# 6.深度神经网络

## 6.1.卷积神经网络简介

在 CNN 出现之前，图像对于人工智能来说是一个难题，有2个原因：  
 • 图像需要处理的数据量太大，导致成本很高，效率很低  
 • 图像在数字化的过程中很难保留原有的特征，导致图像处理的准确率不高

需要处理的数据量太大：

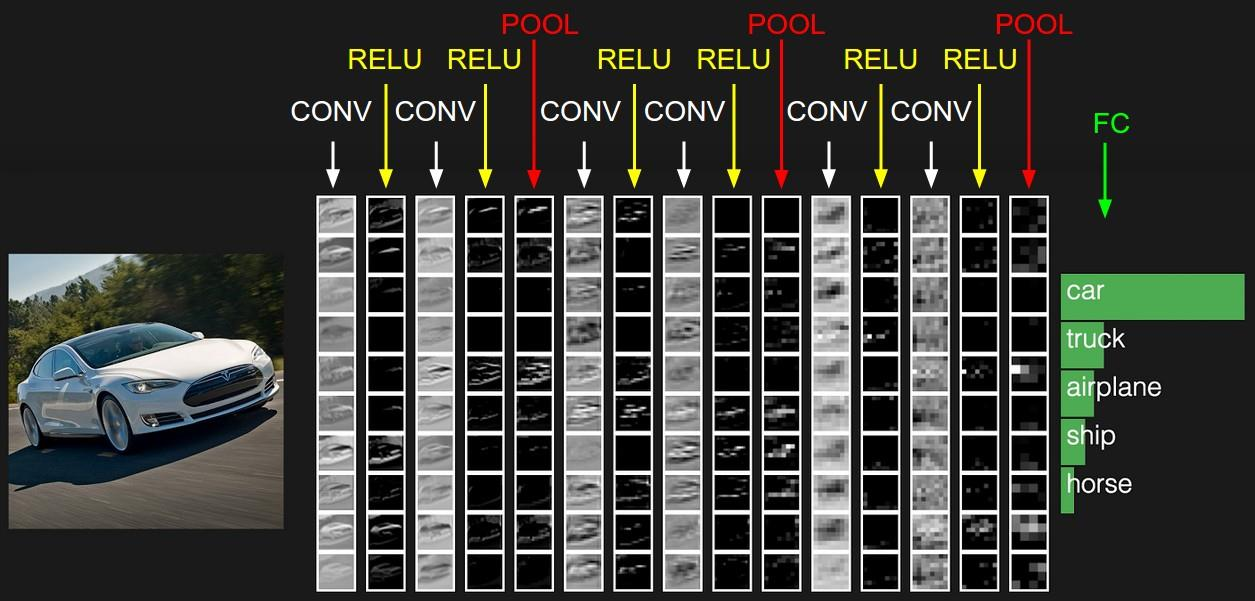
现在随随便便一张图片都是 1000×1000 像素以上的， 每个像素都有RGB 3个参数来表示颜色信息。  
 假如我们处理一张 1000×1000 像素的图片，我们就需要处理3百万个参数！  
 1000×1000×3=3,000,000  
 这么大量的数据处理起来是非常消耗资源的，而且这只是一张不算太大的图片！  
 卷积神经网络 — CNN 解决的第一个问题就是「将复杂问题简化」，把大量参数降维成少量参数，再做处理。  
 更重要的是：我们在大部分场景下，降维并不会影响结果。比如1000像素的图片缩小成200像素，并不影响肉眼认出来图片中是一只猫还是一只狗，机器也是如此。

保留图像特征：

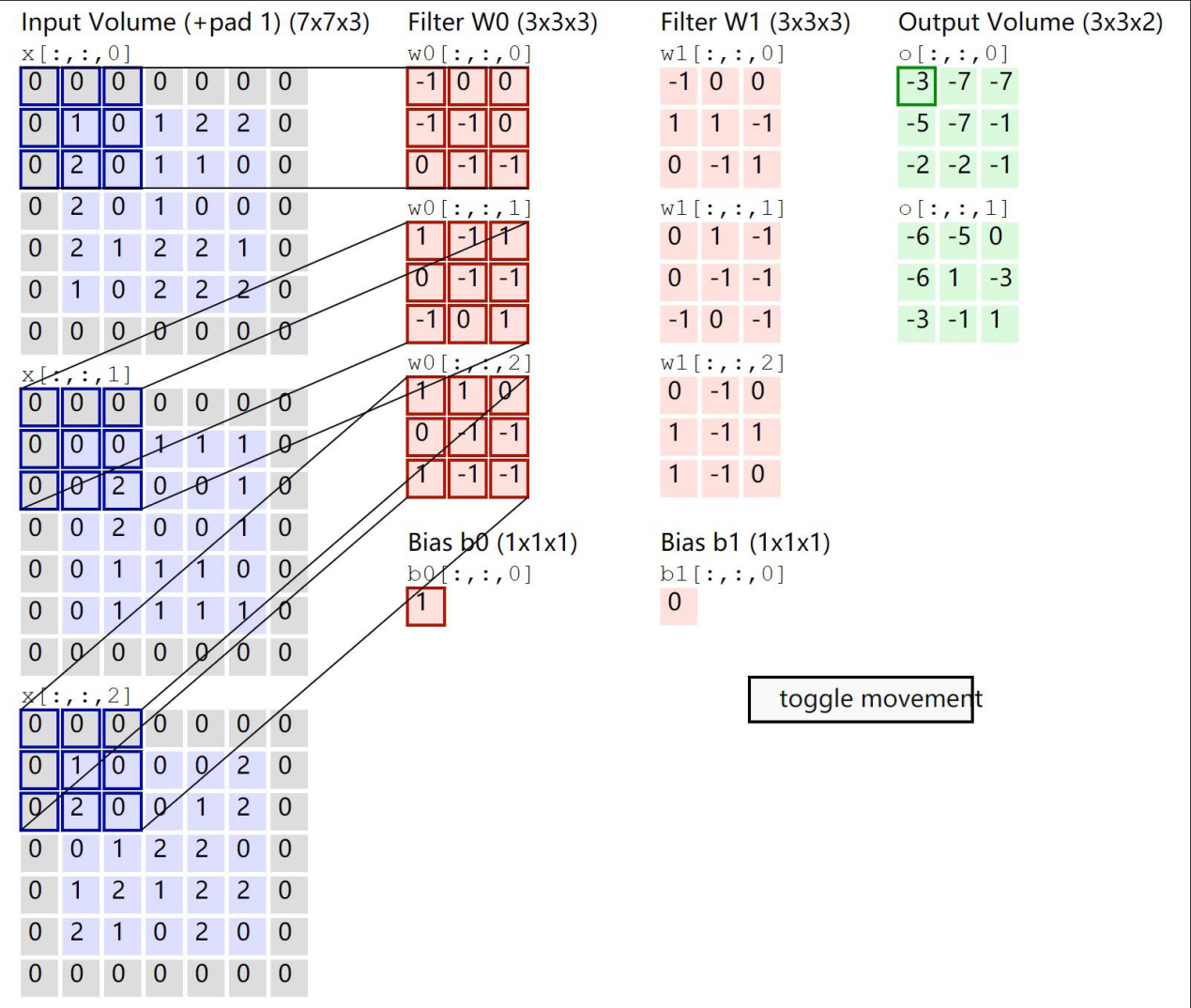
假如有圆形是1，没有圆形是0，那么圆形的位置不同就会产生完全不同的数据表达。但是从视觉的角度来看， 图像的内容（本质）并没有发生变化，只是位置发生了变化。  
 所以当我们移动图像中的物体，用传统的方式得出来的参数会差异很大！这是不符合图像处理的要求的。  
 而 CNN 解决了这个问题，他用类似视觉的方式保留了图像的特征，当图像做翻转，旋转或者变换位置时，它也能有效的识别出来是类似的图像。

CNN 的基本原理：

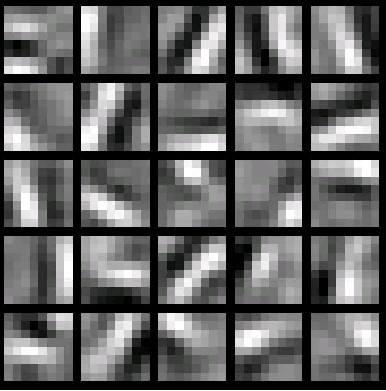
典型的卷积神经网络（CNN） 由3个部分构成：  
 • 卷积层  
 • 池化层  
 • 全连接层  
 如果简单来描述的话：  
 卷积层负责提取图像中的局部特征；池化层用来大幅降低参数量级(降维)；全连接 层类似传统神经网络的部分，用来输出想要的结果。



卷积层的前向传播以及计算方式：



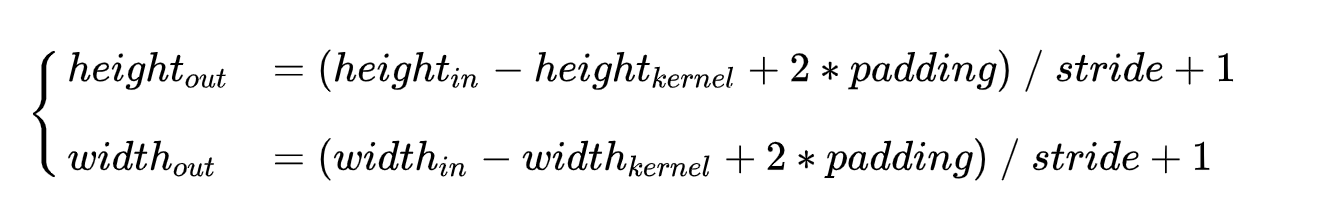
在具体应用中，往往有多个卷积核，可以认为，每个卷积核代表了一种图像模式，如果某个图像块与此卷积核卷积出的值大，则认为此图像块十分接近于此卷积核。如果我们设计了6个卷积核，可以理解为我们认为这个图像上有6种底层纹理模式，也就是我们用6种基础模式就能描绘出一副图像。  
 下图就是25种不同的卷积核的示例：



