Embedding层（嵌入层）的理解

转载自：原文：https://blog.csdn.net/weixin_42078618/article/details/84553940

首先，我们有一个one-hot编码的概念。

假设，我们中文，一共只有10个字。。。只是假设啊，那么我们用0-9就可以表示完

比如，这十个字就是“我从哪里来，要到何处去”

其分别对应“0-9”，如下：

我 从 哪 里 来 要 到 何 处 去

0 1 2 3 4 5 6 7 8 9

那么，其实我们只用一个列表就能表示所有的对话

如：我 从 哪 里 来 要 到 何 处 去 ——>>>[0 1 2 3 4 5 6 7 8 9]

或：我 从 何 处 来 要 到 哪 里 去 ——>>>[0 1 7 8 4 5 6 2 3 9]

但是，我们看看one-hot编码方式（详见：https://blog.csdn.net/tengyuan93/article/details/78930285）

他把上面的编码方式弄成这样

```
# 我从哪里来，要到何处去
[
[1 0 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0 0]
[0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0 0]
[0 0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 0 0 1]
]

# 我从何处来，要到哪里去
[
[1 0 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0 0]
[0 0 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]
[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 0 0]
[0 0 0 1 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 1]
```

]
即：把每一个字都对应成一个十个（样本总数/字总数）元素的数组/列表，其中每一个字都用唯一对应的数组/列表对应，数组/列表的唯一性用1表示。如上，“我”表示成[1。。。。]，“去”表示成[。。。。1]，这样就把每一系列的文本整合成一个稀疏矩阵。

那问题来了，稀疏矩阵（二维）和列表（一维）相比，有什么优势。

很明显，计算简单嘛，稀疏矩阵做矩阵计算的时候，只需要把1对应位置的数相乘求和就行，也许你心算都能算出来；而一维列表，你能很快算出来？何况这个列表还是一行，如果是100行、1000行和或1000列呢?

所以，one-hot编码的优势就体现出来了，计算方便快捷、表达能力强。

然而，缺点也随着来了。

比如：中文大大小小简体繁体常用不常用有十几万，然后一篇文章100W字，你要表示成100W X 10W的矩阵？？？

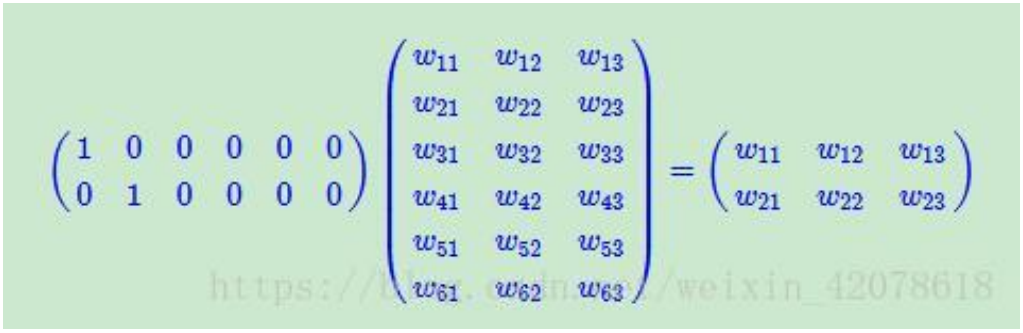
这是它最明显的缺点。过于稀疏时，过度占用资源。

比如：其实我们这篇文章，虽然100W字，但是其实我们整合起来，有99W字是重复的，只有1W字是完全不重复的。那我们用100W X 10W的岂不是白白浪费了99W X 10W的矩阵存储空间。

那怎么办？？？

这时，Embedding层横空出世。

接下来给大家看一张图



假设：我们有一个2 x 6的矩阵，然后乘上一个6 x 3的矩阵后，变成了一个2 x 3的矩阵。

先不管它什么意思，这个过程，我们把一个12个元素的矩阵变成6个元素的矩阵，直观上，大小是不是缩小了一半?

