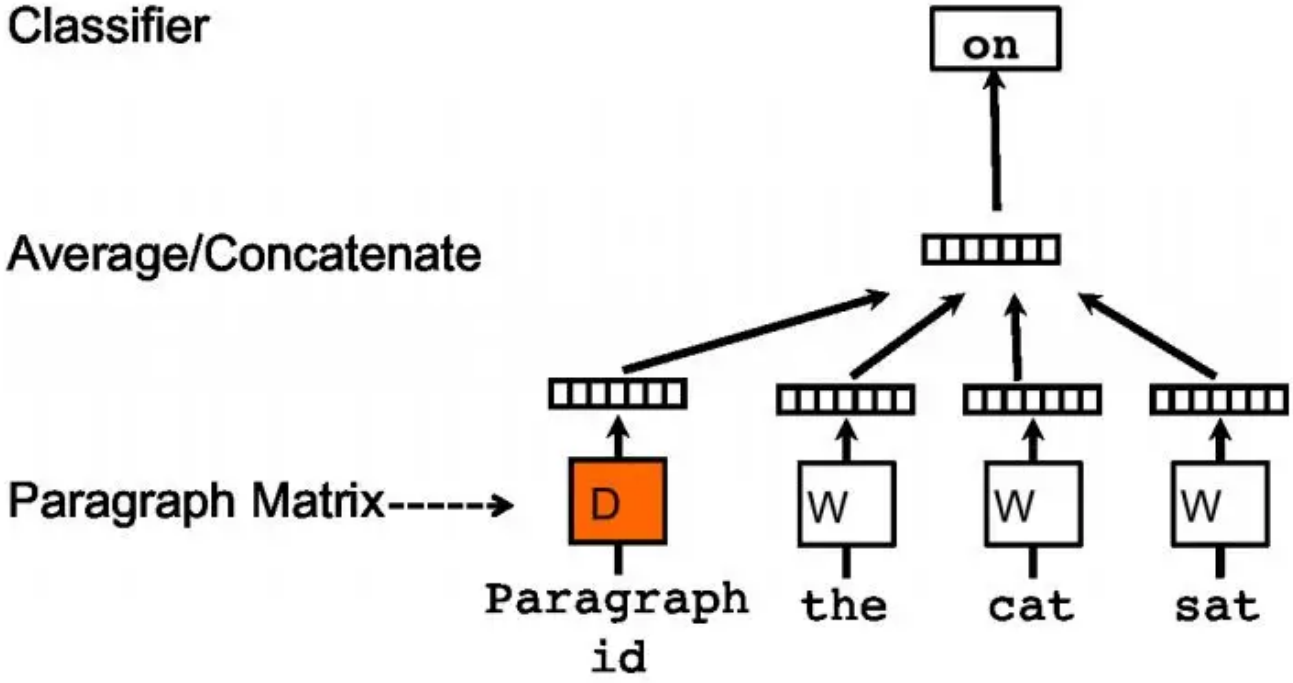
一. Doc2vec原理

前文总结了Word2vec训练词向量的细节，讲解了一个词是如何通过word2vec模型训练出唯一的向量来表示的。那接着可能就会想到，有没有什么办法能够将一个句子甚至一篇短文也用一个向量来表示呢？答案是肯定有的，构建一个句子向量有很多种方法，今天我们接着word2vec来介绍下Doc2vec，看下Doc2vec是怎么训练一个句子向量的。

许多机器学习算法需要的输入是一个固定长度的向量，当涉及到短文时，最常用的固定长度的向量方法是词袋模型（bag-of-words）。尽管它很流行，但是词袋模型存在两个主要的缺点：一个是词袋模型忽略词序，如果两个不同的句子由相同的词但是顺序不同组成，词袋模型会将这两句话定义为同一个表达；另一个是词袋模型忽略了句法，这样训练出来的模型会造成类似'powerful','strong'和'Paris'的距离是相同的，而其实'powerful'应该相对于'Paris'距离'strong'更近才对。