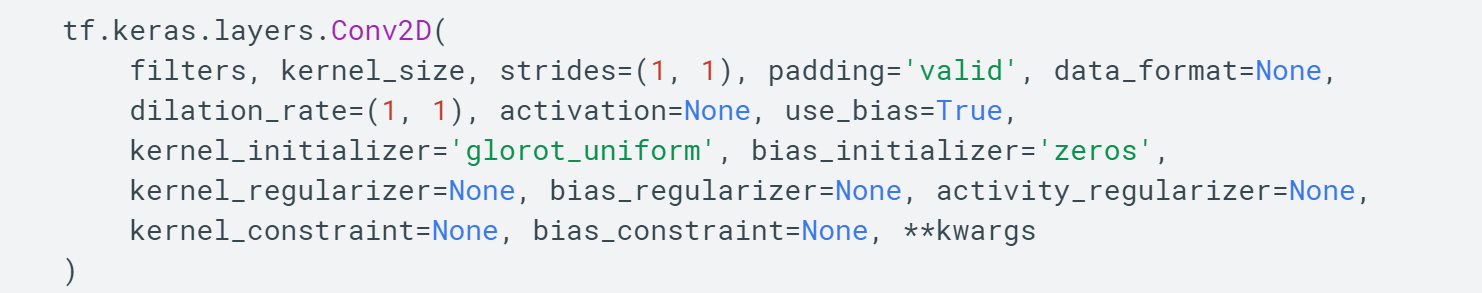
卷积层计算公式：

输入张量格式：四个维度，依次为：样本数、图像高度、图像宽度、图像通道数  
 输出张量格式：与输出矩阵的维度顺序和含义相同，但是后三个维度（图像高度、图像宽度、图像通道数）的尺寸发生变化。  
 卷积核格式：同样是四个维度，但维度的含义与上面两者都不同，为：卷积核高度、卷积核宽度、输入通道数、输出通道数（卷积核个数）

输入张量、 卷积核、 输出张量这三者之间的相互关系：  
 卷积核的输入通道数由输入张量的通道数所决定。  
 输出张量的通道数由卷积核的输出通道数所决定。  
 输出张量的高度和宽度这两个维度的尺寸由输入张量、卷积核、 strides(每一步的 步长)所共同决定。计算公式如下：



tf自带的2D卷积层：



输入：  
 如果data\_format =‘channels\_first‘，则4D张量的形状： (batch\_size, channels, rows, cols) 如果data\_format =’ channels\_last‘，则4D张量的形状： (batch\_size, rows, cols, channels)

输出：  
 如果data\_format =‘channels\_first‘，则4D张量的形状:(batch\_size, filters, new\_rows, new\_cols)

如果data\_format =’ channels\_last‘，则形状的4D张量: (batch\_size, new\_rows, new\_cols, filters) 。

其中rows和cols值可能由于填充而发生了变化。

其他参数：

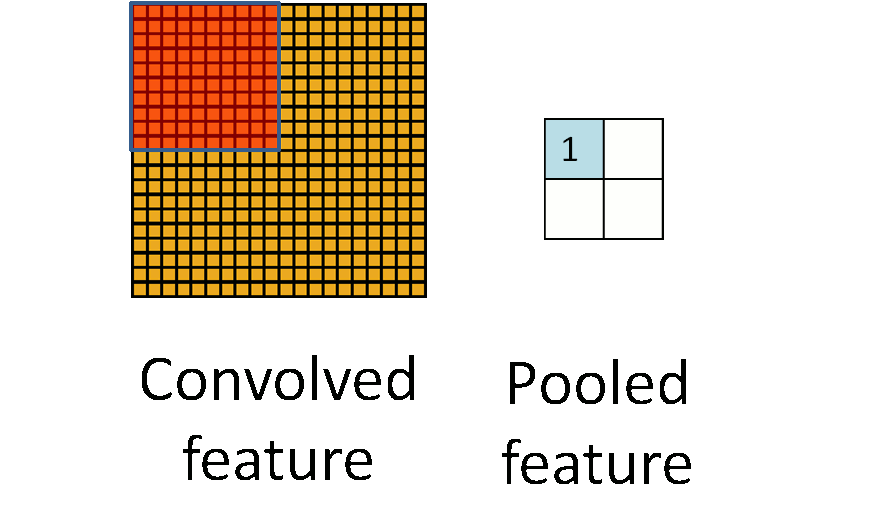
• filters : 输出空间的维数（即卷积中输出滤波器的数量）。  
 • kernel\_size : 过滤器的大小(一个整数或2个整数的元组/列表)。  
 • strides ：横向和纵向的步长。  
 • padding ： valid表示不够卷积核大小的块,则丢弃; same表示不够卷积核大小的块就补0,所以输出和输入形状相同。  
 • data\_format ：一个字符串,表示输入中维度的顺序.支持channels\_last(默认)和channels\_first。 channels\_last对应于具有形状(batch, height, width, channels)的输入,而channels\_first对应于具有形状(batch, channels, height, width)的输入。

• dilation\_rate :表示使用扩张卷积时的扩张率， 注意： strides 不等于1 和dilation\_rate 不等于1 这两种情况不能同时存在。（一般扩张卷积中使用）  
 • activation : 激活函数。  
 • use\_bias ：是否使用偏置。  
 • kernel\_initializer ：卷积核的初始化。  
 • bias\_initializer ：偏差向量的初始化。

• kernel\_regularizer ：卷积核的正则项。  
 • bias\_regularizer ：偏差向量的正则项。  
 • activity\_regularizer ：输出层的正则化函数。  
 • kernel\_constraint ：约束函数应用于卷积核。  
 • bias\_constraint ：将约束函数应用于偏差向量。

**池化层**

池化层简单说就是下采样，他可以大大降低数据的维度。其过程如下：



上图中，我们可以看到，原始图片是20×20的，我们对其进行下采样，采样窗口为10×10，最终将其下采样成为一个2×2大小的特征图。  
 之所以这么做的原因，是因为即使做完了卷积，图像仍然很大（因为卷积核比较小），所以为了降低数据维度，就进行下采样。  
 总结：池化层相比卷积层可以更有效的降低数据维度，这么做不但可以大大减少运算量，还可以有效的避免过拟合。

tf自带的MaxPool2D：



输入形状：

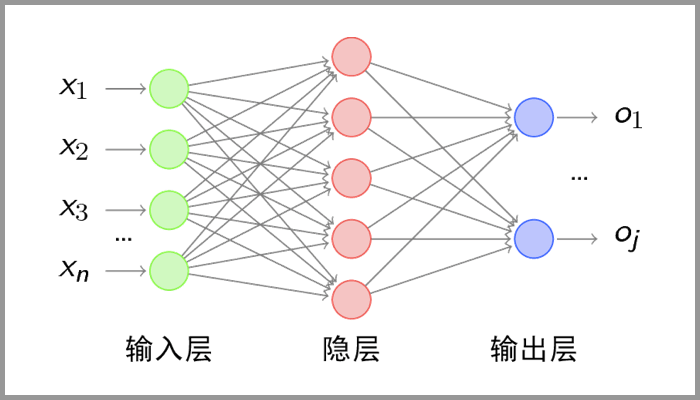
如果data\_format='channels\_last'：具有形状的4D张量(batch\_size, rows,cols, channels)。  
 如果data\_format='channels\_first'：具有形状的4D张量(batch\_size, channels, rows,cols)。  
输出形状：

如果data\_format='channels\_last'：具有形状的4D张量(batch\_size,pooled\_rows, pooled\_cols, channels)。  
 如果data\_format='channels\_first'：具有形状的4D张量(batch\_size, channels,pooled\_rows, pooled\_cols)。

pool\_size： 2个整数的元组/列表： (pool\_height,pool\_width),用于指定池窗口的大小。可以是单个整数，这样所有空间维度具有相同值。  
 strides： 2个整数的元组/列表,用于指定池操作的步幅.可以是单个整数，这样所有空间维度具有相同值。注意：当strides=None，大小和pool\_size一样  
 padding：一个字符串,表示填充方法。  
 data\_format：一个字符串,表示输入中维度的顺序.支持channels\_last(默认)和channels\_first； channels\_last对应于具有形状(batch, height, width, channels)的输入,而channels\_first对应于具有形状(batch, channels, height, width)的输入.

**全连接层**

这个部分就是最后一步了，经过卷积层和池化层处理过的数据输入到全连接层，得到最终想要的结果。  
 经过卷积层和池化层降维过的数据，全连接层才能”跑得动”，不然数据量太大，计算成本高，效率低下。



**常用的CNN架构**

卷积神经网络领域中有几种具有名称的架构。最常见的是：  
 LeNet： Yann LeCun在1990年代开发了卷积网络的第一个成功应用程序。其中，最著名的是LeNet体系结构，该体系结构用于读取邮政编码，数字等。  
 AlexNet： 由Alex Krizhevsky， Ilya Sutskever和Geoff Hinton开发的AlexNet是在计算机视觉中普及卷积网络的第一部作品。 AlexNet 于2012年参加了ImageNet ILSVRC挑战赛，并大大超越了第二名（前5名的错误率为16％，而第二名的错误率为26％）。网络具有与LeNet非常相似的体系结构，但是更深，更大，并且具有彼此堆叠的卷积层（以前通常只有一个CONV层总是紧随其后是POOL层）。

ZF Net： 2013年ILSVRC冠军是Matthew Zeiler和Rob Fergus的卷积网络。它被称为ZFNet（Zeiler＆ Fergus Net的缩写）。通过调整体系结构超参数，特别是通过扩展中间卷积层的大小并减小第一层的步幅和过滤器大小，对AlexNet进行了改进。  
 GoogLeNet： 2014年ILSVRC获奖者是来自Google的Szegedy等人的卷积网络。它的主要贡献是开发了一个Inception模块，该模块大大减少了网络中的参数数量（ 4M，而AlexNet为60M）。此外，本文使用平均池而不是ConvNet顶部的全连接层，从而消除了似乎无关紧要的大量参数。 GoogLeNet也有多个后续版本，最近的是Inception-v4。

VGGNet： 2014年ILSVRC的亚军是来自Karen Simonyan和Andrew Zisserman的网络，该网络被称为VGGNet。它的主要贡献在于表明网络深度是获得良好性能的关键因素。他们最终的最佳网络包含16个CONV / FC层，并且吸引人的是，它具有极其均匀的体系结构，从头到尾仅执行3x3卷积和2x2池化。他们的预训练模型可用于Caffe中的即插即用功能。VGGNet的缺点是评估成本更高，并且使用更多的内存和参数（ 140M）。这些参数中的大多数都位于第一个完全连接的层中，并且由于发现这些FC层可以在不降低性能的情况下被删除，从而大大减少了必要参数的数量。

ResNet： 由Kaiming He等人开发的残差网络。是ILSVRC 2015的获胜者。它具有特殊的跳过连接和大量使用批标准化的功能。该体系结构还缺少网络末端的完全连接的层。读者还可以参考Kaiming的演示文稿（视频，幻灯片），以及一些最近的实验，这些实验在Torch中再现了这些网络。 ResNets目前是最先进的卷积神经网络模型，并且是在实践中使用ConvNets的默认选择（截至2016年5月10日）。特别是，还可以看到更多的最新进展，这些变化将原始架构从何开明等。深度残留网络中的身份映射（ 2016年3月发布）。

