算法专栏（七）word2vec原理学习分享

辣鱼 辣鱼编程 6月27日

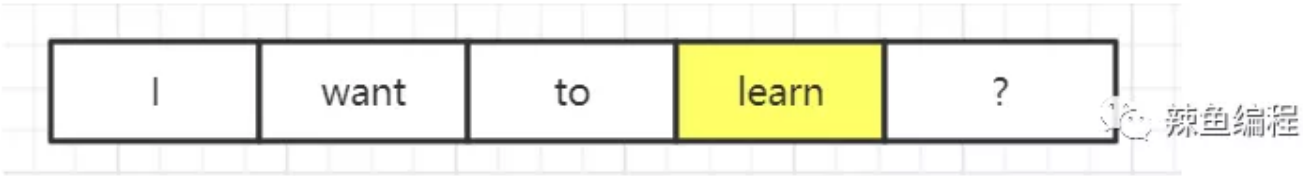
word2vec 原理

一、引入问题

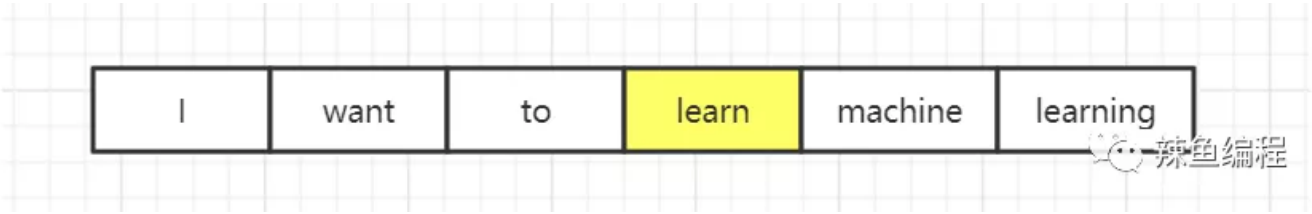
word2vec从字面意义上来说，就是把一个word转成一个vector（向量），比如说 苹果这个单词，用向量表示是[0.2,0.5,0.4,0,3]