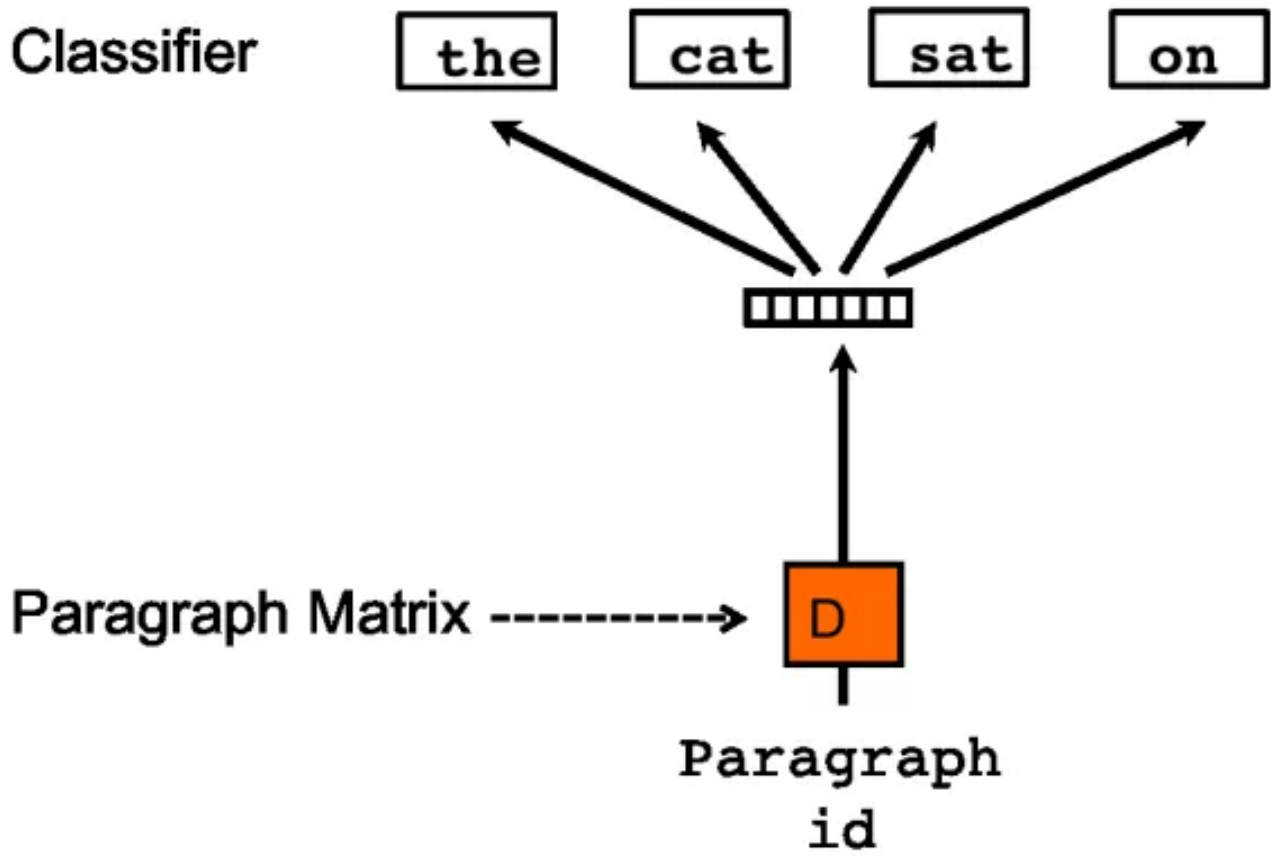
Doc2vec又叫Paragraph Vector是Tomas Mikolov基于word2vec模型提出的，其具有一些优点，比如**不用固定句子长度，接受不同长度的句子做训练样本**，Doc2vec是一个无监督学习算法，该算法用于预测一个向量来表示不同的文档，该模型的结构潜在的克服了词袋模型的缺点。

Doc2vec模型是受到了word2vec模型的启发，word2vec里预测词向量时，预测出来的词是含有词义的，比如上文提到的词向量'powerful'会相对于'Paris'离'strong'距离更近，在Doc2vec中也构建了相同的结构。所以Doc2vec克服了词袋模型中没有语义的去缺点。假设现在存在训练样本，每个句子是训练样本。和word2vec一样，Doc2vec也有两种训练方式，一种是PV-DM（Distributed Memory Model of paragraphvectors）类似于word2vec中的CBOW模型，如图一：



