NLP学习

1、NLTK知识点概要

```
# 使用自己的预料库
from nltk.corpus import PlaintextCorpusReader
corpus_root = '.'  # 设置路径
wordlists = PlaintextCorpusReader(corpus_root, ".*")  # 查询
wordlists.fileids()  # 查看文件夹内容

wordlists.words('my_text.txt')  # 使用
```

词性标注器：训练标注词性数据集，用于词性标注

- 默认标注器（以词频为依据进行标注）
- 正则表达式标注器（正则匹配为依据进行标注）
- 查询标注器（以高频词为依据标注，不匹配则标注为空）
- N-gram（挑选上下文中最可能的词性进行标注；一元以该词词频为准，其他以前N-1词性为准，存在数据稀疏问题）
- 组合标注器（类似于if-else形式来选定标注器）
- Brill基于转换的标注（猜词性，然后根据转换规则进行修改）

文本分类：对文本分类

- 编写特征提取规则，生成特征字典：（特征，标签）
- 使用指定分类器进行训练

> 如根据名字进行性别鉴定：特征可以是名字最后N位；句子分割：特征可以是下一个字符+前一个字符+前一个字符是否是单字符

2、实践

电影评论情感分析

环境：Python 3.7 + Tensorflow 1.14

语言：英语评论

数据集：网上开源标注数据（POS/NEG），共1w左右评论

方法：【数据处理】：选取高频词生成词汇表 + 编码每段评论生成词向量 + 批处理；【模型】：2层LSTM实现

结果：训练+测试很快，准确率在80%左右

古诗生成器

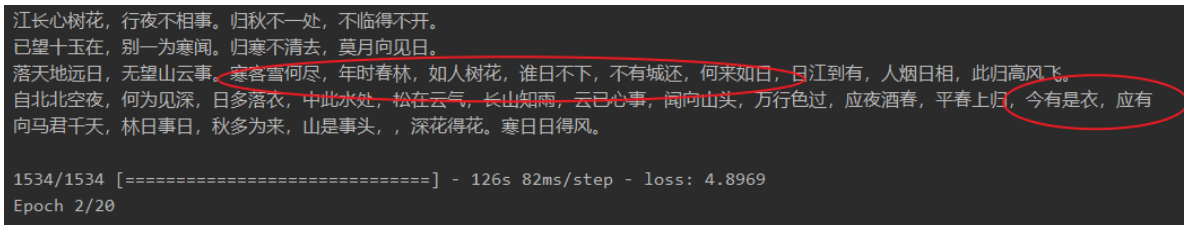
环境：Python 3.7 + Tensorflow 2.0.0

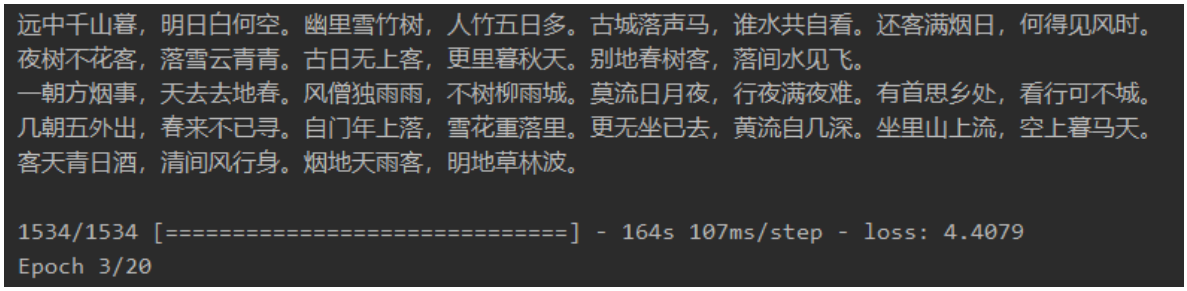
语言：汉语古诗

数据集：网上开源古诗（诗名：诗句），24544首古诗

方法：【数据处理】：过滤低频词 + 添加特殊标记制作词典 + 填充词向量 + 创建批量迭代器；【模型】：2层LSTM + 对每一个时间点都进行预测 + 每个epoch完成后保存最优模型并随机N首生成古诗

格式





随机生成一首古体诗：

```
金鹤有僧心，临天寄旧身。
石松惊枕树，红鸟发禅新。
不到风前远，何人怨夕时。
明期多尔处，闲此不依迟。
```

```
水泉临鸟声，北去暮空行。
林阁多开雪，楼庭起洞城。
夜来疏竹外，柳鸟暗苔清。
寂寂重阳里，悠悠一钓矶。
```

