浙江大学

题目：	Word2Vec && TextCNN实验报告
姓名：	余丛杉
学号：	3190103165
班级：	混合1902
性别：	女
日期：	2022.4.6

Word2Vec && TextCNN实验报告

1 Project Introduction

1.1 Word2Vec

1.1.1 选题

词向量（Word embedding），又叫 Word 嵌入，是自然语言处理（NLP）中的一组语言建模和特征学习技术的统称，其中来自词汇表的单词或短语被映射到实数的向量。

1.1.2 工作简介

本实验主要在 ModelArts 平台上完成词向量的训练，并应用于语义相似词的搜索、扩展。

1.1.3 开发环境及系统运行要求

本实验的开发环境为ModelArts Ascend Notebook 环境，基于`Python 3.7` 和`Gensim`框架实现Word2Vec在Wikipedia语料集上面的应用,并且获取词的词向量以及寻找相近词。

1.2 TextCNN

1.2.1 选题

文本分类(text classification),又称文档分类(document classification),指的是将一个文档归 类到一个或多个类别中的自然语言处理任务。

1.2.2 工作简介

本实验主要基于卷积神经网络对电影评论信息进行情感分析，判断其情感倾向。

1.2.3 开发环境及系统运行要求

本实验开发环境为ModelArts Ascend Notebook环境，基于`MindSpore1.1`搭建 TextCNN模型用于情感分析。

2 Technical Details

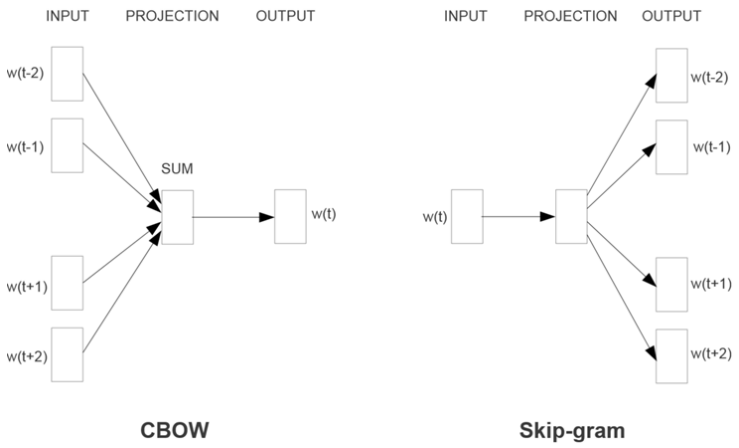
2.1 Word2Vec

2.1.1 理论知识

Word2Vec是用一个一层的神经网络(CBOW & Skip-Gram)把one-hot形式的稀疏词向量映射称为一个n维的稠密向量的过程。为了加快模型训练速度，其中的tricks包括Hierarchical softmax，negative sampling, Huffman Tree等。

NLP中的词语，是人类的抽象总结，是符号形式的（比如中文、英文、拉丁文等等），所以需要把他们转换成数值形式，或者说嵌入到一个数学空间里，这种嵌入方式，就叫词嵌入（Word Embedding），而Word2vec，就是词嵌入的一种。

Word2Vec有两个重要的模型CBOW模型(Continuous Bag-of-Words Model)与Skip-gram模型。在Tomas Mikolov的论文中给出了示意图。CBOW就是根据某个词前面的C个词或者前后C个连续的词，来计算某个词出现的概率。Skip-Gram Model相反，是根据某个词，然后分别计算它前后出现某几个词的各个概率。