**CNN 有哪些实际应用？**

图像分类、检索  
 图像分类是比较基础的应用，他可以节省大量的人工成本，将图像进行有效的分类。对于一些特定领域的图片，分类的准确率可以达到 95%+，已经算是一个可用性很高的应用了。  
 典型场景：图像搜索…

目标定位检测

可以在图像中定位目标，并确定目标的位置及大小。  
 典型场景：自动驾驶、安防、医疗…

目标分割

简单理解就是一个像素级的分类。  
 他可以对前景和背景进行像素级的区分、再高级一点还可以识别出目标并且对目标进行分类。  
 典型场景：美图秀秀、视频后期加工、图像生成…

人脸识别

人脸识别已经是一个非常普及的应用了，在很多领域都有广泛的应用。  
 典型场景：安防、金融、生活…

骨骼识别

骨骼识别是可以识别身体的关键骨骼，以及追踪骨骼的动作。  
 典型场景：安防、电影、图像视频生成、游戏…

**总结**

CNN 的基本原理：  
 • 卷积层 — 主要作用是保留图片的特征  
 • 池化层 — 主要作用是把数据降维，可以有效的避免过拟合  
 • 全连接层 — 根据不同任务输出我们想要的结果

CNN 的价值：  
 • 能够将大数据量的图片有效的降维成小数据量(并不影响结果)  
 • 能够保留图片的特征，类似人类的视觉原理

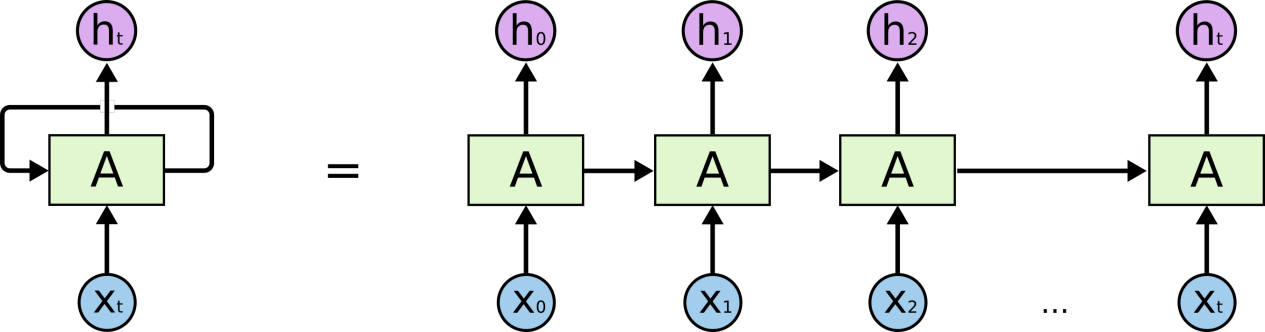
CNN 的实际应用：  
 • 图片分类、检索  
 • 目标定位检测  
 • 目标分割  
 • 人脸识别  
 • 骨骼识别

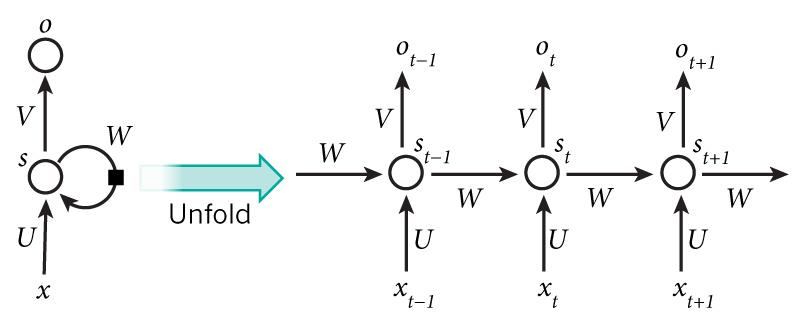
## 6.2.循环神经网络

**什么是RNN?**

在传统的神经网络中，我们假设所有输入（和输出）彼此独立。但是对于许多任务来说，这是一个非常糟糕的主意。如果你想预测句子中的下一个单词，最好知道哪个单词在它之前。 RNN之所以称为循环神经网络， 是因为它们对序列的每个元素执行相同的操作，其输出取决于先前的计算。 因此RNN背后的想法是利用顺序信息。  
 思考RNN的另一种方法是，它们具有“内存”，可以捕获有关到目前为止已计算出的内容的信息。从理论上讲， RNN可以捕捉任意长的顺序信息，但实际上，它们仅能捕捉有限长的顺序信息。

下面就是典型RNN的样子：





**RNN可以做什么？**

RNN已在许多NLP任务中显示出巨大的成功 。在这一点上，我应该提到，最常用的RNN类型是LSTM，它比常规RNN在捕获长期依赖方面要好得多。但是不用担心， LSTM与我们介绍的RNN本质上是同一件事，它们只是具有不同的计算隐藏状态的方式。我们将在后面更详细地介绍LSTM。以下是RNN在NLP中的一些示例应用 。

语言建模和文本生成

给定一个单词序列 ，我们希望在给定先前单词的情况下预测每个单词的概率。 语言模型使我们能够衡量句子的可能性，这是机器翻译的重要输入（因为高概率句子通常是正确的）。 能够预测下一个单词的另外一个作用是我们得到了一个生成模型，该模型使我们能够通过从输出概率中采样来生成新文本。根据我们的训练数据，我们可以生成各种东西。在语言建模中，我们的输入通常是一个单词序列（例如，编码为one-hot向量），而我们的输出是预测单词的序列。

机器翻译

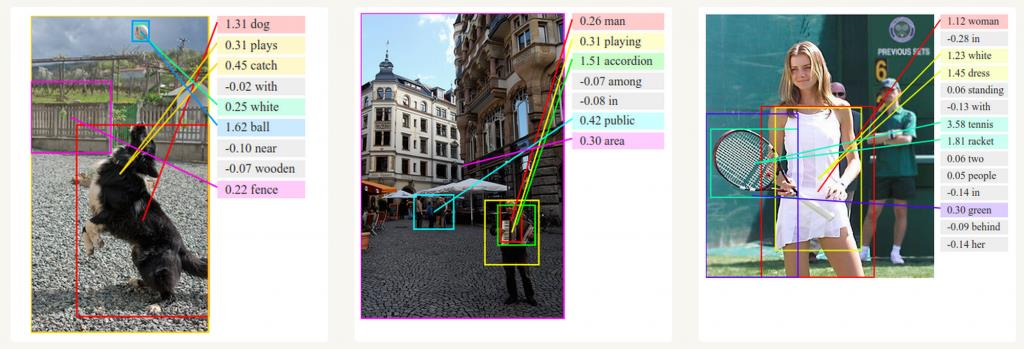
机器翻译类似于语言建模，因为我们的输入是源语言（例如德语 ）中的单词序列。我们希望以目标语言（例如英语）输出一系列单词。一个关键的区别是我们的输出仅在我们看到完整的输入后才开始，因为翻译句子的第一个单词可能需要从完整输入序列中捕获的信息 。

语音识别

给定来自声波的声学信号输入序列，我们可以预测语音段的序列及其概率。

生成图像描述

RNN与卷积神经网络一起被用作模型的一部分，以生成未标记图像的描述。这看起来真是太神奇了。组合模型甚至将生成的单词与图像中找到的特征对齐。



**RNN代码实现**

RNN的API接口

• tf.keras.layers.SimpleRNN  
 • tf.keras.layers.SimpleRNNCell  
 • tf.keras.layers.LSTM  
 • tf.keras.layers.LSTMCell  
 • tf.keras.layers.GRU  
 • tf.keras.layers.GRUCell  
 • tf.keras.layers.RNN

SimpleRNN和SimpleRNNCell的区别

tf.keras.layer.SimpleRNNCell在整个时间序列输入中处理一个步骤，即一个时刻。

tf.keras.layer.SimpleRNN处理整个序列。





• units: 正整数，输出空间的维度。  
 • activation: 要使用的激活函数 (详见 activations)。 默认：双曲正切（ tanh）。 如果  
传入 None，则不使用激活函数 (即 线性激活： a(x) = x)。  
 • use\_bias: 布尔值，该层是否使用偏置向量。  
 • kernel\_initializer: kernel 权值矩阵的初始化器， 用于输入的线性转换。  
 • recurrent\_initializer: recurrent\_kernel 权值矩阵 的初始化器，用于循环层状态的线性转换 。  
 • bias\_initializer:偏置向量的初始化器。

• kernel\_regularizer: 运用到 kernel 权值矩阵的正则化函数recurrent\_regularizer: 运用到 recurrent\_kernel 权值矩阵的正则化函数  
 • bias\_regularizer: 运用到偏置向量的正则化函数 activity\_regularizer: 运用到层输出（它的激活值）的正则化函数  
 • kernel\_constraint: 运用到 kernel 权值矩阵的约束函数  
 • recurrent\_constraint: 运用到 recurrent\_kernel 权值矩阵的约束函数  
 • bias\_constraint: 运用到偏置向量的约束函数  
 • dropout: 在 0 和 1 之间的浮点数。 单元的丢弃比例，用于输入的线性转换。

• recurrent\_dropout: 在 0 和 1 之间的浮点数。 单元的丢弃比例，用于循环层状态的线性转换。  
 • return\_sequences: 布尔值。是返回输出序列中的最后一个输出，还是全部序列。  
 • return\_state: 布尔值。除了输出之外是否返回最后一个状态。  
 • go\_backwards: 布尔值 (默认 False)。 如果为 True，则向后处理输入序列并返回相反的序列。  
 • stateful: 布尔值 (默认 False)。 如果为 True，则批次中索引 i 处的每个样本的最后状态将用作下一批次中索引 i 样本的初始状态。  
 • unroll: 布尔值 (默认 False)。 如果为 True，则网络将展开，否则将使用符号循环。 展开可以加速 RNN，但它往往会占用更多的内存。 展开只适用于短序列。