续写一首古体诗（以"床前明月光，"为例）：

```
床前明月光，翠席覆银丝。
岁气分龙阁，无人入鸟稀。
圣明无泛物，云庙逐雕旗。
永夜重江望，南风正送君。
```

```
床前明月光，清水入寒云。
远景千山雨，萧花入翠微。
影云虚雪润，花影落云斜。
独去江飞夜，谁能作一花。
```

随机生成一首藏头诗（以"海阔天空"为例）：

```
海口多无定，
阔庭何所难。
天山秋色上，
空石昼尘连。
```

```
海庭愁不定，
阔处到南关。
天阙青秋上，
空城雁渐催。
```

BERT

BERT是一种预训练语言表示的方法，这意味着我们在大型文本语料库（例如Wikipedia）上训练通用的“语言理解”模型，然后将该模型用于我们关心的下游NLP任务（例如问题回答）。 BERT优于以前的方法，因为它是第一个用于预训练NLP的无监督，深度双向系统。

1、基本概念

> **BERT基本组成**为Seq2Seq网络：输入和输出都为序列的网络，中间为Transformer

为什么要Transformer? 因为传统RNN网络每个循环体都需要上一个循环体的中间结果，无法并行运算（不独立）；采用Self-Attention机制并行计算，取代RNN

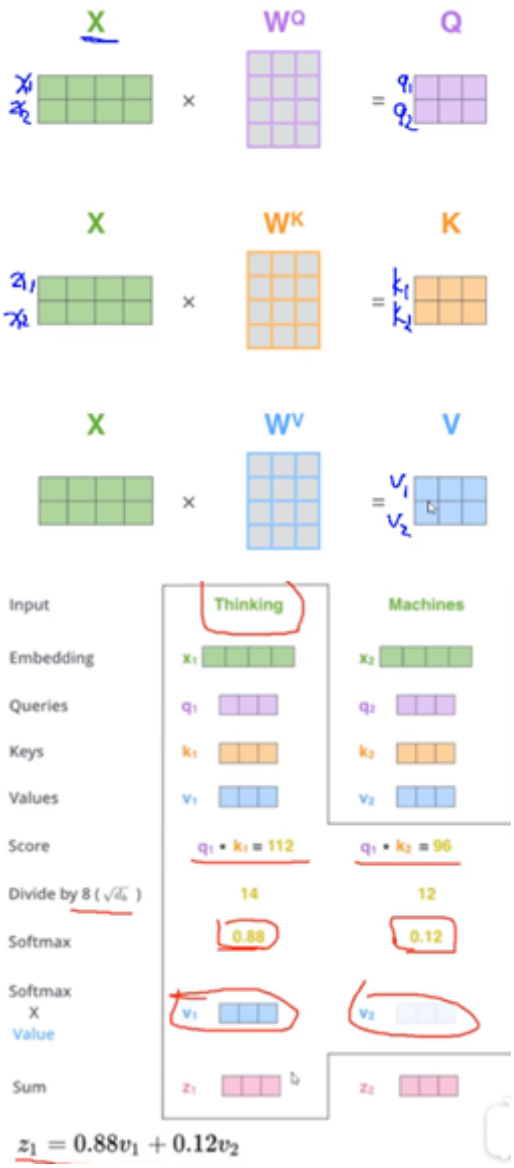
为什么不用word2vec? 相同的词表达含义一样，实际上不是这样的

Self-Attention机制：在不同的语境下不同的词有不同的权重。

方法：词->向量编码x1->权重向量编码z1

预先定义三个矩阵（权重）Wq、Wk、Wv，分别表示要被查询的、等待被查的和实际的特征信息，使输入词向量分别与三个矩阵进行运算。然后用softmax来进行归一化，求得每个词在当前句子中占有的权重（影响程度）