看上去，input是一个word，output是一个vector，但是我们先去了解另一个问题，后面再绕回来。

这个问题就是 怎么用前面的词预测后面的词，举个例子来说，比如下图的句子

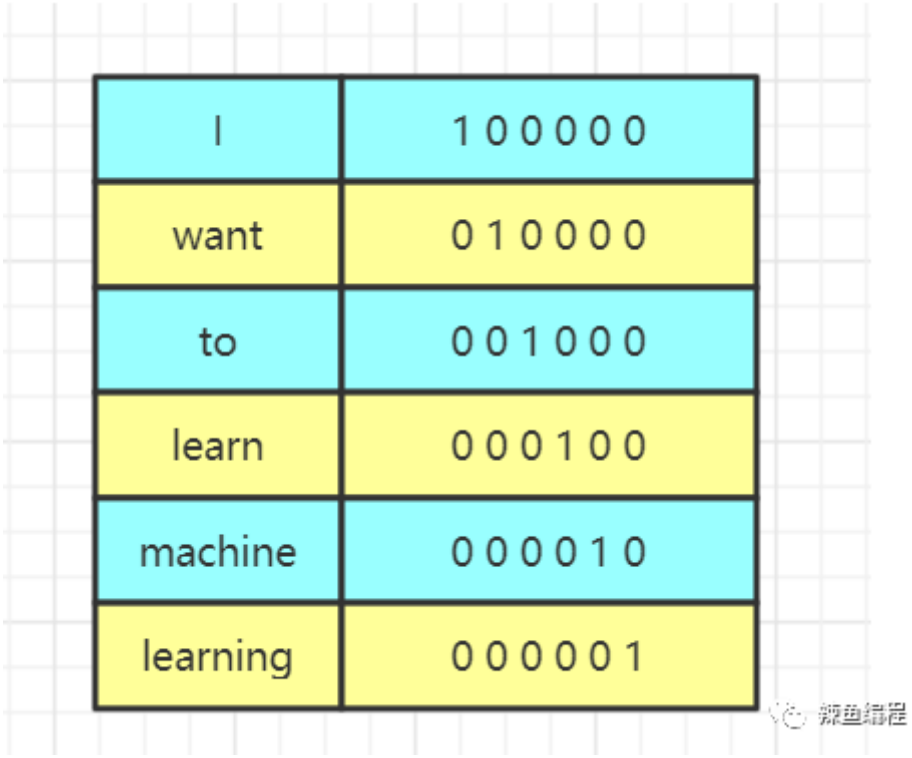


但我知道当前单词是learn的时候，我可以预测下一个词



二、word2vec 训练数据长什么屌样

我们想让机器学会，learn的下一个单词是machine ，这件事，但是机器肯定是不认识“learn”，还有“machine”，所以我们需要给这些单词编码，用常见的one-hot编码可以得到如下编码：

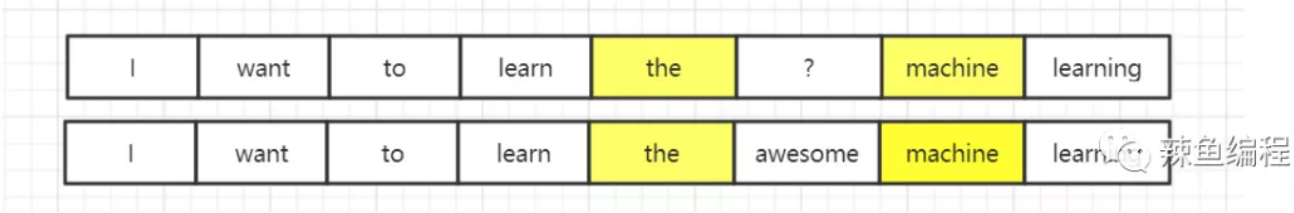


one-hot编码还是很容易理解的，首先有多少个单词，one-hot向量的长度就是多少，然后在自己的位置是1，其他位置都是0，这样我们就把所有的单词给编码好了

所以我们原本的问题 如何通过 "learn" 预测 "machine"

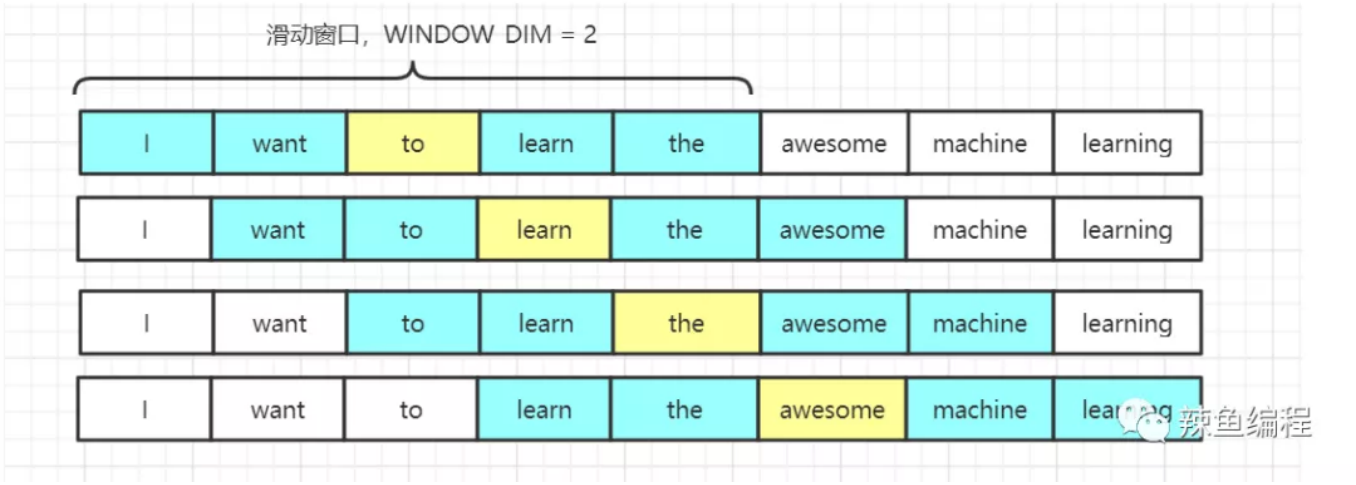
就转换成了，如何通过[0,0,0,1,0,0] 预测 [0,0,0,0,1,0]