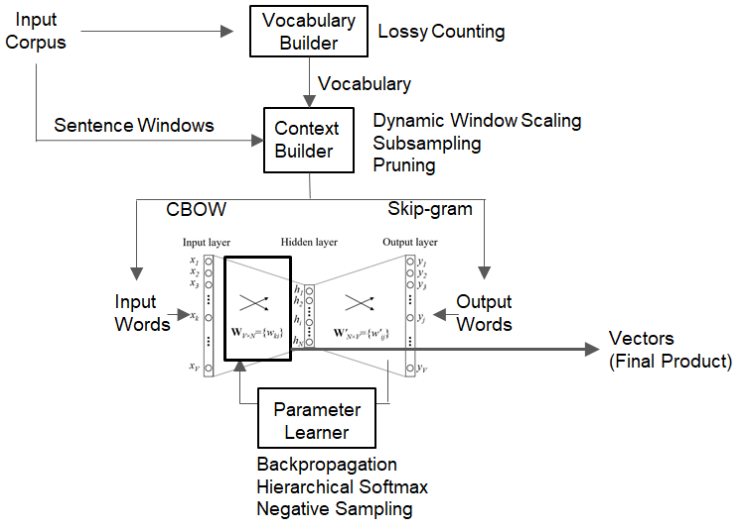


具体计算过程为：

从$input \rightarrow hidden$：$W^T \times x$， W为$V \times N$矩阵， x为$V \times 1$向量，最最终隐层的结果为 $N \times 1$.

从$hidden \rightarrow output$：$x^T \times W'$，其中x为$N \times 1$向量，W'为$V \times N$，最终结果为$1 \times V$.

2.1.2 具体算法



Word2Vec相当于有两个全连接层，通过反向传播训练$W_{V \times N}$矩阵。

2.1.3 技术细节

Word2Vec的代码来自于实验指导书，没有做任何改动。

- 词向量的训练

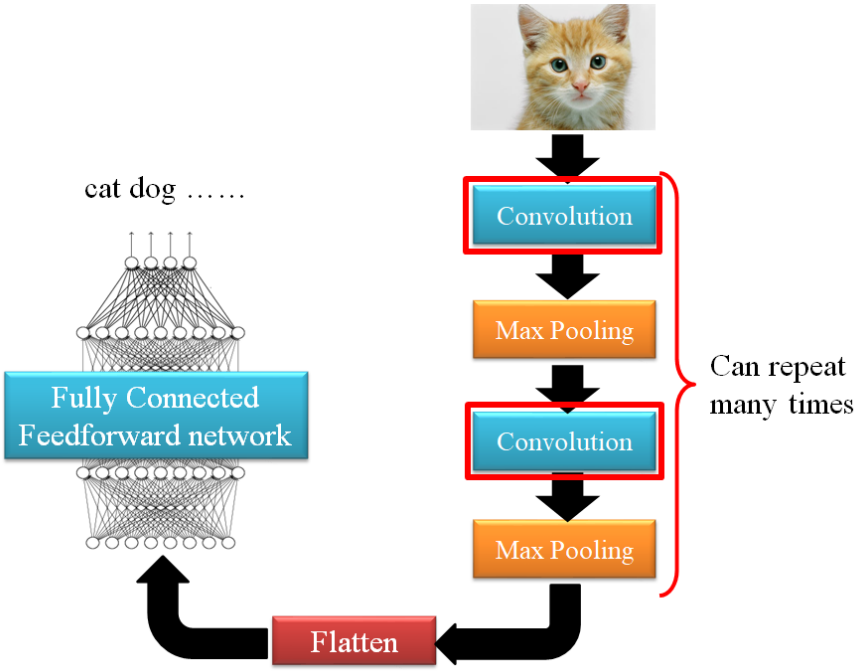
```
model = Word2Vec(corpus_file=corpus_file, vector_size=100, window=5, min_count=5,
workers=cpu_count(), sg=1)  # 调用库函数进行训练
model.wv.save_word2vec_format(out_embedding_file, binary=False) # 保存训练好的模型
```

- 重要函数`Word2Vec  KeyedVectors`主要来自于`gensim.models`库

2.2 TextCNN

2.2.1 理论知识

以图像为例：



CNN一般包括多层的`Convolution`和`Max Pooling`提取特征,经过`Flatten`和`Fully Connected Feedforward network`完成分类等不同任务。

其中卷积层是关键，由全连接层利用平移不变性和局部性，得到卷积层：

$$[\mathbf{H}]_{i,j} = u + \sum_{a=-\Delta}^{\Delta}\sum_{b=-\Delta}^{\Delta}[\mathbf{V}]_{a,b}[\mathbf{X}]_{i+a,j+b}.$$

V被称为卷积核（convolution kernel）或者滤波器（filter），亦或简单地称之为该卷积层的权重。使用$[X]_{i,j}$和$[H]_{i,j}$分别表示输入图像和隐藏表示中位置(i,j)处的像素。