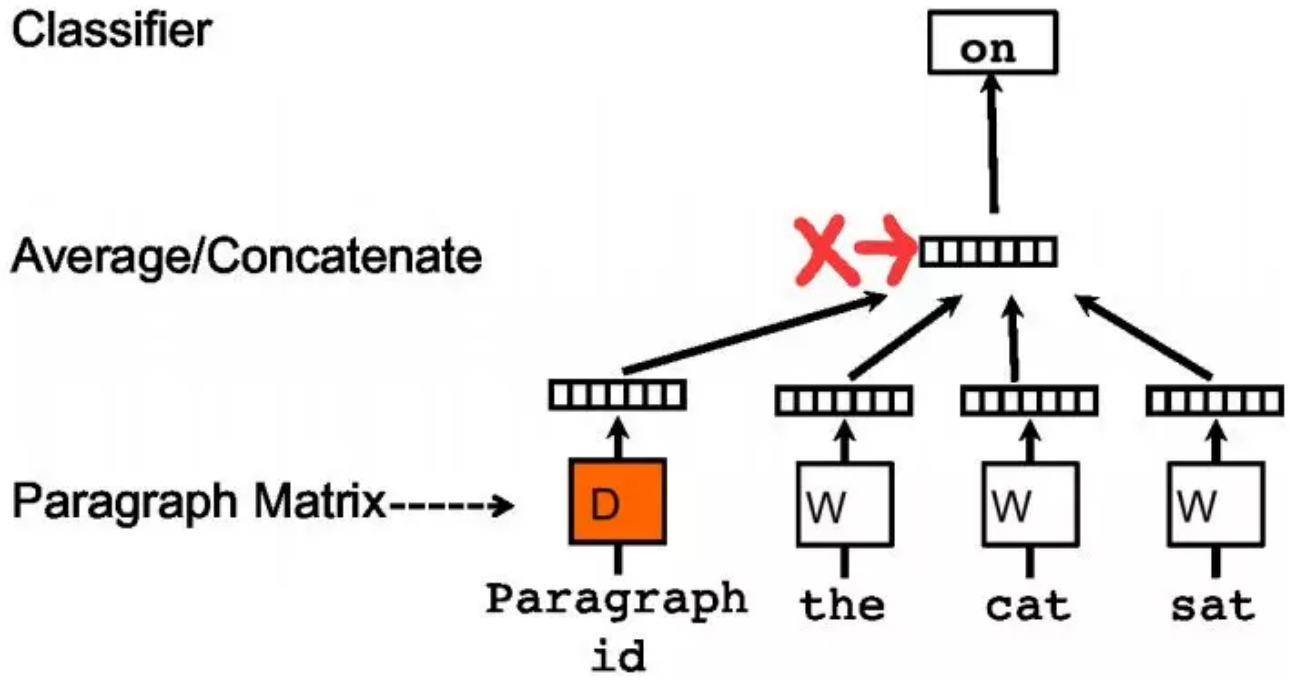
图一

另一种是PV-DBOW（Distributed Bag of Words of paragraph vector)类似于word2vec中的skip-gram模型，如图二：



图二

在Doc2vec中，每一句话用唯一的向量来表示，用矩阵D的某一列来代表。每一个词也用唯一的向量来表示，用矩阵W的某一列来表示。以PV-DM模型为例，如图三：



图三

每次从一句话中滑动采样固定长度的词，取其中一个词作预测词，其他的作输入词。输入词对应的词向量word vector和本句话对应的句子向量Paragraph vector作为输入层的输入，将本句话的向量和本次采样的词向量相加求平均或者累加构成一个新的向量X，进而使用这个向量X预测此次窗口内的预测词。

Doc2vec相对于word2vec不同之处在于，在输入层，**增添了一个新句子向量Paragraph vector**，Paragraph vector可以被看作是另一个词向量，它扮演了一个记忆，词袋模型中，因为每次训练只会截取句子中一小部分词训练，而忽略了除了本次训练词以外该句子中的其他词，这样仅仅训练出来每个词的向量表达，句子只是每个词的向量累加在一起表达的。正如上文所说的词袋模型的缺点，忽略了文本的词序问题。而Doc2vec中的Paragraph vector则弥补了这方面的不足，它每次训练也是滑动截取句子中一小部分词来训练，**Paragraph Vector在同一个句子的若干次训练中是共享的**，所以同一句话会有多次训练，每次训练中输入都包含Paragraph vector。它可以被看作是句子的主旨，有了它，该句子的主旨每次都会被放入作为输入的一部分来训练。**这样每次训练过程中，不光是训练了词，得到了词向量。同时随着一句话每次滑动取若干词训练的过程中，作为每次训练的输入层一部分的共享Paragraph vector，该向**