但是我们考虑到还有这种情况



当我们预测上图的？的时候，我们除了参考前面的 the, 还参考了后面的machine, 所以我们才知道这里要填的是一个adj，所以其实预测词填什么，除了跟这个词前面的单词有关，其实还跟后面的词有关

所以自然而然，我们就引入一个概念叫 滑动窗口



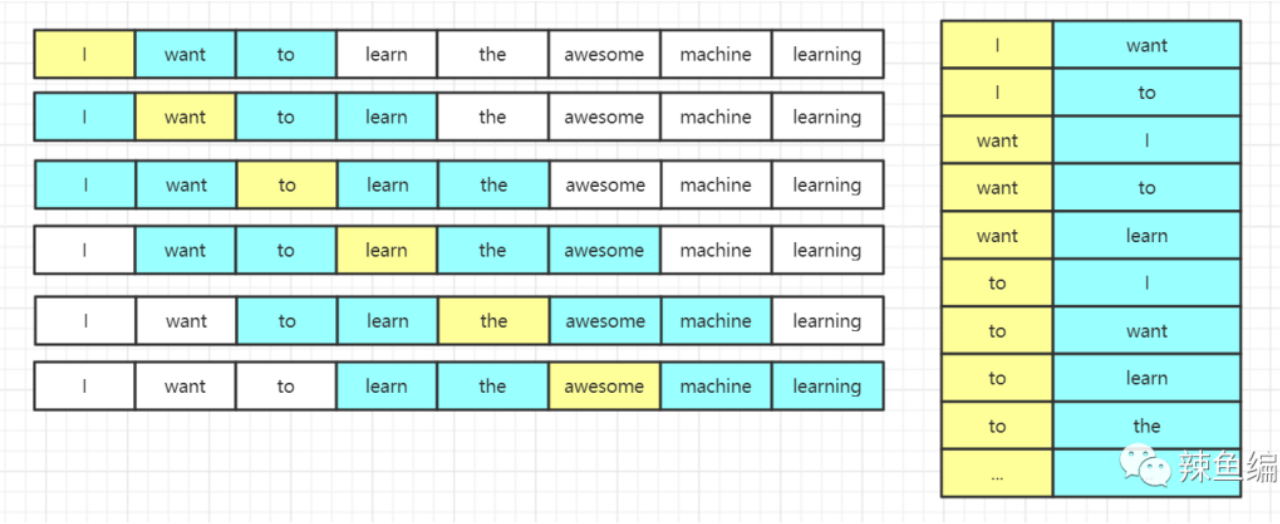
当我们滑到中心词是“to”的时候，左右两边两格就是它对应的背景词；再往右滑一格，中心词就变成了“learn”，背景词有want、to、the、awesome这四个，以此类推。这里可以讲word2vec的两种模型了， CBOW 与Skip-Gram。 CBOW模型， 全称是Continuous Bag-of-Words， 当然畏惧英文的话，我们可以叫他的中文名，叫连续词袋模型；然后Skip-Gram，叫跳字模型 。这两个模型听上去挺高大上，其实就是，如果你是中心词去预测背景词的话，就叫CBOW；如果反过来，你是用背景词去预测中心词的话，就是Skip-Gram。

拿上面的图做例子就是：

连续词袋模型：
（to）-> （I, want, learn, the）
//用 黄色 预测 蓝色

跳字模型：
（I, want, llearn, the） -> (to)
// 用 蓝色 预测 黄色

了解完两个模型之后，我们下一步，开始采集训练数据



对于上图所示的采集训练数据，有几个注意点你可能需要注意

* 1、开头或者结束位置，左右窗口有一些是没有的，比如当中心词是“I"的时候，只有右窗口有，那就只采集右边窗口
* 2、如果中心词和背景词是同一个，那么不采集（不采集的原因后面会说明）
* 3、这里是用连续词袋模型示例，如果是跳字模型，把两列换一下就可以了，具体是[[want， I]，[to, I]，[I want]...]