2.2.2 具体算法

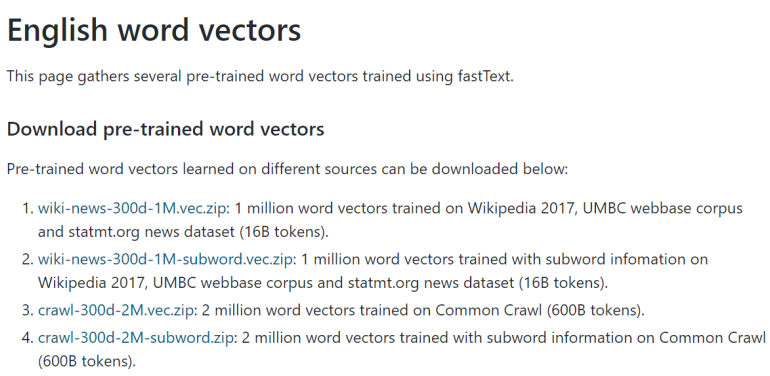
传统上，CNN是用来分析图像的，它由一个或多个卷积层和一个或多个线性层组成。卷积层使用过滤器(也称卷积核)扫描一张图像，并生成一张经过处理的图像。这个处理过的图像版本可以被送入另一个卷积层或线性层。每个过滤器都有一个形状，例如一个3×3滤镜覆盖图像的3像素宽×3像素高的区域，滤镜的每个元素都有一个权重，3×3滤镜有9个权重。在传统的图像处理中，这些权值是由工程师手工指定的，然而，神经网络中的卷积层的主要优点是这些权值是通过反向传播学习的。

学习权重背后的直觉思想是卷积层表现得像特征提取器,提取图像的部分是CNN最重要的目标,例如,如果使用一个CNN在一个图像中检测面部,CNN可能在图像中寻找鼻子,嘴巴和一双眼睛的特征。以同样的方式，一个3×3滤镜可以查看一个图像的碎片，一个1×2滤镜可以查看一段文本中的2个连续的单词，即`bi-gram`。在CNN模型中,可以使用多个大小不同的过滤器来观察`bi-grams`(1×2 filter),`tri-grams` (1×3 filter)和`n-grams`($1\times n$ filter)内的文本。

2.2.3 技术细节

这个实验的代码来自于实验指导书，在跑通此代码的基础上，在`Text2Vec`做了`Word2Vec`的尝试。以下主要讲述修改的部分

- `Word2Vec`加载模型



English word vectors

This page gathers several pre-trained word vectors trained using fastText.

Download pre-trained word vectors

Pre-trained word vectors learned on different sources can be downloaded below:

1. wiki-news-300d-1M.vec.zip: 1 million word vectors trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16B tokens).
2. wiki-news-300d-1M-subword.vec.zip: 1 million word vectors trained with subword infomation on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16B tokens).
3. crawl-300d-2M.vec.zip: 2 million word vectors trained on Common Crawl (600B tokens).
4. crawl-300d-2M-subword.zip: 2 million word vectors trained with subword information on Common Crawl (600B tokens).

把训练好的向量集加载在word2vec文件夹下，并同步数据至本地。

```
word2vec_model = KeyedVectors.load_word2vec_format("./word2vec/wiki-news-300d-1M.vec")
```

此向量集中的向量维度为300，但是实验环境不支持300维度的向量计算（太大了），所以我在进入CNN网络之前加了一层全连接层，把输出向量的维度定为200，需要修改`vec_length`超参数：

```
cfg = edict({
 'name': 'movie review',
 'pre_trained': False,
 'num_classes': 2,
 'batch_size': 64,
 'epoch_size': 4,
 'weight_decay': 3e-5,
 'data_path': './data/',
 'device_target': 'Ascend',
 'device_id': 0,
 'keep_checkpoint_max': 1,
 'checkpoint_path': './ckpt/train_textcnn-4_149.ckpt',
 'word_len': 51,
 'vec_length': 200  # 输入到网络的向量维数,其余参数不变
})
```

- 修改`text2vec`函数

```
def text2vec(self, maxlen):
        '''
        将句子转化为向量
        '''
        for SentenceLabel in self.Pos+self.Neg: # 取出文本中的每一句
```

```
            vector = np.zeros((maxlen, 300), dtype=np.float32)  # 初始化vector向量，并定义维度大小和数据类型
            for index, word in enumerate(SentenceLabel[0]): # 取出句子的编号，把句子打碎成word
                if index >= maxlen:
                    break
                if word not in word2vec_model:  # 如果word不在加载的模型中，则continue
                    continue
                vector[index] = word2vec_model[word] # 取出word的向量并赋值

            SentenceLabel[0] = vector
        self.doConvert = True
```