量表达的主旨会越来越准确。Doc2vec中PV-DM模型具体的训练过程和word2vec中的CBOW模型训练方式相同，在之前我写的基于Word2vec训练词向量（一）里有详细介绍，这里就不在重复。

训练完了以后，就会得到训练样本中所有的词向量和每句话对应的句子向量，那么Doc2vec是怎么预测新的句子Paragraph vector呢？其实在预测新的句子的时候，还是会将该Paragraph vector随机初始化，放入模型中再重新根据随机梯度下降不断迭代求得最终稳定下来的句子向量。**不过在预测过程中，模型里的词向量还有投影层到输出层的softmax weights参数是不会变的，这样在不断迭代中只会更新Paragraph vector**，其他参数均已固定，只需很少的时间就能计算出带预测的Paragraph vector。

二. 代码实现

在python中使用gensim包调用Doc2vec方便快捷，在这简单演示下，gensim下Doc2vec详细的参数不在此详细阐述。本次的数据是之前比赛中公开的旅游数据集，里边每一条都是游客对于景点的评价。具体的Doc2vec训练Paragraph vector步骤如下：

1)导包：导入必要的包，其中的jieba是为了给文本进行分词。

```
import gensim
import jieba
import pandas as pd
import os
from gensim.models.doc2vec import Doc2Vec
```

2)导入数据集，提取Discuss列（该列是用户评价的内容）。

```
os.chdir(r'F:\YUNEDU2018\YNU.EDU2018-ScenicWord')
```

```
def getText():
    df_train=pd.read_csv('train_first.csv')
    discuss_train=list(df_train['Discuss'])
    return discuss_train
```

```
text=getText()
text
```

```
'迪斯尼一日游',
'方便',
'看水看山都可以。感受古人的智慧结晶，秋景美丽如画，红黄绿相间！对于身体状况不佳的人来说，走平路还是可以接受的！',
'赞',
'唯一糟点',
'周末周边游',
'景点服务不错，就是排队 太长了，好玩的项目都是人，晚上的烟火一定jrvytqlamf要看，真的不错，做好攻',
'绍兴护城河夜游',
'感觉还不错，作为一日游不错的选择~',
'有趣hai\u2006xing',
'荡气回肠，10年去的，居然没有留下来照片，必然要再去！<br />n',
'景色超级棒，有美丽的故事，可以乘船游览，也可以沿湖浏览，累了可以乘坐观光车！关键是没有门票！！！',
'南锣鼓巷是北京市中心一条老胡同，因为其地理位置靠近什刹海，成为北京休闲娱乐的好去处，特别是外国游人去的特别多，一方面
文化商品买，另一方面这里的餐饮商家也各具鲜明的特点，使整个地区有一种中西合璧的感觉。在每次世界杯期间，这里气氛是无比的
定也会被感染的。当然来之前银两必须准备充足了。。。',
'个人感觉就是个卖小商品的地方，还不便宜，但是晚上夜景挺好看',
'性价比超高',
'挺普通的吧，就在楼下拍了几张图片，反正也是进不去的啊',
```