也许你已经想到了！！！对！！！不管你想的对不对，但是embedding层，在某种程度上，就是用来降维的，降维的原理就是矩阵乘法。在卷积网络中，可以理解为特殊全连接层操作，跟1x1卷积核异曲同工！！！484很神奇！！！

也就是说，假如我们有一个100W X10W的矩阵，用它乘上一个10W X 20的矩阵，我们可以把它降到100W X 20，瞬间量级降了。。。10W/20=5000倍！！！

这就是嵌入层的一个作用——降维。

然后中间那个10W X 20的矩阵，可以理解为查询表，也可以理解为映射表，也可以理解为过度表，whatever。

接着，既然可以降维，当然也可以升维。为什么要升维?

这张图，我要你在10米开外找出五处不同！。。。What？烦请出题者走近两步，我先把我的刀拿出来，您再说一遍题目我没听清。

当然，目测这是不可能完成的。但是我让你在一米外，也许你一瞬间就发现衣服上有个心是不同的，然后再走近半米，你又发现左上角和右上角也是不同的。再走近20厘米，又发现耳朵也不同，最后，在距离屏幕10厘米的地方，终于发现第五个不同的地方在耳朵下面一点的云。

但是，其实无限靠近并不代表认知度就高了，比如，你只能距离屏幕1厘米远的地方找，找出五处不同。。。出题人你是不是脑袋被门挤了。。。

由此可见，距离的远近会影响我们的观察效果。同理也是一样的，低维的数据可能包含的特征是非常笼统的，我们需要不停地拉近拉远来改变我们的感受野，让我们对这幅图有不同的观察点，找出我们要的茬。

embedding的又一个作用体现了。对低维的数据进行升维时，可能把一些其他特征给放大了，或者把笼统的特征给分开了。同时，这个embedding是一直在学习在优化的，就使得整个拉近拉远的过程慢慢形成一个良好的观察点。比如：我来回靠近和远离屏幕，发现45厘米是最佳观测点，这个距离能10秒就把5个不同点找出来了。

回想一下为什么CNN层数越深准确率越高，卷积层卷了又卷，池化层池了又升，升了又降，全连接层连了又连。因为我们也不知道它什么时候突然就学到了某个有用特征。但是不管怎样，学习都是好事，所以让机器多卷一卷，多连一连，反正错了多少我会用交叉熵告诉你，怎么做才是对的我会用梯度下降算法告诉你，只要给你时间，你迟早会学懂。因此，理论上，只要层数深，只要参数足够，NN能拟合任何特征。总之，它类似于虚拟出一个关系对当前数据进行映射。这个东西也许一言难尽吧，但是目前各位只需要知道它有这些功能的就行了。

接下来，继续假设我们有一句话，叫“公主很漂亮”，如果我们使用one-hot编码，可能得到的编码如下：

公 [0 0 0 0 1]
主 [0 0 0 1 0]
很 [0 0 1 0 0]
漂 [0 1 0 0 0]
亮 [1 0 0 0 0]
乍一眼看过似乎没毛病，其实本来人家也没毛病，或者假设咱们的词袋更大一些

公 [0 0 0 0 1 0 0 0 0 0]
主 [0 0 0 1 0 0 0 0 0 0]
很 [0 0 1 0 0 0 0 0 0 0]
漂 [0 1 0 0 0 0 0 0 0 0]

亮 [1 0 0 0 0 0 0 0 0 0]
假设吧，就假设咱们的词袋一共就10个字，则这一句话的编码如上所示。

这样的编码，最大的好处就是，不管你是什么字，我们都能在一个一维的数组里用01给你表示出来。并且不同的字绝对不一样，以致于一点重复都没有，表达本征的能力极强。

但是，因为其完全独立，其劣势就出来了。表达关联特征的能力几乎为0！！！

我给你举个例子，我们又有一句话“王妃很漂亮”

那么在这基础上，我们可以把这句话表示为

王 [0 0 0 0 0 0 0 0 0 1]
妃 [0 0 0 0 0 0 0 0 1 0]
很 [0 0 1 0 0 0 0 0 0 0]
漂 [0 1 0 0 0 0 0 0 0 0]
亮 [1 0 0 0 0 0 0 0 0 0]
从中文表示来看，我们一下就跟感觉到，王妃跟公主其实是有很大关系的，比如：公主是皇帝的女儿，王妃是皇帝的妃子，可以从“皇帝”这个词进行关联上；公主住在宫里，王妃住在宫里，可以从“宫里”这个词关联上；公主是女的，王妃也是女的，可以从“女”这个字关联上。