现在我们已经有训练数据了，具体的，我们有输入，和对应的输出

i - > want

i -> to

.....

结合上面的one-hot，我们的input-output pair是

[1 0 0 0 0 0] -> [0 1 0 0 0 0]

[1 0 0 0 0 0] -> [0 0 1 0 0 0]

.....

到现在，我们已经知道了数据是怎么来的，下一步我们介绍一下模型

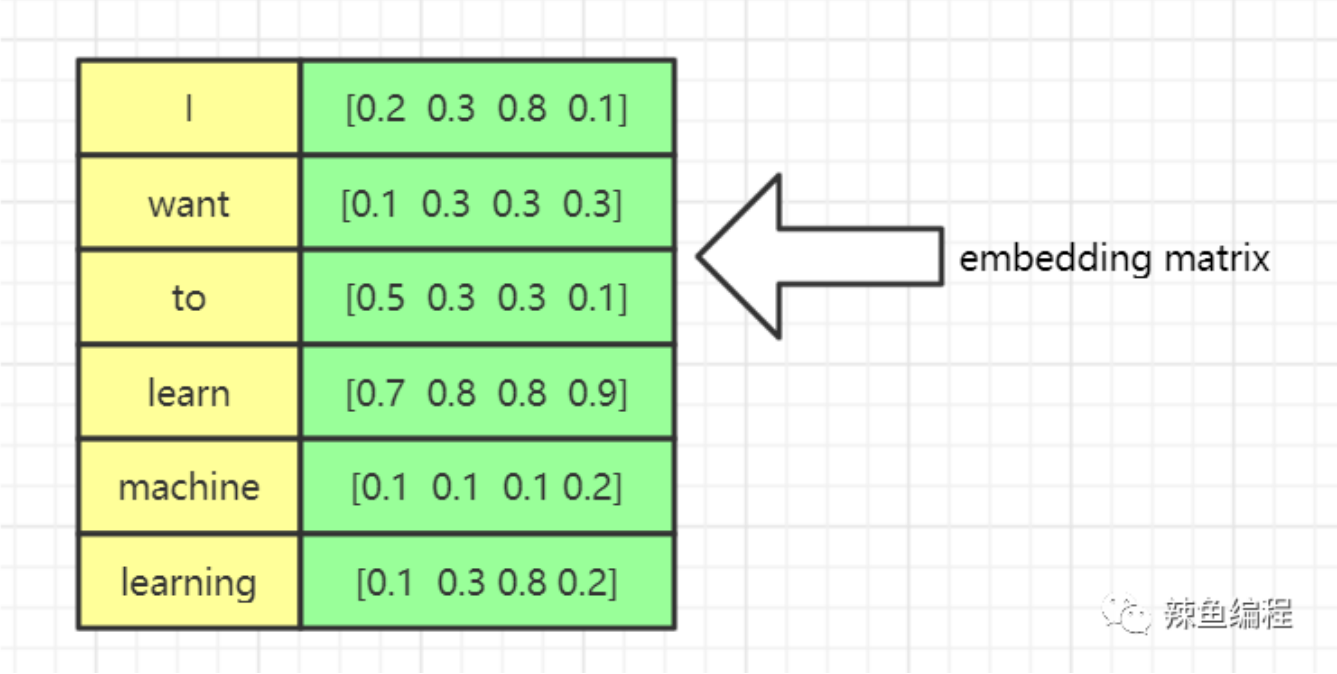
三、word2vec模型长什么屌样

3.1 介绍两个矩阵

为了更好地理解模型，需要先介绍两个矩阵。首先，我们知道，每个词，有两重身份！当滑到它的时候，它的身份是“中心词”；如果在其他“中心词”旁边的话，它的身份就成为了“背景词”。所以同一个词，比如“learn”这个词

