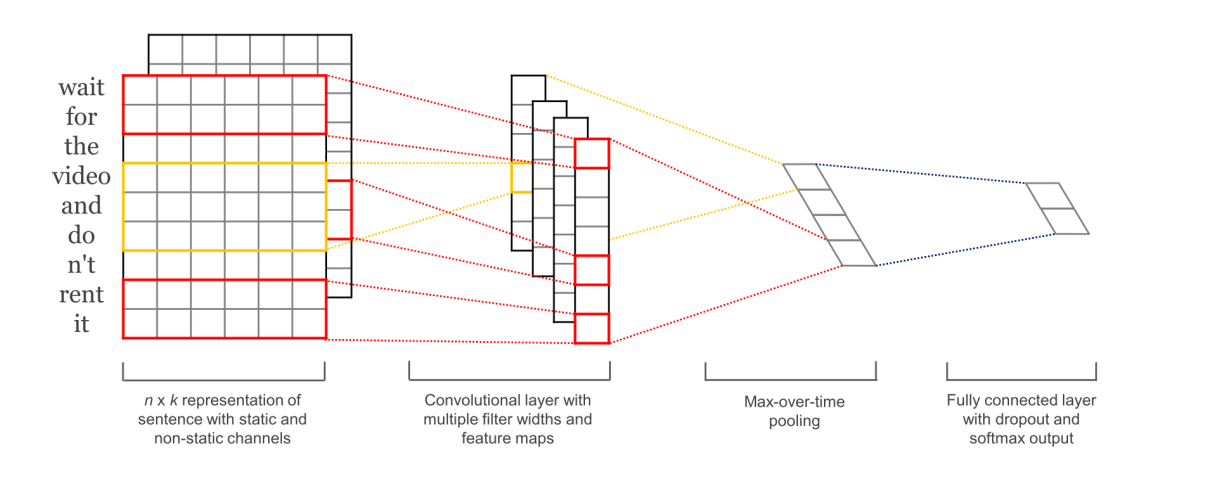
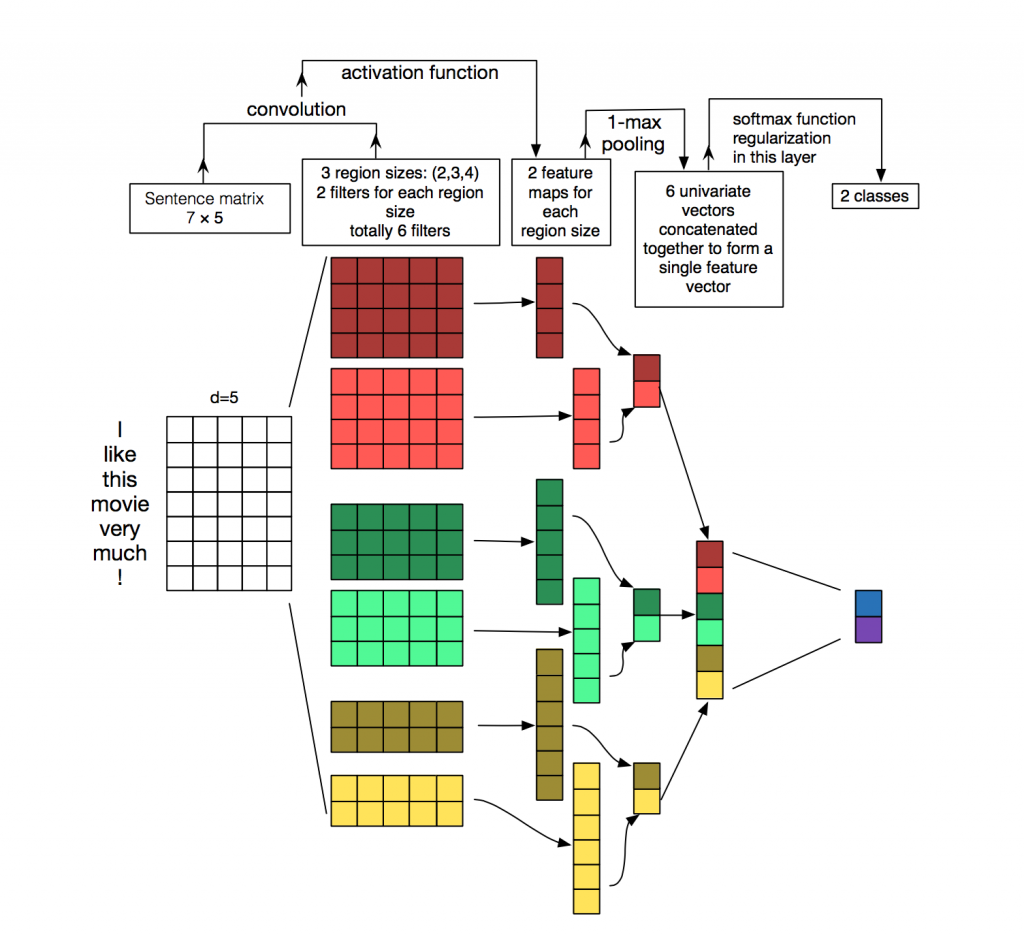
**Text-CNN 文本分类**