但是呢，我们用了one-hot编码，公主和王妃就变成了这样：

公 [0 0 0 0 1 0 0 0 0 0]
主 [0 0 0 1 0 0 0 0 0 0]
王 [0 0 0 0 0 0 0 0 0 1]
妃 [0 0 0 0 0 0 0 0 1 0]
你说，你要是不看前面的中文注解，你知道这四行向量有什么内部关系吗？看不出来，那怎么办？

既然，通过刚才的假设关联，我们关联出了“皇帝”、“宫里”和“女”三个词，那我们尝试这么去定义公主和王妃

公主一定是皇帝的女儿，我们假设她跟皇帝的关系相似度为1.0；公主从一出生就住在宫里，直到20岁才嫁到府上，活了80岁，我们假设她跟宫里的关系相似度为0.25；公主一定是女的，跟女的关系相似度为1.0；

王妃是皇帝的妃子，没有亲缘关系，但是有存在着某种关系，我们就假设她跟皇帝的关系相似度为0.6吧；妃子从20岁就住在宫里，活了80岁，我们假设她跟宫里的关系相似度为0.75；王妃一定是女的，跟女的关系相似度为1.0；

于是公主王妃四个字我们可以这么表示：

皇 宫
帝 里 女
公主 [1.0 0.25 1.0]
王妃 [0.6 0.75 1.0]
这样我们就把公主和王妃两个词，跟皇帝、宫里、女这几个字（特征）关联起来了，我们可以认为：

公主=1.0 *皇帝 +0.25*宫里 +1.0*女

王妃=0.6 *皇帝 +0.75*宫里 +1.0*女

或者这样，我们假设没歌词的每个字都是对等（注意：只是假设，为了方便解释）：

皇 宫
帝 里 女
公 [0.5 0.125 0.5]
主 [0.5 0.125 0.5]
王 [0.3 0.375 0.5]
妃 [0.3 0.375 0.5]
这样，我们就把一些词甚至一个字，用三个特征给表征出来了。然后，我们把皇帝叫做特征（1），宫里叫做特征（2），女叫做特征（3），于是乎，我们就得出了公主和王妃的隐含特征关系：

王妃=公主的特征（1） * 0.6 +公主的特征（2） * 3 +公主的特征（3） * 1

于是乎，我们把文字的one-hot编码，从稀疏态变成了密集态，并且让相互独立向量变成了有内在联系的关系向量。

所以，embedding层做了个什么呢？它把我们的稀疏矩阵，通过一些线性变换（在CNN中用全连接层进行转换，也称为查表操作），变成了一个密集矩阵，这个密集矩阵用了N（例子中N=3）个特征来表征所有的文字，在这个密集矩阵中，表象上代表着密集矩阵跟单个字的一一对应关系，实际上还蕴含了大量的字与字之间，词与词之间甚至句子与句子之间的内在关系（如：我们得出的王妃跟公主的关系）。他们之间的关系，用的是嵌入层学习来的参数进行表征。从稀疏矩阵到密集矩阵的过程，叫做embedding，很多人也把它叫做查表，因为他们之间也是一个一一映射的关系。

更重要的是，这种关系在反向传播的过程中，是一直在更新的，因此能在多次epoch后，使得这个关系变成相对成熟，即：正确的表达整个语义以及各个语句之间的关系。这个成熟的关系，就是embedding层的所有权重参数。

Embedding是NPL领域最重要的发明之一，他把独立的向量一下子就关联起来了。这就相当于什么呢，相当于你是你爸的儿子，你爸是A的同事，B是A的儿子，似乎跟你是八竿子才打得着的关系。结果你一看B，是你的同桌。Embedding层就是用来发现这个秘密的武器。

原文：https://blog.csdn.net/weixin_42078618/article/details/84553940