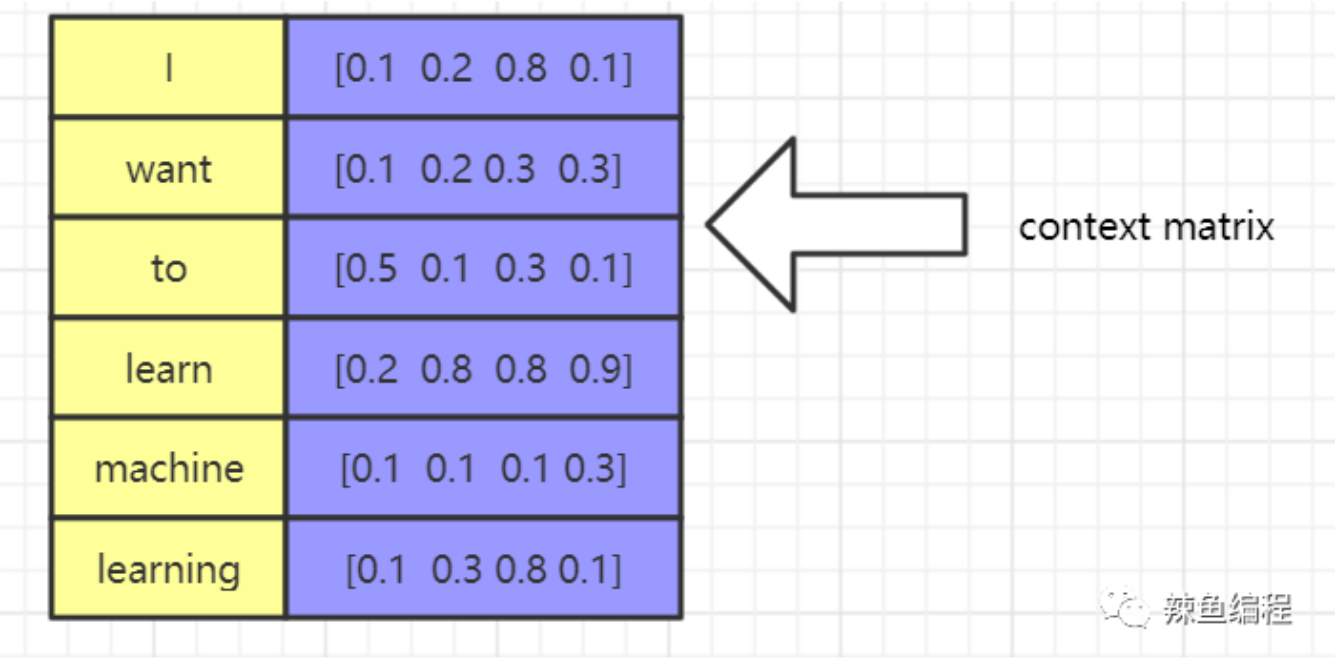
- 作为中心词的时候，向量表示有可能是[0.2 0.3 0.3 0.5]
- 而作为背景词的时候，可能又是另一个向量 [0.3 0.8 0.2 0.1]