- 网络结构

  使用`nn.Conv2d`来实现卷积层。`in_channels`参数是图像进入卷积层的“通道”的数量。`out_channels`是过滤器的数量，`kernel_size`是过滤器的大小。每个`kernel_sizes`大小都是$[n \times emb_dim]$，其中n是`n-grams`的大小。

```
def construct(self,x):
        x = self.unsqueeze(x, 1)    # ExpandDims
        x = self.ReLu(self.embed_fc(x)) # 经过一层全连接层
        x1 = self.layer1(x)
        x2 = self.layer2(x)
        x3 = self.layer3(x) # 三种不同卷积核
        x1 = self.reducemean(x1, (2, 3)) # 调用ops.ReduceMax(keep_dims=False)，求最大值
        x2 = self.reducemean(x2, (2, 3))
        x3 = self.reducemean(x3, (2, 3))
        x = self.concat((x1, x2, x3)) # 分别做卷积后拼接在一起
        x = self.drop(x) # 如果训练样本较少，为了防止模型过拟合，Dropout可以作为一种trikc供选择
        x = self.fc(x)  # 全连接层
        return x
```

- 数据类型

  解决完向量维度之后，还需要修改数据类型，由于加载的模型向量值在1e-2大小，所以需要修改输入类型为`np.float32`.

3 Experiment Results

3.1 Word2Vec

- 获取单个词向量

```python
word2vec_model['中国']
```

```
array([-0.11945462,  0.80811197, -0.2192669 , -0.35121506, -0.2546237 ,
       -0.00980086,  0.7964671 ,  0.36271024,  0.31742495, -0.3251099 ,
       -0.13367757, -0.512894  ,  0.18774055,  0.28745985, -0.08601924,
        0.0150877 ,  0.73333174, -0.6884307 ,  0.09896889, -0.4288761 ,
        0.7132578 ,  0.396008  ,  0.4390735 , -0.09943724,  0.20115437,
        0.07450946,  0.01946275,  0.5919555 ,  0.11550876, -0.2978358 ,
       -0.18850829, -0.279784  , -0.56403273, -0.6189067 , -0.2669414 ,
       -0.1924906 ,  0.56643754,  0.05672867,  0.20835687,  0.53800774,
        0.05601315,  0.01130022, -0.13399325,  0.01399448,  0.44789463,
       -0.16624065,  0.23680332,  0.48087487,  0.2066727 ,  0.30111927,
        0.23712948, -0.01020607,  0.09069001, -0.35201445,  0.30945036,
       -0.29315245,  0.0818078 , -0.15353864, -0.21203907,  0.33678278,
       -0.26471385, -0.40392622,  0.12019241,  0.37057823,  0.10111269,
        0.29821014,  0.1682322 ,  0.3837336 , -0.48044497,  0.00659311,
       -0.12619178,  0.08527908, -0.10650562,  0.01504959, -0.17726953,
       -0.42102095,  0.4874898 ,  0.4325759 ,  0.54365766, -0.02363636,
       -0.3604603 ,  0.21117018, -0.56193644, -0.2412196 ,  0.33035398,
       -0.09440871,  0.8251365 ,  0.16836917, -0.2605197 ,  0.40633464,
        0.7236247 , -0.36978406,  0.37283292,  0.40109146, -0.09860536,
       -0.48287332,  0.05875314,  0.15065585, -0.8095557 ,  0.62114567],
      dtype=float32)
```