若要查询第一个词与每个词的关系，则用Wq1*Wk1（内积：若无关系则内积为0）表示与第一个词的关系，Wq1*Wk2表示与第二个词的关系，以此类推



Multi-headed机制：提取多种词向量特征（上述只是提取一种特征）

通过多个头机制（一般8个）得到多个特征表达，然后将所有特征拼接起来，再加一层全连接来降维

注：通过上述过程得到词特征向量，一般情况下还要堆叠多层来得到最终的词特征向量

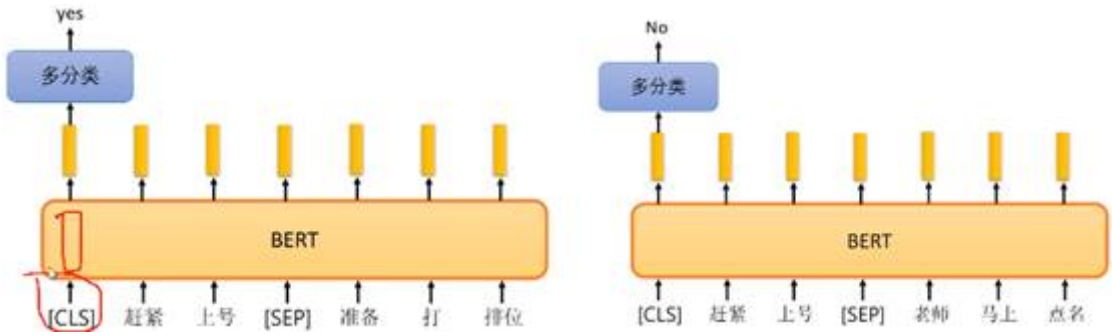
位置信息表达：相对于上述得到的词特征向量再加上位置信息编码，一般是周期信号（正余弦等）

LayerNormalize：对每个词的所有特征进行归一化处理（区别于对每批数据进行归一化），为了使得训练更快，更稳定。其中词的特征向量由上述处理得到

连接：层归一化的同时，加入残差连接

Decoder：相对于Encoder，Self-Attention计算不同（用q来查），加入了MASK机制。

训练BERT：将句子中15%（替换为mask，随机变为其他词或不变）的词随机mask掉，让模型去预测被mask的是什么；预测两个句子是否应该连在一起（[seq]连接符，[cls]分类向量）



2、源码简介

数据：

1、加载数据（一般是两句话或一句话）==> 按模板格式化数据 + 定义标签

2、制作TF_record格式数据集==> 制作label字典（label：index） + 数据分词 + 数据补全（并加CLS等标识符） + 词典查ID（同时设置input_mask，作用是指示哪些位置的词是有效的） + 段标识符（用来指示这是第几个话，0/1表示。最后用0补全长度）

```
546     # 生成tf_record文件
547     writer = tf.python_io.TFRecordWriter(output_file)
548
549     # 遍历每一条数据
550     for (ex_index, example) in enumerate(examples):
551       if ex_index % 10000 == 0:
552         tf.logging.info("Writing example %d of %d" % (ex_index, len(examples)))
553       # 对每个样本制作feature
554       feature = convert_single_example(ex_index, example, label_list,
555                                        max_seq_length, tokenizer)
556
557       def create_int_feature(values):
558         f = tf.train.Feature(int64_list=tf.train.Int64List(value=list(values)))
559         return f
560
561       features = collections.OrderedDict()
562       features["input_ids"] = create_int_feature(feature.input_ids)
563       features["input_mask"] = create_int_feature(feature.input_mask)
564       features["segment_ids"] = create_int_feature(feature.segment_ids)
565       features["label_ids"] = create_int_feature([feature.label_id])
566       features["is_real_example"] = create_int_feature(
567           [int(feature.is_real_example)])
568
569       tf_example = tf.train.Example(features=tf.train.Features(feature=features))
570       writer.write(tf_example.SerializeToString())
571     writer.close()
```

模型：

> 刚开始num_warmup_steps次迭代学习率偏小，之后恢复设置的学习率

构建Embedding层===>

词向量编码（和变换维度）+位置和段从属关系嵌入（相加）+层归一化（和dropout）

构建Transformer层（encoder，自注意机制）===>

词向量扩维（目的是为了使每一个词对应每一个句子；同时创建mask标记有效性，即实现自注意）+多头机制（每个头专注于N种特征/输出的提取，后面是对每一个头的处理）+Q矩阵初始化（len(输入)xlen(特征)）+K矩阵初始化（len(输出)xlen(特征)）+V矩阵初始化（len(输出)xlen(特征)）+QK内积（接着结合mask标记运算，类似于单神经网络）+结果与V内积+（池化+训练等）