我们把所有单词的，中心词向量一个个叠起来，就得到了一个嵌入矩阵



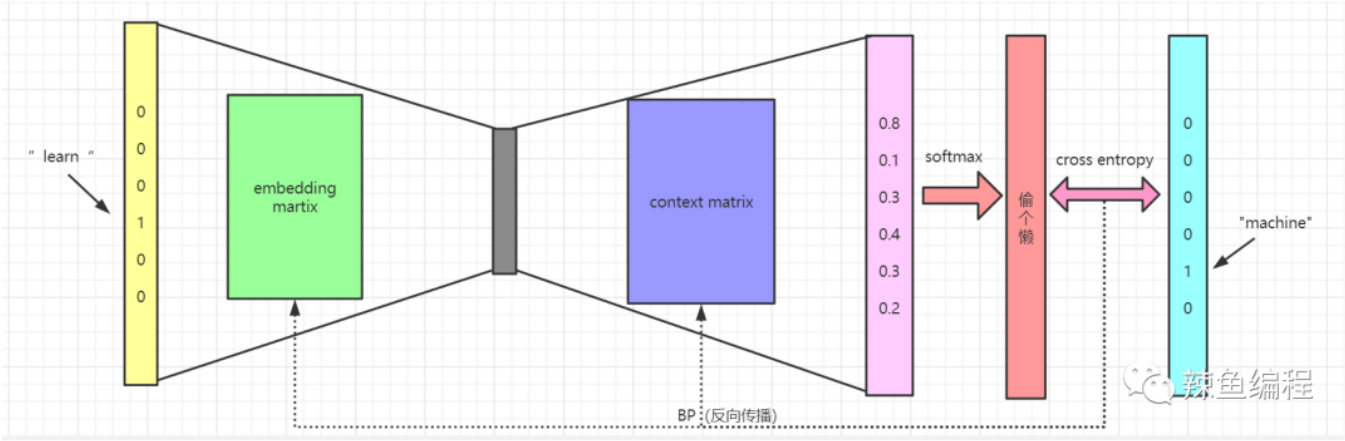
至于里面这些向量的数值，只是随便填进去的，因为后面我们会训练，所以随机初始化就可以了

同理，我们把所有单词的背景词向量一行行叠起来，就能得到另一个矩阵，叫做背景矩阵



3.2 模型介绍

有了这两个矩阵之后，就可以介绍一下模型了，其实就是一个简单的三层神经网络



input vector 经过embedding matrix 到隐藏层（hidden layer），然后hidden layer再经过context matrix 得到粉红色那个，可以理解为每个单词的得分，比如 learn 后面接 i 的得分是0.8，learn后面接want 的得分是0.1。我们经过一层softmax（不了解softmax的童鞋可以先去搜一下）

$$softmax(0.8) = \frac{e^{0.8}}{e^{0.1} + e^{0.3} + e^{0.4} + e^{0.3} + e^{0.2}}$$