成功输出中国的embedding向量。

- 相似度测试

```python
testwords = ['金融', '喜欢', "中国", "北京"]
for word in testwords:
    res = word2vec_model.most_similar(word)
    print (word)
    print (res)
```

```
金融
[('金融服务', 0.7745651006698608), ('证券期货', 0.772100031375885), ('信贷', 0.7595240473747253), ('投资银行',
0.7579774260520935), ('国际金融', 0.7491804361343384), ('银行学', 0.7486739158630371), ('金融保险', 0.7417786717414856),
('保险业', 0.7401484251022339), ('期货', 0.7384737133979797), ('金融市场', 0.738309383392334)]
喜欢
[('讨厌', 0.7161690592765808), ('吃喝', 0.6951901912689209), ('很会', 0.6930608153343201), ('喝酒', 0.6888067722320557),
('吓人', 0.6877806186676025), ('偏爱', 0.686855137348175), ('迟钝', 0.6796375513076782), ('卖弄', 0.6752473711967468),
('没什么', 0.6695799231529236), ('鞋子', 0.6682382822036743)]
中国
[('大陆', 0.712977409362793), ('顾诚', 0.6032476425170898), ('李泽厚', 0.6030479669570923), ('榜上有名',
0.6003459692001343), ('世界史', 0.5973314046859741), ('中国地图出版社', 0.5888921618461609), ('内地',
0.5883994102478027), ('少帅', 0.5853191018104553), ('戏曲史', 0.5793296098709106), ('瑰宝', 0.5780259370803833)]
北京
[('上海', 0.7301627993583679), ('天津', 0.7032418847084045), ('杭州', 0.6845793128013611), ('北京市', 0.660686194896698),
('沈阳', 0.6576042175292969), ('南京', 0.6466558575630188), ('上海市', 0.6343791484832764), ('武汉', 0.6275659203529358),
('西安', 0.618435263633728), ('首都国际机场', 0.6135655641555786)]
```

可以看出，相关性较强的词，向量值较为接近，达到了embedding想要的效果。

3.2 TextCNN

3.2.1 指导书中的代码测试评估

- 准确度

```
load checkpoint from [./ckpt/train_textcnn-4_596.ckpt].
accuracy:  {'acc': 0.763671875}
```

由图可知，训练好的模型在验证集上的准确度为0.763671875

- 在线测试

```python
review_en = "the movie is so boring"
inference(review_en)
```

```
Negative comments
```

得出"the movie is so boring"是负面影评，符合人类语言。

3.2.2 加入Word2Vec代码评估

- 成功跑通

```
model.train(cfg.epoch_size, dataset, callbacks=[time_cb, ckpoint_cb, loss_cb])
print("train success")
```

```
epoch: 1 step: 596, loss is 0.40294343
epoch time: 103989.798 ms, per step time: 174.480 ms
epoch: 2 step: 596, loss is 0.19032332
epoch time: 52176.938 ms, per step time: 87.545 ms
epoch: 3 step: 596, loss is 0.21809569
epoch time: 52109.314 ms, per step time: 87.432 ms
epoch: 4 step: 596, loss is 0.07222549
epoch time: 52169.522 ms, per step time: 87.533 ms
train success
```

- 准确度

[17]:
```
# 导入训练生成的 checkpoint
checkpoint_path = './ckpt/train_textcnn-4_596.ckpt'
# 生成测试数据集
dataset = instance.create_test_dataset(batch_size=cfg.batch_size)
# 定义评估损失、网络
loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
net = TextCNN(vocab_len=instance.get_dict_len(),word_len=cfg.word_len,
            num_classes=cfg.num_classes,vec_length=cfg.vec_length)
if checkpoint_path is not None:
    param_dict = load_checkpoint(checkpoint_path)
    print("load checkpoint from [{}].".format(checkpoint_path))
else:
    param_dict = load_checkpoint(cfg.checkpoint_path)
    print("load checkpoint from [{}].".format(cfg.checkpoint_path))
load_param_into_net(net, param_dict)
net.set_train(False)
model = Model(net, loss_fn=loss, metrics={'acc': Accuracy()})

acc = model.eval(dataset)
print("accuracy: ", acc)
```

```
load checkpoint from [./ckpt/train_textcnn-4_596.ckpt].
accuracy:  {'acc': 0.8212890625}
```

由测试结果可知，准确度为0.8212890625，高于前一种方法，得到了不错的效果。

4 References:

［1］ TextCNN中Word2Vec数据来源 https://fasttext.cc/docs/en/english-vectors.html

［2］ Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space[J]. arXiv preprint arXiv:1301.3781, 2013.

［3］ Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality[J]. Advances in neural information processing systems, 2013, 26.

［4］ Aston Zhang, Zack C. Lipton, Mu Li, et al. Dive into Deep Learning https://d2l.ai/chapter_convolutional-neural-networks/why-conv.html

［5］ 华为实验指导书《《自然语言处理》 -文本分类》

［6］ 华为实验指导书《《自然语言处理》 -词向量实验》