**1.简介**

TextCNN 是利用卷积神经网络对文本进行分类的算法，由 Yoon Kim 在 “Convolutional Neural Networks for Sentence Classification” 一文 (见参考[1]) 中提出. 是2014年的算法.

  
图1-1 参考[1] 中的论文配图

以下是阅读TextCNN后的理解

步骤：

　　1.先对句子进行分词，一般使用“jieba”库进行分词。

　　2.在原文中，用了6个卷积核对原词向量矩阵进行卷积。

卷积具有局部特征提取的功能, 所以可用 CNN 来提取句子中类似 n-gram 的关键信息.

3.6个卷积核大小：2个4\*6、2个3\*6和2个2\*6，如上图所示；然后进行池化，对相同卷积核产生的特征图进行连接；再进行softmax输出2个类别。

　　　　1).这里对no-static进行阐述，采用不固定的词向量，这样更加贴近自然生活中，不同长度的句子代表的意思不同，所以在我看来采用no-static比static更加的贴近语义。

　　　　2).对一个词向量进行卷积和池化后，产生的特征只有一个1\*1的向量，所以说不管采用static和no-static得到的特征都只有一个，对本生网络没什么影响。

4.**为什么采用不同大小的卷积核**，不同的感受视野，卷积核的宽取词汇表的纬度，有利于语义的提取。

体现在代码中

tf.flags.DEFINE\_string("filter\_sizes", "3,4,5", "Comma-separated filter sizes (default: '3,4,5')")

filter\_sizes=list(map(int, FLAGS.filter\_sizes.split(","))),

是一个list

　　5.研究证明为什么要采用字，而不采用字，原因是词粒度准确率>字粒度准确率。存在两种模型，一种是词袋模型，第二种是词向量模型。下面对词向量模型来进行讲述。

　　　　词向量模型：

　　　　一般开始为高纬度，高稀疏向量，利用嵌入层对其进行降维，增加稠密性。

　　　　使用词向量进行文本分类的步骤为：

　　　　　　①.先使用分词工具提取词汇表。

　　　　　　②.将要分类的内容转换为词向量。

　　　　　　　　a.分词

　　　　　　　　b.将每个词转换为word2vec向量。

　　　　　　　　c.按顺序组合word2vec，那么久组合成了一个词向量。

　　　　　　　　d.降维，由原来的高纬度降维为我们设定的低纬度。

　　　　　　　　e.卷积、池化和连接，然后进行分类。

　　6.嵌入层

　　通过一个隐藏层将word2vec高纬度的词向量转换到低纬度空间的词向量，这个层的本质是特征提取，提取高纬度词向量的特征到低纬度，这样可以使语义相近的词映射到低维空间以后，欧式距离更近。

**2.参数与超参数**

* sequence\_length   
  Q: 对于CNN, 输入与输出都是固定的，可每个句子长短不一, 怎么处理?   
  A: 需要做定长处理, 比如定为n, 超过的截断, 不足的补0. 注意补充的0对后面的结果没有影响，因为后面的max-pooling只会输出最大值，补零的项会被过滤掉.
* num\_classes   
  多分类, 分为几类.
* vocabulary\_size   
  语料库的词典大小, 记为|D|.
* embedding\_size   
  将词向量的维度, 由原始的 |D| 降维到 embedding\_size.
* filter\_size\_arr   
  多个不同size的filter.