3)将提取好的Discuss列中的内容进行分词，并去除停用词。

```
def cut_sentence(text):
    stop_list=[line[:-1] for line in open('中文停用词.txt')]
    result=[]
    for each in text:
        each_cut=jieba.cut(each)
        each_split=' '.join(each_cut).split()
        each_result=[word for word in each_split if word not in stop_list]
        result.append(' '.join(each_result))
    return result
```

```
b=cut_sentence(text)
b
```

```
'迪斯尼 一日游',
'方便',
'看水 看山 。 感受 古人 智慧结晶 ， 秋景 美丽 如画 ， 红黄绿 相间 ！ 身体 状况不佳 人 来说 ， 走平路 接受 ！',
'赞',
'唯一 糟点',
'周末 周边游',
'景点 服务 不错 ， 排队 太长 ， 好玩 项目 人 ， 晚上 烟火 一定 jrvytqlamf ， 真的 不错 ， 做好 攻',
'绍兴 护城河 夜游',
'感觉 不错 ， 一日游 不错 选择',
'有趣 hai xing',
'荡气回肠 ， 10 年 ， 留下来 照片 ， 必然 再 ！ br n',
'景色 超级 棒 ， 美丽 故事 ， 乘船 游览 ， 沿湖 浏览 ， 累 乘坐 观光车 ！ 关键 门票 ！ ！ ！',
'南锣鼓巷 北京市 中心 一条 胡同 ， 地理位置 靠近 什刹海 ， 成为 北京 休闲 娱乐 好去处 ， 特别 外国 游人 特别 ， 许多 中国
买 ， 餐饮 商家 各具 鲜明 特点 ， 使 整个 地区 一种 中西合璧 感觉 。 每次 世界杯 期间 ， 气氛 无比 热烈 ， 来到 一定 感染
备 充足 。 。 。',
```

4)改变成Doc2vec所需要的输入样本格式，由于gensim里Doc2vec模型需要的输入为固定格式，输入样本为：[句子，句子序号],这里需要用gensim中Doc2vec里的TaggedDocument来包装输入的句子。

```
TaggededDocument=gensim.models.doc2vec.TaggedDocument
```

```
def X_train(cut_sentence):
    x_train=[]
    for i,text in enumerate(cut_sentence):
        word_list=text.split(' ')
        l=len(word_list)
        word_list[l-1]=word_list[l-1].strip()
        document=TaggededDocument(word_list,tags=[i])
        x_train.append(document)
    return x_train
```

```
c=X_train(b)
c
```

```
TaggedDocument(words=['感觉', '不错', ',', ',', '一日游', '不错', '选择'], tags=[12]),
TaggedDocument(words=['有趣', 'hai', 'xing'], tags=[13]),
TaggedDocument(words=['荡气回肠', ',', ',', '10', '年', ',', ',', '留下来', '照片', ',', ',', '必然', '再', '!', 'br', 'n'], tags=[14]),
TaggedDocument(words=['景色', '超级', '棒', ',', ',', '美丽', '故事', ',', ',', '乘船', '游览', ',', ',', '沿湖', '浏览', ',', ',', '累', '乘坐', '观光车', '!', '关键', '门票', '!', '!', '!'], tags=[15]),
TaggedDocument(words=['南锣鼓巷', '北京市', '中心', '一条', '胡同', ',', ',', '地理位置', '靠近', '什刹海', ',', ',', '成为', '北京', '休闲', '娱乐', '好去处', ',', ',', '特别', '外国', '游人', '特别', ',', ',', '许多', '中国', '符号', '文化', '商品', '买', ',', ',', '餐饮', '商家', '各具', '鲜明', '特点', ',', ',', '使', '整个', '地区', '一种', '中西合璧', '感觉', '。', '每次', '世界杯', '期间', ',', ',', '气氛', '无比', '热烈', ',', ',', '来到', '一定', '感染', '。', '之前', '银两', '准备', '充足', '。', '。', '。', '。'], tags=[16]),
TaggedDocument(words=['个人感觉', '卖', '小商品', '地方', ',', ',', '便宜', ',', ',', '晚上', '夜景', '挺', '好看'], tags=[17]),
TaggedDocument(words=['性价比', '超高'], tags=[18]),
TaggedDocument(words=['挺', '普通', ',', ',', '楼下', '拍', '几张', '图片', ',', ',', '反正', '进不去'], tags=[19]),
TaggedDocument(words=['太大', ',', ',', '里面', '走', '好长时间', '不到', '五分之一', '。', '周围', '交通', '方便', ',', ',', '值得', '看一看'], tags=[20]),
TaggedDocument(words=['迪士尼'], tags=[21]),
TaggedDocument(words=['亲子', '游'], tags=[22]),
TaggedDocument(words=['苏州', '总是', '欣赏', '一下', '古典', '园林', '。', '可惜', '园林', '不太', '感冒', '。', '逛逛', '玩玩', '不错', '。'], tags=[23]),
TaggedDocument(words=['不到长城非好汉', ',', ',', '爬', '华山', '来说', ',', ',', '长城', '太', '简单', ',', ',', '值得', '一去', '。'], tags=[24]),
```