RNN的API接口

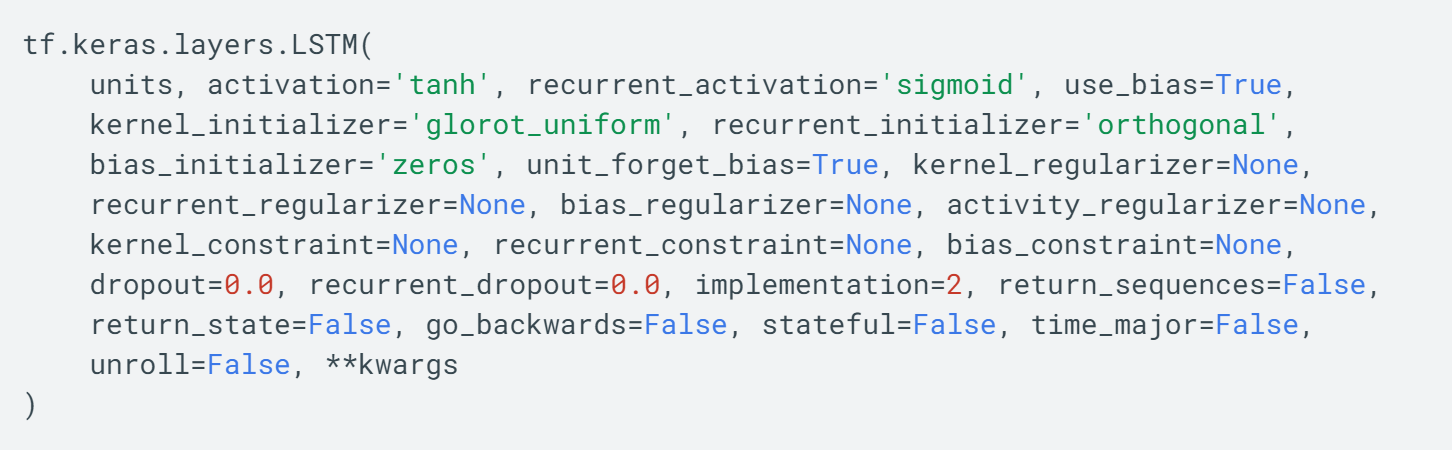


• cell: 一个 RNN 单元实例。 RNN 单元是一个具有以下几项的类：  
 ◼ 一个 call(input\_at\_t, states\_at\_t) 方法， 它返回 (output\_at\_t, states\_at\_t\_plus\_1)。 单元的调用方法也可以引入可选参数 constants。  
 ◼ 一个 state\_size 属性。这可以是单个整数（单个状态）， 在这种情况下，它是 循环层状态的大小（应该与单元输出的大小相同）。 这也可以是整数表示的列表/元组 （每个状态一个大小）。  
 ◼ 一个 output\_size 属性。 这可以是单个整数或者是一个 TensorShape， 它表示 输出的尺寸。出于向后兼容的原因，如果此属性对于当前单元不可用， 则该值将由 state\_size 的第一个元素推断。  
 ◼ cell 也可能是 RNN 单元实例的列表，在这种情况下， RNN 的单元将堆叠在另 一个单元上，实现高效的堆叠 RNN。

• return\_sequences: 布尔值。是返回输出序列中的最后一个输出，还是全部序列。  
 • return\_state: 布尔值。除了输出之外是否返回最后一个状态。  
 • go\_backwards: 布尔值 (默认 False)。 如果为 True，则向后处理输入序列并返回相反的序列。  
 • stateful: 布尔值 (默认 False)。 如果为 True，则批次中索引 i 处的每个样品的最后状态将用作下一批次中索引 i 样品的初始状态。

• unroll: 布尔值 (默认 False)。 如果为 True，则网络将展开，否则将使用符号循环。 展开可以加速 RNN，但它往往会占用更多的内存。 展开只适用于短序列。  
 • input\_dim: 输入的维度（整数）。 将此层用作模型中的第一层时，此参数（或者，关键字参数 input\_shape）是必需的。  
 • input\_length: 输入序列的长度，在恒定时指定。 如果你要在上游连接 Flatten和 Dense 层， 则需要此参数（如果没有它，无法计算全连接输出的尺寸）。 请注意，如果循环神经网络层不是模型中的第一层， 则需要在第一层的层级指定输入长度（例如，通过 input\_shape 参数）。

LSTM的tensorflow接口：



GRU的tensorflow接口：



## 6.3.word2vec

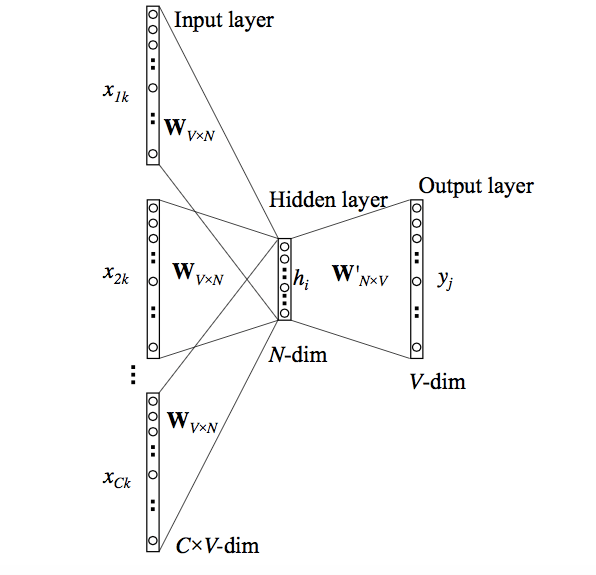
词向量：

One-hot 表示 ：维度灾难、语义鸿沟；  
 分布式表示 (distributed representation) ：  
 • 矩阵分解（LSA）：利用全局语料特征，但SVD求解计算复杂度大；  
 • 基于NNLM/RNNLM的词向量：词向量为副产物，存在效率不高等问题；  
 • word2vec、 fastText：优化效率高，但是基于局部语料；  
 • glove：基于全局预料，结合了LSA和word2vec的优点；  
 • elmo、 GPT、 bert：动态特征；

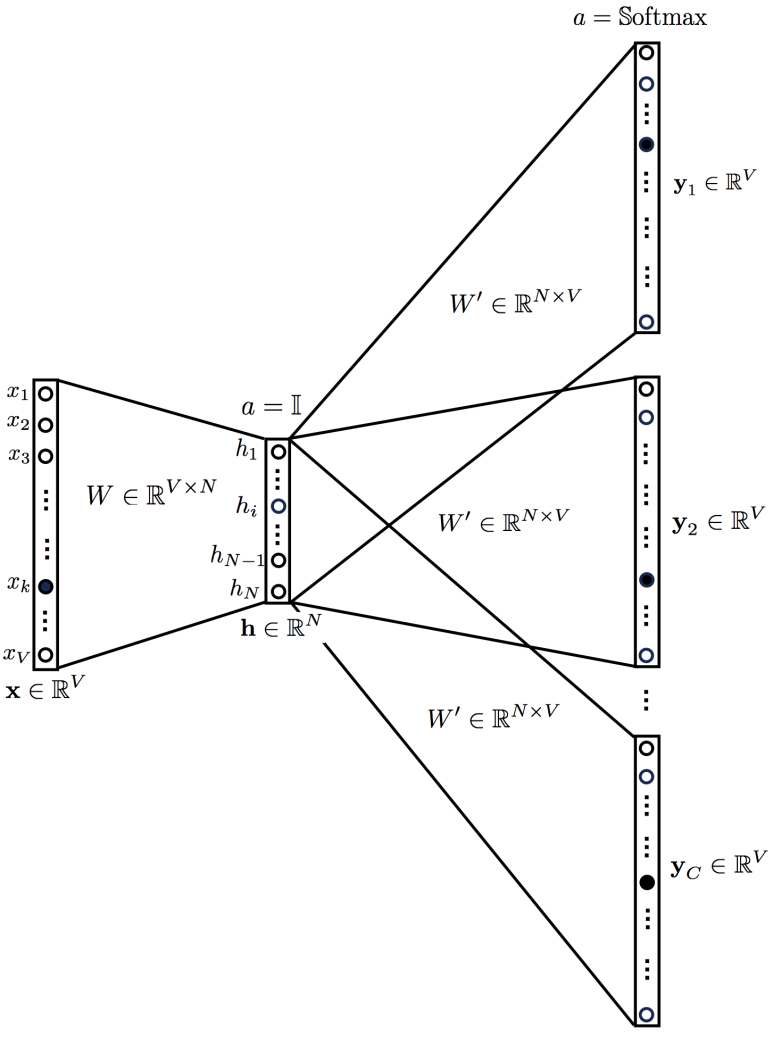
Word2vec简介：

通俗的来讲， word2vec就是把 x 看做一个句子里的一个词语， y 是这个词语的上下文词语，那么这里的 f，便是 NLP 中经常出现的『语言模型』 （language model），这个模型的目的，就是判断 (x,y) 这个样本，是否符合自然语言的法则，更通俗点说就是：词语x和词语y放在一起，是不是人话。  
 • 两个算法： CBOW 和 skip-gram  
 • 两种训练方法：负采样和层级softmax