**3.Embedding Layer**

**首先用 VocabularyProcessor 将 每一句话 转为 词id向量**



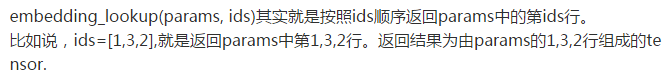
**然后**定义了词嵌入矩阵，将输入的词id转化成词向量，这里的词嵌入矩阵是可以训练的，我们希望得到的是训练完以后，输入经过W矩阵转换得到的固定维度的隐藏层，及词向量矩阵

通过一个词嵌入矩阵, 将 编码的词投影到一个低维空间中.

本质上是特征提取器，在指定维度中编码语义特征. 这样, 语义相近的词, 它们的欧氏距离或余弦距离也比较近.

https://blog.csdn.net/u013041398/article/details/60955847

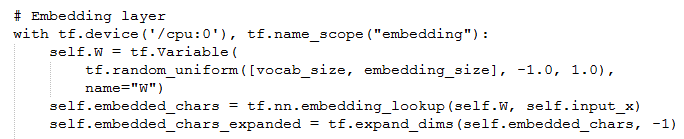
self.embedded\_chars = tf.nn.embedding\_lookup(W, self.input\_x)



如果先用*word2vec\_helpers处理完，然后代用textCNN的情况下，这里的embeding层是不是就不要了？*

*原始的*[*https://github.com/dennybritz/cnn-text-classification-tf*](https://github.com/dennybritz/cnn-text-classification-tf)

*在没有预先使用 word2vec 的情况下：在网络层有embeding层*



而预先使用了*word2vec\_helpers处理完之后，*[*https://github.com/clayandgithub/zh\_cnn\_text\_classify/blob/master/text\_cnn.py*](https://github.com/clayandgithub/zh_cnn_text_classify/blob/master/text_cnn.py)



**4.Convolution Layer**

为不同尺寸的 filter 都建立一个卷积层. 所以会有多个 feature map.   
图像是像素点组成的二维数据, 有时还会有RGB三个通道, 所以它们的卷积核至少是二维的.   
从某种程度上讲, word is to text as pixel is to image, 所以这个卷积核的 size 与 stride 会有些不一样.

* xixi   
  xi∈Rkxi∈Rk, 一个长度为n的句子中, 第 i 个词语的词向量, 维度为k.
* xi:jxi:j   
  xi:j=xi⊕xi+1⊕...⊕xjxi:j=xi⊕xi+1⊕...⊕xj   
  表示在长度为n的句子中, 第 [i,j] 个词语的词向量的拼接.
* hh   
  卷积核所围窗口中单词的个数, 卷积核的尺寸其实就是 hkhk.
* ww   
  w∈Rhkw∈Rhk, 卷积核的权重矩阵.
* cici   
  ci=f(w⋅xi:i+h−1+b)ci=f(w⋅xi:i+h−1+b), 卷积核在单词i位置上的输出. b∈RKb∈RK, 是 bias. ff 是双曲正切之类的激活函数.
* c=[c1,c2,...,cn−h+1]c=[c1,c2,...,cn−h+1]   
  filter在句中单词上进行所有可能的滑动, 得到的 feature mapfeature map.

**5.Max-Pooling Layer**

max-pooling只会输出最大值, 对输入中的补0 做过滤.

**6.SoftMax 分类 Layer**

最后接一层全连接的 softmax 层，输出每个类别的概率。

**7.小的变种**

在 word representation 处理上会有一些变种.

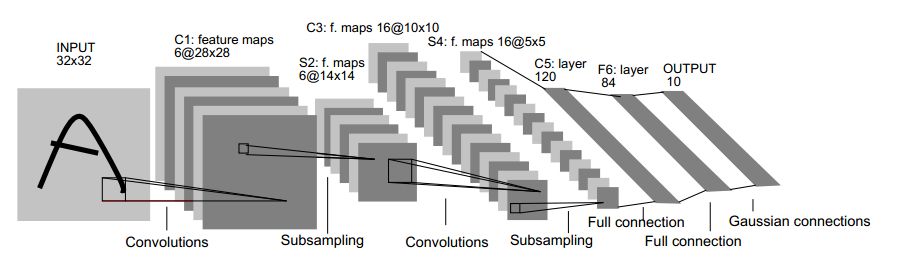
* CNN-rand   
  设计好 embedding\_size 这个 Hyperparameter 后, 对不同单词的向量作随机初始化, 后续BP的时候作调整.
* static   
  拿 pre-trained vectors from word2vec, FastText or GloVe 直接用, 训练过程中不再调整词向量. 这也算是迁移学习的一种思想.
* non-static   
  pre-trained vectors + fine tuning , 即拿word2vec训练好的词向量初始化, 训练过程中再对它们微调.
* multiple channel   
  类比于图像中的RGB通道, 这里也可以用 static 与 non-static 搭两个通道来搞.