5)加载Doc2vec模型，并开始训练。

```
def train(x_train,size=300):
    model=Doc2Vec(x_train,min_count=1,window=3,size=size,sample=1e-3,nagative=5,workers=4)
    model.train(x_train,total_examples=model.corpus_count,epochs=10)
    return model
```

```
model_dm=train(c)
```

6)模型训练完毕以后，就可以预测新的句子的向量Paragraph vector了，这里用gensim里Doc2Vec.infer_vector()预测新的句子，这里根据经验，alpha（学习步长）设置小一些，迭代次数设置大一些。找到训练样本中与这个句子最相近的10个句子。可以看到训练出来的结果与测试的新句子是有关联的。

```
str1=u'不到长城非好汉 爬 华山 长城 太 简单 值得 一去'
test_text=str1.split(' ')
#the default parameters of infer_vector() may be far from optimal for many models.
#In particular, a larger steps (at least as large as the number of model training iterations,
#but perhaps even many times larger) is often helpful
#Also, a smaller starting alpha, perhaps just the common default for bulk training of 0.025, may give better results
inferred_vector=model_dm.infer_vector(doc_words=test_text,alpha=0.025,steps=500)
```

```
sims=model_dm.docvecs.most_similar([inferred_vector],topn=10)
```

```
#TaggedDocument有两个值，一个是word，一个是tage。分别用X[0],X[1]来表示
for count, sim in sims:
    print(count,sim)
    sentence = text[count]
#    sentence=c[count]
    words = ''

    for word in sentence:
#    for word in sentence[0]:
        words = words + word + ' '
#    print(words,sim,len(sentence[0]))
    print (words,sim,len(sentence))
```

```
24 0.9122493267059326
不 到 长 城 非 好 汉 ， 对 于 爬 过 华 山 的 我 来 说 ， 长 城 太 简 单 了 ， 值 得 一 去 。  0.9122493267059326 31
47053 0.7222123146057129
人 多 的 时 候 不 要 去 ， 太 挤 兑 人 了 ， 根 本 没 办 法 移 动 。 长 城 爬 上 去 还 是 比 较 累 的 了 ， 我 们 那 时 候 是 没 爬 到 顶 吧
 0.7222123146057129 47
97821 0.6355876326560974
不 到 长 城 非 好 汉 ， 来 北 京 一 定 要 到 八 达 岭 长 城 看 看 ， 太 雄 伟 了 ！ < b r  / > n  0.6355876326560974 35
55282 0.6207307577133179
太 适 合 家 庭 游  0.6207307577133179 6
55926 0.6203120350837708
电 子 票 太 方 便 了  0.6203120350837708 7
89156 0.6135756969451904
有 山 有 水 有 长 城  值 得 一 去  0.6135756969451904 12
```

三. 总结

Doc2vec是基于Word2vec基础上构建的，相比于Word2vec，Doc2vec不仅能训练处词向量还能训练处句子向量并预测新的句子向量。Doc2vec模型结构相对于Word2vec，不同点在于在输入层上多增加了一个Paragraph vector句子向量，该向量在同一句下的不同的训练中是权值共享的，这样训练出来的Paragraph vector就会逐渐在每句子中的几次训练中不断稳定下来，形成该句子的主旨。这样就训练出来了我们需要的句子向量。在预测新的句子向量时，是需要重新训练的，此时该模型的词向量和投影层到输出层的soft weights参数固定，只剩下Paragraph vector用梯度下降法求得，所以预测新句子时虽然也要放入模型中不断迭代求出，相比于训练时，速度会快得多。本次使用的数据集为情感分析，且大多数样本偏向于好评，样本内容比较单一，所以训练出来的结果都是偏向于哪里好玩，好不好这类的意思，对于一些特定的问题之类的句子准确性还没有验证，目前用于情感分析还是可以的。下次会尝试使用新的数据集，调试参数看是否会取得更好的结果。

> *Tips：欢迎大家点击最下方二维码关注我们的公众号，点击**干货资源专栏或发送关键字“资源”**获取更多资源推荐。关注我们的历史文章，一起畅游在深度学习的世界中。*

欢迎扫码进入相应技术交流群：