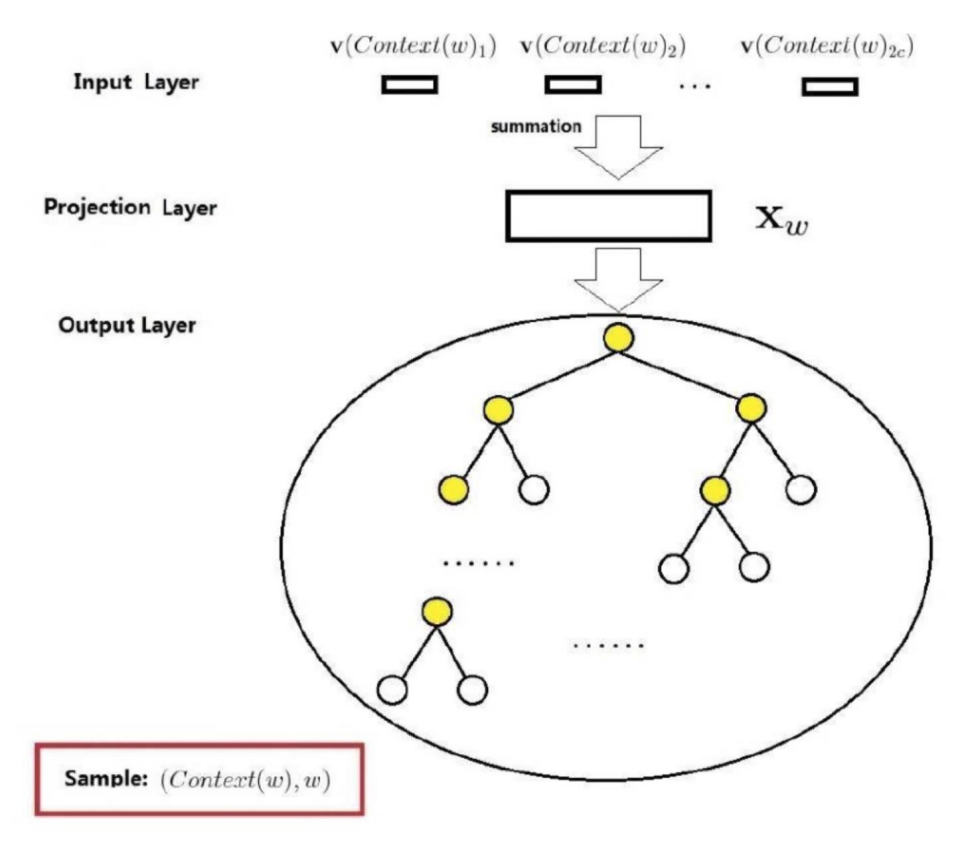
CBOW



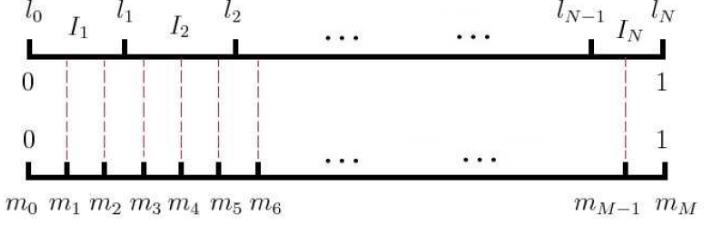
skip-gram



层级softmax(Hierarchical Softmax)的网络结构图



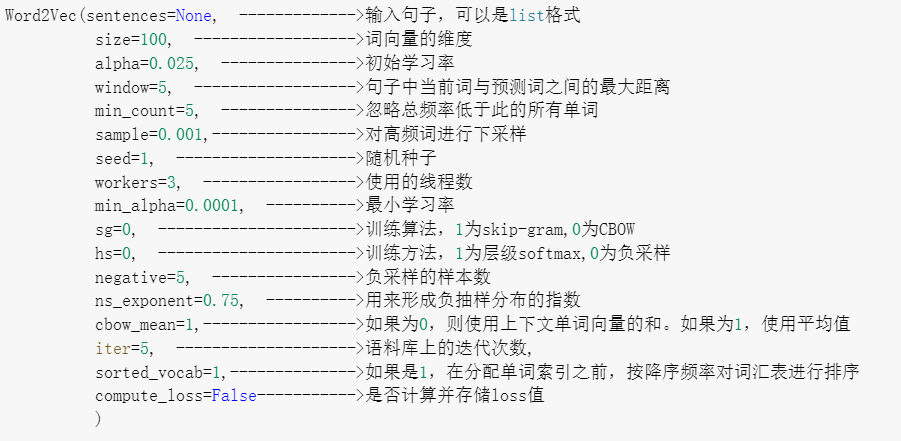
负采样的网络结构图词汇表大小为N将其分为M等份M>>N



两者哪个更好呢？  
 • Skip-gram可以很好的处理少量数据，并且可以很好的表示稀疏单词。  
 • CBOW速度更快，对于更频繁的单词具有更好的表示。  
训练方法哪个更好呢？  
 • 层级softmax 对高频词效果更好；  
 • 对应的负采样对低频词效果更好，向量维度较低时效果更好。

gensim实现Word2vec

!pip install --upgrade gensim  
 from gensim.test.utils import common\_texts, get\_tmpfile  
 from gensim.models import Word2Vec  
 path = get\_tmpfile("word2vec.model")  
 model = Word2Vec(common\_texts, size=100, window=5, min\_c  
 ount=1, workers=4)  
 model.save("word2vec.model")  
 model = Word2Vec.load("word2vec.model")  
 vector = model.wv['computer']



案例：

*#导入包*

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**from** **gensim.models** **import** Word2Vec

*#读取数据集*

train = pd.read\_csv('./cnews/train.tsv',sep='**\t**',header=**None**,names=['label','content'])

val = pd.read\_csv('./cnews/dev.tsv',sep='**\t**',header=**None**,names=['label','content'])

test = pd.read\_csv('./cnews/test.tsv',sep='**\t**',header=**None**,names=['label','content'])

train.head()

|  | **label** | **content** |
| --- | --- | --- |
| **0** | 体育 | 马晓旭意外受伤让国奥警惕 无奈大雨格外青睐殷家军记者傅亚雨沈阳报道 来到沈阳，国奥队依然没有... |
| **1** | 体育 | 商瑞华首战复仇心切 中国玫瑰要用美国方式攻克瑞典多曼来了，瑞典来了，商瑞华首战求3分的信心也... |
| **2** | 体育 | 冠军球队迎新欢乐派对 黄旭获大奖张军赢下PK赛新浪体育讯12月27日晚，“冠军高尔夫球队迎新... |
| **3** | 体育 | 辽足签约危机引注册难关 高层威逼利诱合同笑里藏刀新浪体育讯2月24日，辽足爆发了集体拒签风波... |
| **4** | 体育 | 揭秘谢亚龙被带走：总局电话骗局 复制南杨轨迹体坛周报特约记者张锐北京报道  谢亚龙已经被公安... |

利用jieba分词

**import** **jieba**

**def** content\_cut(x):

x = jieba.lcut(x)

x = " ".join(x)

**return** x

train['content'] = train['content'].map(**lambda** x: content\_cut(x))

val['content'] = val['content'].map(**lambda** x: content\_cut(x))

test['content'] = test['content'].map(**lambda** x: content\_cut(x))

df = pd.concat([train,val,test],axis=0)

df.head()

|  | **label** | **content** |
| --- | --- | --- |
| **0** | 体育 | 马晓旭 意外 受伤 让 国奥 警惕 无奈 大雨 格外 青睐 殷家 军 记者 傅亚雨 沈阳... |
| **1** | 体育 | 商瑞华 首战 复仇 心切 中国 玫瑰 要 用 美国 方式 攻克 瑞典 多曼来 了 ， 瑞... |
| **2** | 体育 | 冠军 球队 迎新 欢乐 派对 黄旭获 大奖 张军 赢 下 PK 赛 新浪 体育讯 12 ... |
| **3** | 体育 | 辽足 签约 危机 引 注册 难关 高层 威逼利诱 合同 笑里藏刀 新浪 体育讯 2 月 ... |
| **4** | 体育 | 揭秘 谢亚龙 被 带走 ： 总局 电话 骗局 复制 南杨 轨迹 体坛周报 特约记者 张锐... |

训练word2vec

sentences = [document.split(' ') **for** document **in** df['content'].values]

model = Word2Vec(sentences=sentences,

size=200,*#维度*

alpha=0.025, *#默认*

window=5, *#默认*

min\_count=2,*#2，3*

sample=0.001,*#*

seed=2018, *#*

workers=11, *#线程*

min\_alpha=0.0001,

sg=0, *#cbow*

hs=0, *#负采样*

negative=5,*#负采样个数*

ns\_exponent=0.75,

cbow\_mean=1,*#求和再取平均*

iter=10, *#10到20*

compute\_loss =**True**

)

*## 保存word2vec*

model.save("./word2vec/word2vec\_word\_200")

model = Word2Vec.load("./word2vec/word2vec\_word\_200")

*#查看单词的词汇*

model.wv.vocab

{'马晓旭': <gensim.models.keyedvectors.Vocab at 0x28f38610a90>,

'意外': <gensim.models.keyedvectors.Vocab at 0x28e7e56a198>,

...

'往年': <gensim.models.keyedvectors.Vocab at 0x28e7e923a58>,

...}

*#查看单词的向量*

model.wv['受伤'].shape

(200,)

*#和这个单词最相似的单词*

model.most\_similar("受伤",topn=20)

[('扭伤', 0.7356216907501221),

('伤势', 0.699065089225769),

...

('肿胀', 0.5933536887168884)]

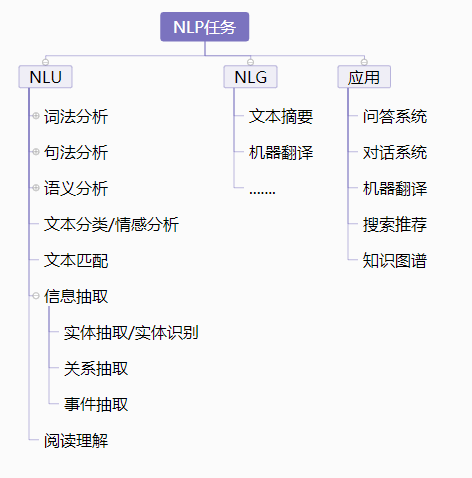
*#计算两个单词之间相似性*

model.wv.similarity("期盼","期待")

0.75351274

## 6.4.LSTM实现新闻分类算法

NLP任务：



背景介绍:

THUCNews是根据新浪新闻RSS订阅频道2005~2011年间的历史数据筛选过滤生  
成，包含74万篇新闻文档（2.19 GB），均为UTF-8纯文本格式。我们在原始新浪  
新闻分类体系的基础上，重新整合划分出14个候选分类类别：财经、彩票、房产、  
股票、家居、教育、科技、社会、时尚、时政、体育、星座、游戏、娱乐。

任务参考： http://thuctc.thunlp.org/  
 http://thuctc.thunlp.org/message

数据案例：

家居

...