附：

（一）skip-gram

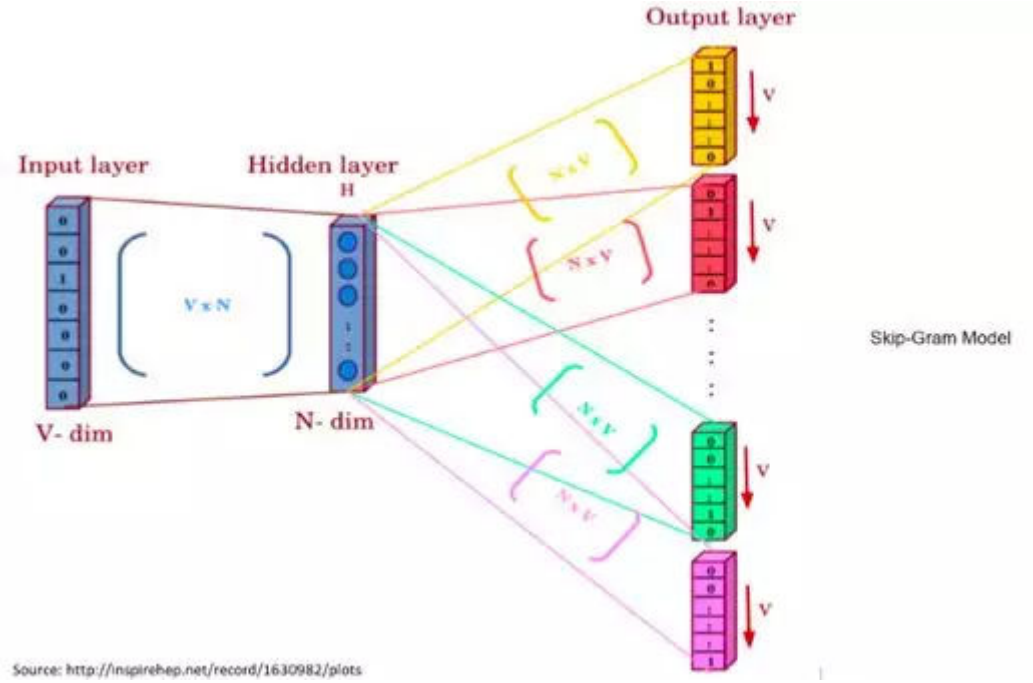
> 分布相似性==>密集型向量；与词汇相对位置无关

skip-gram：利用中心词来预测上下文。

1、利用one-hot编码将单词转换为向量，维度为[1, V]（V一般表示总次数）

2、与权重向量W[V, N]（N是上下文窗口）点积运算，无激活函数，最后得到v个向量H[1, N]（相对每一个上下文词都会生成一个向量）

3、与输出层权重向量W'[N, V]点积得到向量U[1, V]，再经过softmax函数得到one-hot编码形式的预测向量。概率最大的那个单词就是结果，若预测有误，会通过反向传播算法来修正权重矩阵W和W'。



奇异值分解（SVD）：

有一个$m \times n$的实数矩阵A，我们想要把它分解成如下的形式

$$A = U\Sigma V^T \tag{2-1}$$

其中U和V均为单位正交阵，即有$UU^T = I$和$VV^T = I$，U称为 左奇异矩阵 ，V称为 右奇异矩阵 ，Σ仅在主对角线上有值，我们称它为 奇异值 ，其它元素均为0。上面矩阵的维度分别为$U \in R^{m \times m}$, $\Sigma \in R^{m \times n}$, $V \in R^{n \times n}$。

一般地Σ有如下形式

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \end{bmatrix}_{m \times n}$$

其中对于奇异值，取其前面的某些值便可还原原本的数据（即使有丢失但是很少），且奇异值在变化上是非常大的，后面的值相对原数据影响最小。

> **交叉熵**（Cross Entropy）： 主要用于度量两个概率分布间的差异性信息。

（二）NLP中文数据集

GITHUB：https://github.com/InsaneLife/ChineseNLPCorpus

- 任务型对话（语音+文本理解）
- 文本分类（新闻、评论、微博等）
- 实体识别/标注（微博、人民日报、微软亚洲研究院数据集等）
- 搜索匹配
- 推荐系统（电影、餐馆、商品等）
- 百科数据（百度百科、维基百科）
- 指代消岐
- 完形填空
- 中华诗词
- 保险行业
- 汉语拆字字典
- 预训练模型BERT及词向量
- NLP工具（5种）