一些结果表明，max-pooling 总是优于 average-pooling ，理想的 filter sizes 是重要的，但具体任务具体考量，而用不用正则化似乎在NLP任务中并没有很大的不同。

**8. Text CNN 的tf实现**  
图 8-1 Text CNN 网络中的卷积与池化 结构

需要注意的细节有。   
tf.nn.embedding\_lookup()

**9. 与 LeNet 作比较**

  
figure LeNet-5 网络结构

# LeNet5

conv1\_weights = tf.get\_variable(

"weight",

[CONV1\_SIZE, CONV1\_SIZE, NUM\_CHANNELS, CONV1\_DEEP],

initializer=tf.truncated\_normal\_initializer(stddev=0.1))

tf.nn.conv2d(

input\_tensor,

conv1\_weights,

strides=[1, 1, 1, 1],

padding='SAME')

tf.nn.max\_pool(

relu1,

ksize = [1,POOL1\_SIZE,POOL1\_SIZE,1],

strides=[1,POOL1\_SIZE,POOL1\_SIZE,1],

padding="SAME")

#TextCNN

conv1\_weights = tf.get\_variable(

"weight",

[FILTER\_SIZE, EMBEDDING\_SIZE, 1, NUM\_FILTERS],

initializer=tf.truncated\_normal\_initializer(stddev=0.1))

tf.nn.conv2d(

self.embedded\_chars\_expanded,

conv1\_weights,

strides=[1, 1, 1, 1],

padding="VALID")

tf.nn.max\_pool(

h,

ksize=[1, SEQUENCE\_LENGTH - FILTER\_SIZE + 1, 1, 1],

strides=[1, 1, 1, 1],

padding='VALID')

**先来比较卷积**   
LeNet 的 filter 是**正方形的**, 且每一层都只用了同一种尺寸的卷积核. Text-CNN中, filter 是矩形, 矩形的长度有好几种, 一般取 **(2,3,4),** 而矩形的宽度是定长的, 同 word 的 embedding\_size 相同. 每种尺寸都配有 NUM\_FILTERS个数目, 类比于LeNet中的output\_depth,所以得到的feature\_map是长条状, 宽度为1.   
因为是卷积, 所以stride每个维度都是1.

**再说池化层.**  
池化处理, 也叫下采样. 这里依旧可以对比 LeNet 网络.   
LeNet 的 kernel 是正方形, 一般也是2\*2等, 所以会把卷积后的feature\_map尺寸缩小一半.   
Text-CNN 的 kernel 依旧是长方形, 将整个feature\_map 映射到一个点上. 一步到位, 只有一个池化层.

**全连接层**   
都是多分类, 这一步的处理比较类似. 将池化后的矩阵 reshape为二维, 用 tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits() 计算损失.

**10. TextCNN 论文中的网络结构**

windows size 分别取 (3,4,5), 每个尺寸都会有100个filter.

3.1 Hyperparameters and Training

For all datasets we use: rectified linear units, filter

windows (h) of 3, 4, 5 with 100 feature maps each,

dropout rate (p) of 0.5, l2 constraint (s) of 3, and

mini-batch size of 50. These values were chosen

via a grid search on the SST-2 dev set.

* 1

**参考**

1. [Convolutional Neural Networks for Sentence Classification](https://arxiv.org/pdf/1408.5882.pdf)
2. [Tensorflow版TextCNN主要代码解析](http://blog.csdn.net/u013818406/article/details/69530762)
3. [Recurrent Neural Network for Text Classification with Multi-Task Learning](https://www.ijcai.org/Proceedings/16/Papers/408.pdf)
4. [implementing-a-cnn-for-text-classification-in-tensorflow](http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/)
5. [understanding-convolutional-neural-networks-for-nlp](http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/)
6. [textcnn实现-github](https://github.com/dennybritz/cnn-text-classification-tf/blob/master/text_cnn.py)

https://blog.csdn.net/chuchus/article/details/77847476

<https://blog.csdn.net/u013818406/article/details/69530762>

https://github.com/dennybritz/cnn-text-classification-tf