房产

科技

游戏

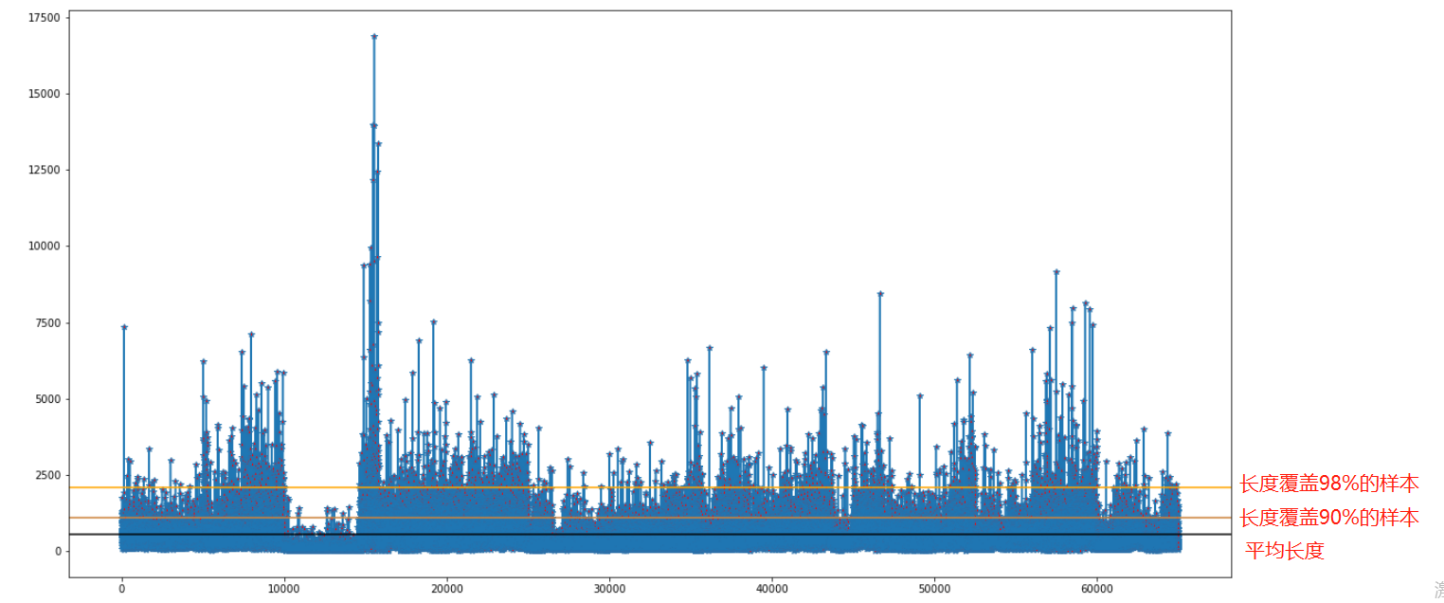
娱乐

体育

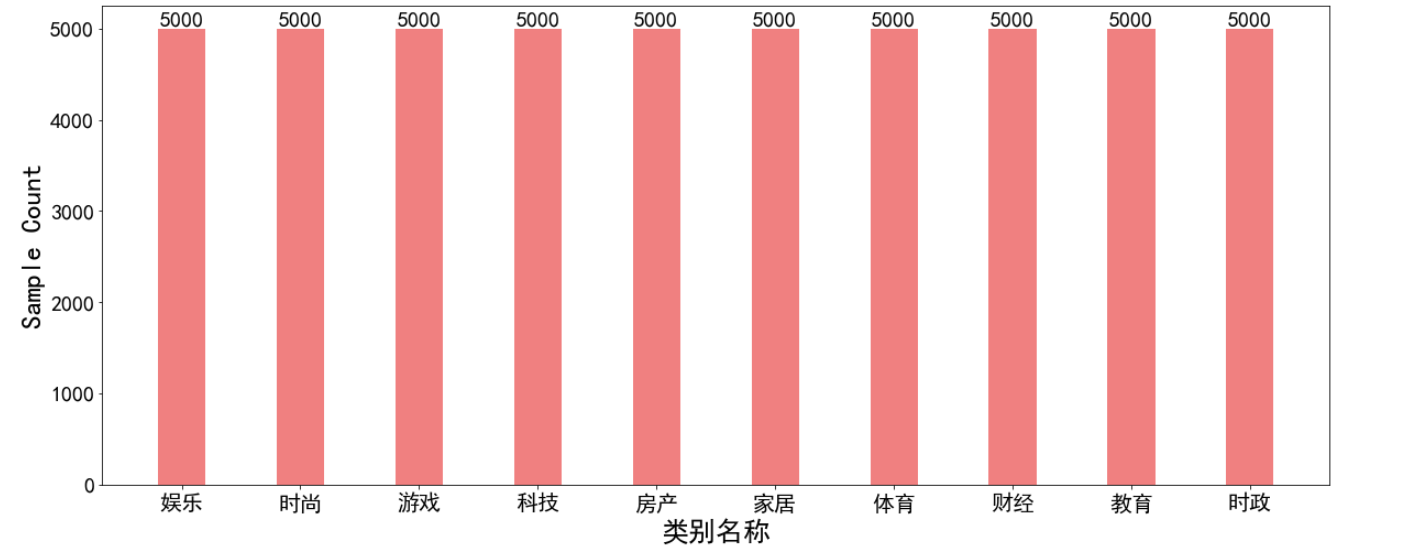
马晓旭意外受伤让国奥警惕 无奈大雨  
格外青睐殷家军记者傅亚雨沈阳报道  
来到沈阳，国奥队依然没有摆脱雨水的  
困扰。 7月31日下午6点，国奥队的日  
常训练再度受到大雨的干扰，无奈之下  
队员们只慢跑了25分钟就草草收场。  
31日上午10点，国奥队在奥体中心外  
场训练的时候，天就是阴沉沉的，气象  
预报显示当天下午沈阳就有大雨，但幸  
好队伍上午的训练并没有受到任何干扰。  
下午6点，当球队抵达训练场时，大雨  
已经下了几个小时，而且丝毫没有停下  
来的意思。抱着试一试的态度，球队开  
始了当天下午的例行训练， 25分钟过  
去了，天气没有任何转好的迹象，为了  
保护球员们，国奥队决定中止当天的训  
练，全队立即返回酒店。在雨中训练对  
足球队来说并不是什么稀罕事……

**数据分析与处理**

文本长度统计



label统计



文本截断、补全

文本截断：前截断、后截断

文本补全：前补全、后补全

Tensorflow.keras接口提供一套文本编码、处理的工具。

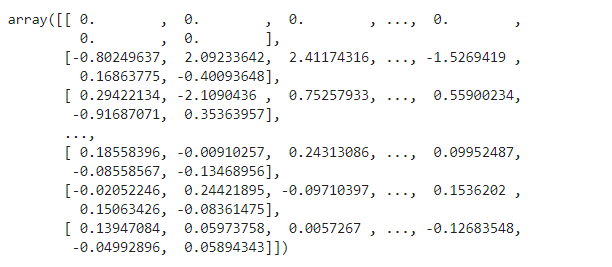
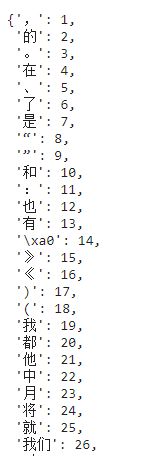


• num\_words: 需要保留的最大词数，基于词频。只有最常出现的 num\_words 词会被保留。  
 • filters: 一个字符串，其中每个元素是一个将从文本中过滤掉的字符。默认值是所有标点符号，加上制表符和换行符，减去 ' 字符。  
 • lower: 布尔值。是否将文本转换为小写。  
 • split: 字符串。按该字符串切割文本。  
 • char\_level: 如果为 True，则每个字符都将被视为标记。  
 • oov\_token: 如果给出，它将被添加到 word\_index 中，并用于在 text\_to\_sequence 调用期间替换词汇表外的单词。

**模型建模流程：**

数据处理->词向量训练->Embedding构建->构建dataset->模型建模方法->模型结构设置->模型效果评估->模型优化、方法优化

Word Embedding构建:

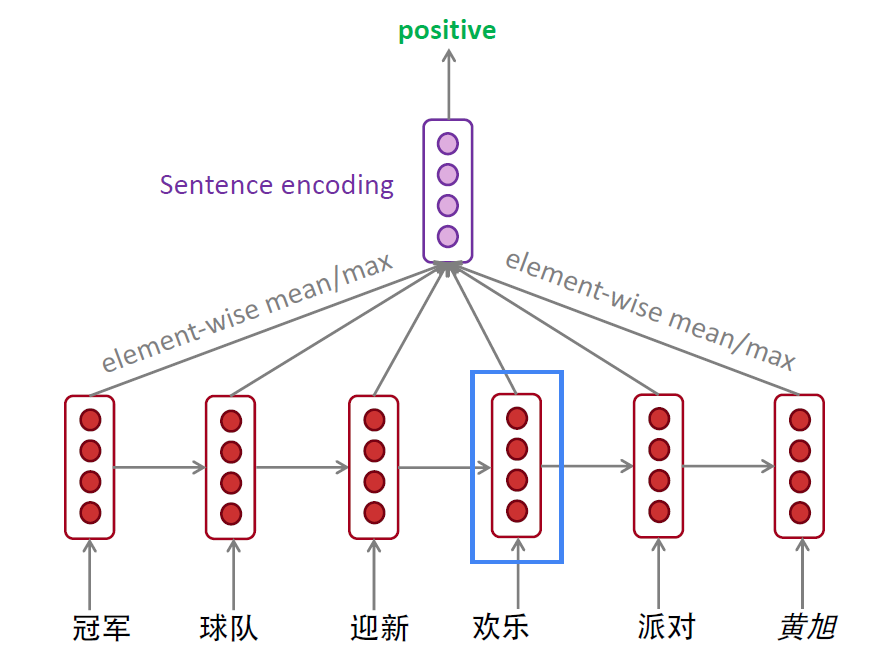


模型框架：

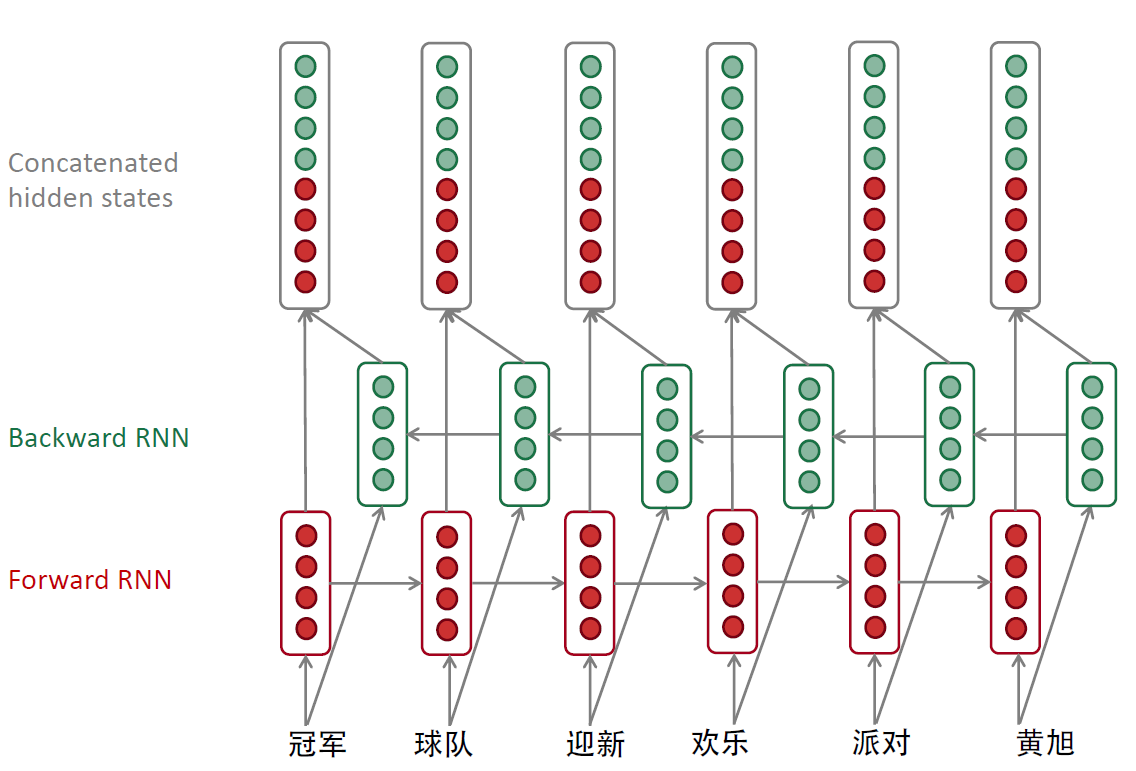
数据输入->Embedding->编码层->池化层->全连接层->输出层

编程层：Bidirectional RNNs (GRU、 LSTM)

单层RNN



双向RNN

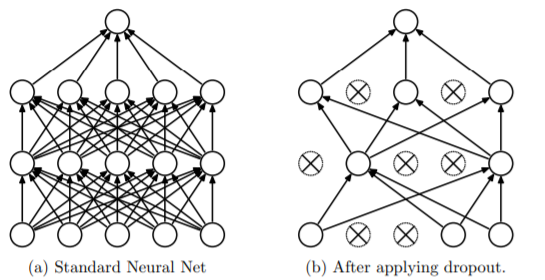


池化层：

• 最大池化 tf.keras.layers.GlobalMaxPool1D  
 • 平均池化 tf.keras.layers.GlobalAveragePooling1D  
 • Attention池化

Dropout：

作用：减少过拟合的风险

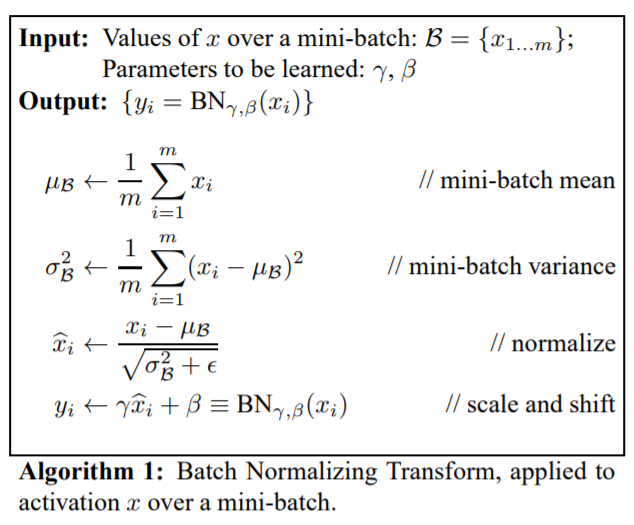


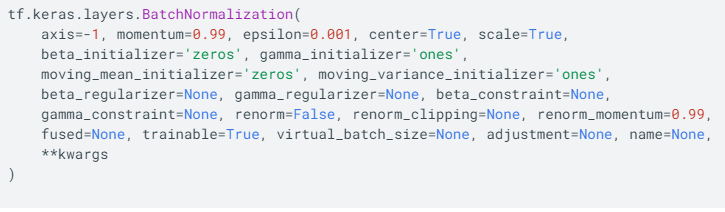


rate：在 0 和 1 之间浮动。需要丢弃的输入比例。  
 noise\_shape: 1D 整数张量， 表示将与输入相乘的二进制 dropout 掩层的形状。 例如，如果你的输入尺寸为 (batch\_size, timesteps, features)，然后 你希望 dropout 掩层在所有时间步都是一样的， 你可以使用 noise\_shape=(batch\_size, 1, features)。  
 seed: 一个作为随机种子的 Python 整数。

Batch Normalization：

• 提升了训练速度，收敛过程大大加快  
 • 增加分类效果





• axis: 整数，需要标准化的轴 （通常是特征轴）。 例如，在 data\_format="channels\_first"的 Conv2D 层之后， 在 BatchNormalization 中设置 axis=1。  
 • momentum: 移动均值和移动方差的动量。  
 • epsilon: 增加到方差的小的浮点数，以避免除以零。  
 • center: 如果为 True，把 beta 的偏移量加到标准化的张量上。 如果为 False， beta 被忽略。

• scale: 如果为 True，乘以 gamma。 如果为 False， gamma 不使用。 当下一层为线性层（或者例如 nn.relu）， 这可以被禁用，因为缩放将由下一层完成。  
 • beta\_initializer: beta 权重的初始化方法。  
 • gamma\_initializer: gamma 权重的初始化方法。  
 • moving\_mean\_initializer: 移动均值的初始化方法。  
 • moving\_variance\_initializer: 移动方差的初始化方法。  
 • beta\_regularizer: 可选的 beta 权重的正则化方法。  
 • gamma\_regularizer: 可选的 gamma 权重的正则化方法。  
 • beta\_constraint: 可选的 beta 权重的约束方法。  
 • gamma\_constraint: 可选的 gamma 权重的约束方法。

**模型训练**

• Early Stopping  
 • 学习率衰减  
 • 保存最优权重

代码实现：

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**from** **gensim.models** **import** Word2Vec

**import** **tensorflow** **as** **tf**

*#读取数据集*

train = pd.read\_csv('./cnews/train.tsv',sep='**\t**',header=**None**,names=['label','content'])

val = pd.read\_csv('./cnews/dev.tsv',sep='**\t**',header=**None**,names=['label','content'])

test = pd.read\_csv('./cnews/test.tsv',sep='**\t**',header=**None**,names=['label','content'])

train.shape

(50000, 2)

val.shape

(5000, 2)

test.shape

(10000, 2)

train['content'][0]

'马晓旭意外受伤让国奥警惕 无奈大雨格外青睐殷家军记者傅亚雨沈阳报道 来到沈阳，国奥队依然没有摆脱雨水的困扰。7月31日下午6点，国奥队的日常训练再度受到大雨的干扰，无奈之下队员们只慢跑了25分钟就草草收场。31日

...

的“青睐”有些不解。'

**import** **jieba**

**def** content\_cut(x):

x = jieba.lcut(x)

x = " ".join(x)

**return** x

train['content'] = train['content'].map(**lambda** x: content\_cut(x))

val['content'] = val['content'].map(**lambda** x: content\_cut(x))

test['content'] = test['content'].map(**lambda** x: content\_cut(x))

df = pd.concat([train,val,test],axis=0)

df['content\_len'] = df['content'].map(**lambda** x:len(x.split(" ")))

np.percentile(df['content\_len'].values,80) *# 覆盖80%的新闻长度*

768.0

*#数据分析*

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

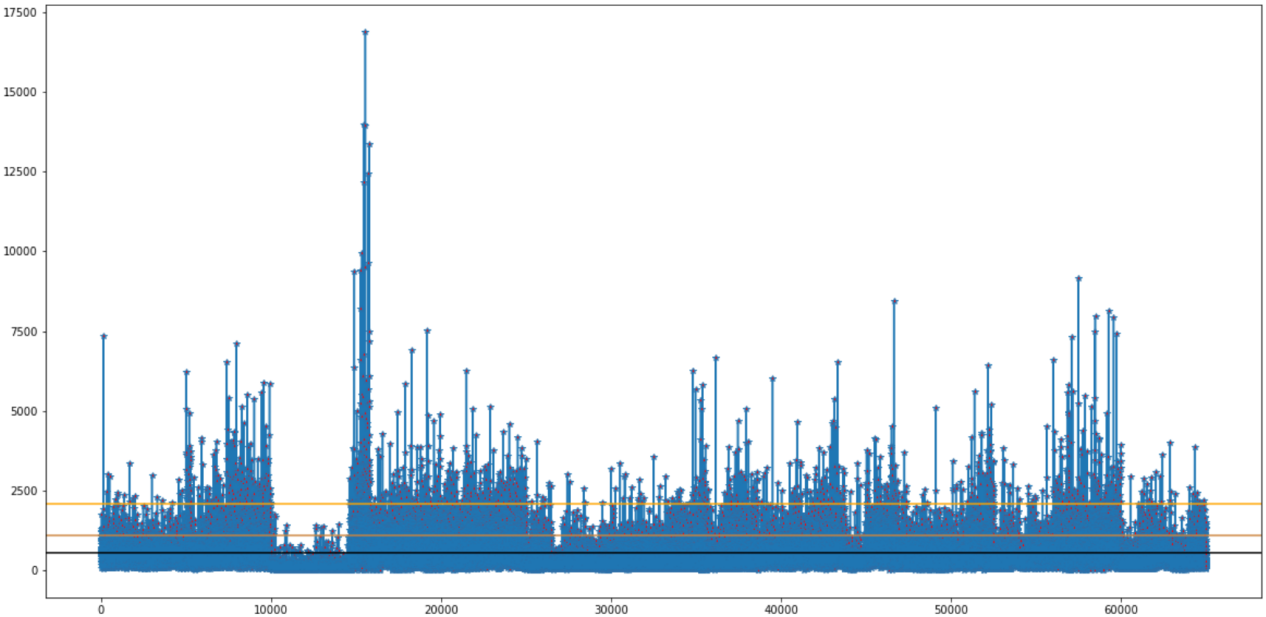
plt.figure(figsize=(20,10))

plt.plot(df['content\_len'].tolist(),marker='\*',markerfacecolor='red')

plt.axhline(y=np.mean(df['content\_len'].tolist()),color="black",)

plt.axhline(y=np.percentile(df['content\_len'].values,90),color="peru")

plt.axhline(y=np.percentile(df['content\_len'].values,98),color="orange")



**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

plt.rcParams['font.family'] = ['sans-serif']

plt.rcParams['font.sans-serif'] = ['SimHei']

count\_class = train['label'].value\_counts()

plt.figure(figsize=(20,8))

class\_bar = plt.bar(x=count\_class.index, height=count\_class.tolist(),width=0.4,color='lightcoral')

plt.xticks(fontsize=20)

plt.yticks(fontsize=20)

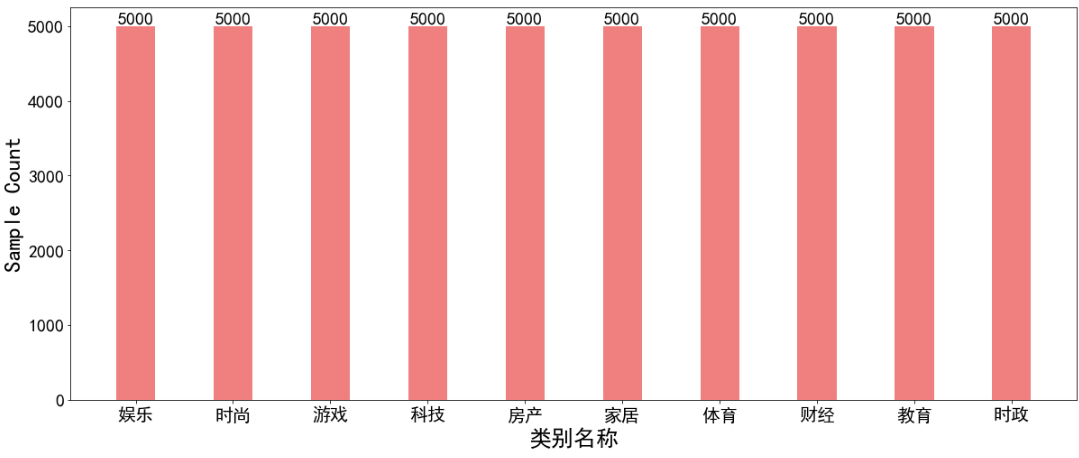
**for** bar **in** class\_bar:

height = bar.get\_height()

plt.text(bar.get\_x() + bar.get\_width() / 2, height+1, str(height), ha="center", va="bottom",fontsize=20)

plt.ylabel("Sample Count",fontsize=25)

plt.xlabel("类别名称",fontsize=25)



*# 训练词向量*

**import** **os**

file\_name = './embedding/Word2Vec\_word\_200.model'

**if** **not** os.path.exists(file\_name):

model = Word2Vec([document.split(' ')**for** document **in** df['content'].values],

size=200,

window=5,

iter=10,

workers=11,

seed=2018,

min\_count=2)

model.save(file\_name)

**else**:

model = Word2Vec.load(file\_name)

print("add word2vec finished....")

add word2vec finished....

tokenizer = tf.keras.preprocessing.text.Tokenizer(num\_words=50000,

lower=**False**,filters="")

tokenizer.fit\_on\_texts(df['content'].tolist())

train\_ = tokenizer.texts\_to\_sequences(train['content'].values)

val\_ = tokenizer.texts\_to\_sequences(val['content'].values)

test\_ = tokenizer.texts\_to\_sequences(test['content'].values)

*#Word Embedding*

word\_vocab = tokenizer.word\_index

count = 0

embedding\_matrix = np.zeros((len(word\_vocab) + 1, 200))

**for** word, i **in** word\_vocab.items():

embedding\_vector = model.wv[word] **if** word **in** model **else** **None**

**if** embedding\_vector **is** **not** **None**:

count += 1

embedding\_matrix[i] = embedding\_vector

**else**:

unk\_vec = np.random.random(200) \* 0.5

unk\_vec = unk\_vec - unk\_vec.mean()

embedding\_matrix[i] = unk\_vec

*#label 编码*

**from** **sklearn.preprocessing** **import** LabelEncoder

**from** **tensorflow.keras.utils** **import** to\_categorical

lb = LabelEncoder()

train\_label = lb.fit\_transform(train['label'].values)

val\_label = lb.transform(val['label'].values)

test\_label = lb.transform(test['label'].values)

*#建模*

content = tf.keras.layers.Input(shape=(800), dtype='int32')

embedding = tf.keras.layers.Embedding(

name="word\_embedding",

input\_dim=embedding\_matrix.shape[0],

weights=[embedding\_matrix],

output\_dim=embedding\_matrix.shape[1],

trainable=**False**)

x = tf.keras.layers.SpatialDropout1D(0.2)(embedding(content))

*#编码层*

*#bi-GRU*

x = tf.keras.layers.Bidirectional(tf.keras.layers.GRU(200, return\_sequences=**True**))(x) *#（batch,800,400）*

x = tf.keras.layers.Bidirectional(tf.keras.layers.GRU(200, return\_sequences=**True**))(x)

*#池化层*

avg\_pool = tf.keras.layers.GlobalAveragePooling1D()(x) *#（batch,400）*

max\_pool = tf.keras.layers.GlobalMaxPooling1D()(x)*#(batch,400)*

conc = tf.keras.layers.concatenate([avg\_pool, max\_pool])

x = tf.keras.layers.Dense(1000)(conc)

x = tf.keras.layers.BatchNormalization()(x)

x = tf.keras.layers.Activation(activation="relu")(x)

x = tf.keras.layers.Dropout(0.2)(x)

x = tf.keras.layers.Dense(500)(x)

x = tf.keras.layers.BatchNormalization()(x)

x = tf.keras.layers.Activation(activation="relu")(x)

x = tf.keras.layers.Dense(10)(x)

output = tf.nn.softmax(x)

model = tf.keras.models.Model(inputs=content, outputs=output)

*# label进行one-hot编码*

train\_label = tf.keras.utils.to\_categorical(train\_label,num\_classes=10,dtype='int')

val\_label = tf.keras.utils.to\_categorical(val\_label,num\_classes=10,dtype='int')

test\_label = tf.keras.utils.to\_categorical(test\_label,num\_classes=10,dtype='int')

train\_label.shape

(50000, 10)

train\_dataset = tf.data.Dataset.from\_tensor\_slices((train\_,train\_label))

train\_label

array([[1, 0, 0, ..., 0, 0, 0],

[1, 0, 0, ..., 0, 0, 0],

[1, 0, 0, ..., 0, 0, 0],

...,

[0, 0, 0, ..., 0, 0, 1],

[0, 0, 0, ..., 0, 0, 1],

[0, 0, 0, ..., 0, 0, 1]])

train\_dataset = train\_dataset.prefetch(buffer\_size = tf.data.experimental.AUTOTUNE)

train\_dataset = train\_dataset.shuffle(buffer\_size = 23000)

train\_dataset = train\_dataset.batch(batch\_size=128) *# 32*

val\_dataset = tf.data.Dataset.from\_tensor\_slices((val\_,val\_label))

val\_dataset = val\_dataset.prefetch(tf.data.experimental.AUTOTUNE)

val\_dataset = val\_dataset.shuffle(buffer\_size=23000)

val\_dataset = val\_dataset.batch(batch\_size=256) *#64*

**for** a,b **in** train\_dataset.take(1):

print(a.shape,b.shape)

(128, 800) (128, 10)

learning\_rate = 0.001

loss\_object = tf.keras.losses.CategoricalCrossentropy()

optimizer = tf.keras.optimizers.Adam(learning\_rate=learning\_rate)

train\_loss = tf.keras.metrics.Mean(name='train\_loss')

train\_accuracy = tf.keras.metrics.CategoricalAccuracy(name='train\_accuracy')

test\_loss = tf.keras.metrics.Mean(name='test\_loss')

test\_accuracy = tf.keras.metrics.CategoricalAccuracy(name='test\_accuracy')

*#mini-batch*

**def** train\_one\_step(contents, labels):

**with** tf.GradientTape() **as** tape:

predictions = model(contents)

loss = loss\_object(labels, predictions)

gradients = tape.gradient(loss, model.trainable\_variables)

optimizer.apply\_gradients(zip(gradients, model.trainable\_variables))

train\_loss(loss) *#update*

train\_accuracy(labels, predictions)*#update*

**def** test\_one\_step(contents, labels):

predictions = model(contents)

t\_loss = loss\_object(labels, predictions)

test\_loss(t\_loss)

test\_accuracy(labels, predictions)

EPOCHS=10

**for** epoch **in** range(EPOCHS):

*# 在下一个epoch开始时，重置评估指标*

train\_loss.reset\_states()

train\_accuracy.reset\_states()

test\_loss.reset\_states()

test\_accuracy.reset\_states()

**for** content, labels **in** train\_dataset:

train\_one\_step(content, labels) *#mini-batch 更新*

**for** val\_content, val\_labels **in** val\_dataset:

test\_one\_step(val\_content, val\_labels)

template = 'Epoch **{}**, Loss: **{}**, Accuracy: **{}**, Test Loss: **{}**, Test Accuracy: **{}**'

print(template.format(epoch + 1,

train\_loss.result(),

train\_accuracy.result() \* 100,

test\_loss.result(),

test\_accuracy.result() \* 100

))

Epoch 1, Loss: 0.20004434883594513, Accuracy: 94.01599884033203, Test Loss: 0.10956119000911713, Test Accuracy: 97.15999603271484

...

Epoch 10, Loss: 0.009419434703886509, Accuracy: 99.71800231933594, Test Loss: 0.10924376547336578, Test Accuracy: 97.68000030517578

model.summary()

Model: "model"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

==================================================================================================

input\_1 (InputLayer) [(None, 800)] 0

...

tf\_op\_layer\_Softmax (TensorFlow [(None, 10)] 0 dense\_2[0][0]

==================================================================================================

Total params: 85,220,510

Trainable params: 2,514,310

Non-trainable params: 82,706,200

model(tf.constant([test\_[0]]))

<tf.Tensor: id=3682937, shape=(1, 10), dtype=float32, numpy=

array([[1.0000000e+00, 3.1490074e-11, 9.4406474e-17, 1.0289336e-14,

5.9060578e-14, 4.8893082e-16, 1.5400805e-16, 2.4614913e-13,

1.4353953e-12, 9.7358359e-16]], dtype=float32)>

test\_label[0]

array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0])

test\_dataset = tf.data.Dataset.from\_tensor\_slices(test\_)

test\_dataset = test\_dataset.batch(batch\_size=256)

predictions=[]

**for** line **in** test\_dataset:

prediction = model(line)

predictions.extend(list(np.argmax(prediction.numpy(),axis=1)))

test\_.shape

(10000, 800)

**from** **sklearn.metrics** **import** accuracy\_score

test\_true = list(np.argmax(test\_label,axis=1))

accuracy\_score(test\_true,predictions)

0.9827

**from** **sklearn.metrics** **import** classification\_report

print(classification\_report(test\_true,predictions,target\_names=list(lb.classes\_)))

precision recall f1-score support

体育 1.00 1.00 1.00 1000

娱乐 0.99 0.99 0.99 1000

家居 0.99 0.94 0.96 1000

房产 1.00 1.00 1.00 1000

教育 0.99 0.95 0.97 1000

时尚 0.98 0.99 0.98 1000

时政 0.98 0.98 0.98 1000

游戏 0.99 0.99 0.99 1000

科技 0.95 1.00 0.97 1000

财经 0.97 0.99 0.98 1000

accuracy 0.98 10000

macro avg 0.98 0.98 0.98 10000

weighted avg 0.98 0.98 0.98 10000