然后，就是交叉熵（cross entropy），如果不了解交叉熵，可以搜一下，顺便学一下

$$cross\_entropy = -\sum_{i=1}^{n} y_i log(\hat{y}_i)$$

有交叉熵作为 loss function， 接下来就是一轮一轮的训练了，通过BP（反向传播）修改两个矩阵的权重，使得损失函数最小

花了这么大篇幅，终于把原来的问题预测下一个词解决了

3.3 醉翁之意不在酒，其实我们也可以要绿色那块

当整个模型训练好之后，我们就可以根据input预测下一个单词是什么了

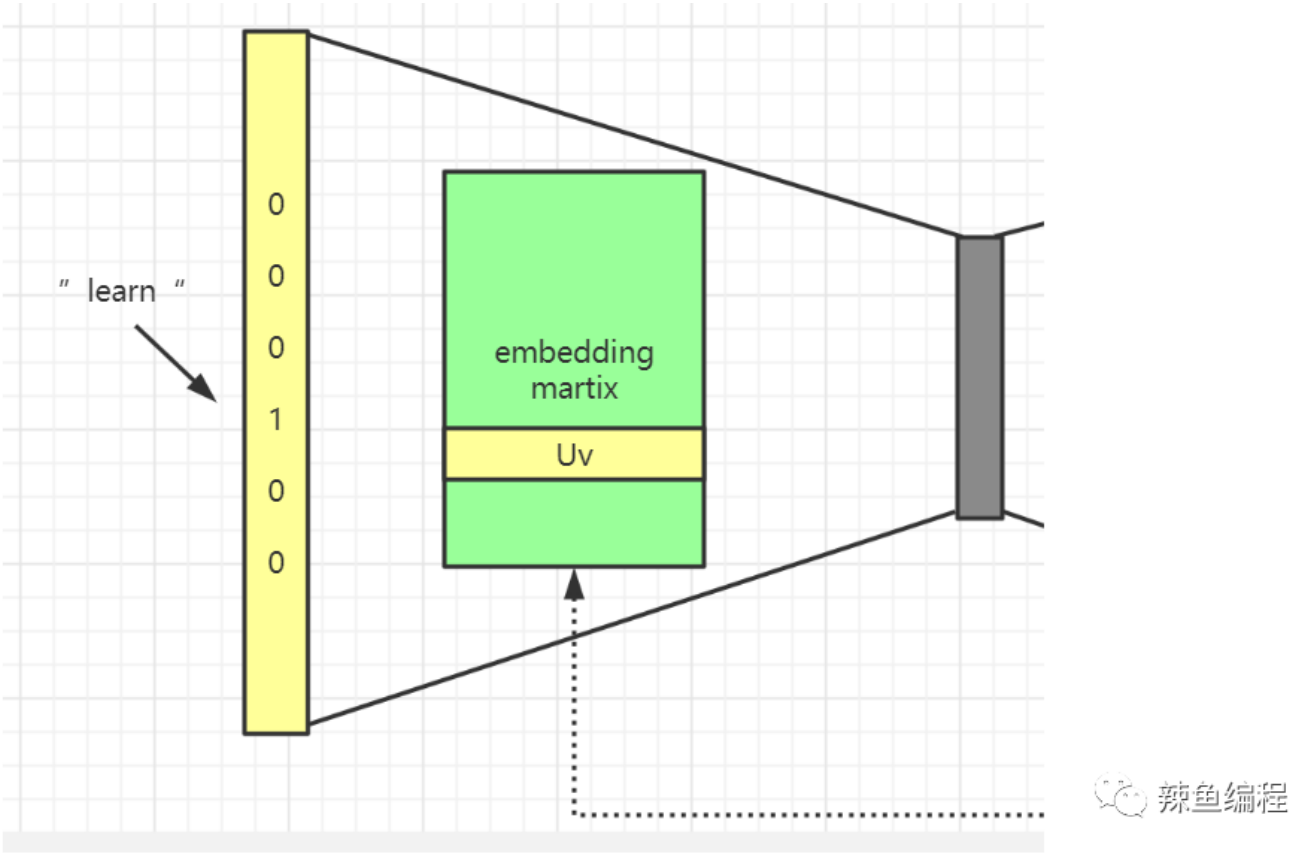
但是换个角度，这块绿色的东西好像挺有用的

我们重新看下input，是one-hot编码的，经过embedding matrix 得到hidden layer，那么input是只有一个位置是1，其余地方都是0

乘以embedding matrix的时候，就相当于 pick up 了一行，所以有的资料也把这个 embedding matrix 叫做 look-up table

高是vocab_size(所有的单词数量)，宽是embedding dim(这个可以随便定，就是把一个单词向量化为多少维度的，你喜欢)

然后每一行其实就可以理解为每个word的特征表示(如下图的Uv)



- 这个特征，我用embedding vector 来表示， embedding vector 跟 one-hot 相比，有两个好处

1. 本质上来说，降维了（vocab_size -> embedding size)
2. 能够找到词之间的关系，比如粤语的"钟意"，跟普通话的"喜欢"，其实是差不多意思

为什么能得"钟意"跟"喜欢"的相似度很近，因为他们的上下文都是差不多的，我喜欢你，我钟意你

这样，任意给定两个词，我可以计算他们的相似度，也就是"语义"上面的相似度

任意给定一个词，我可以找到最接近它的词是什么

这样我们就把word2vec的基本模型讲清楚了，回到上面提到的一个问题，因为我们想比较两个词向量之间的相似度，所以（input，output）

如果是同一个词的话，我们就不希望它加进来，因为两个词向量是一样的，他们的相似度当然是0

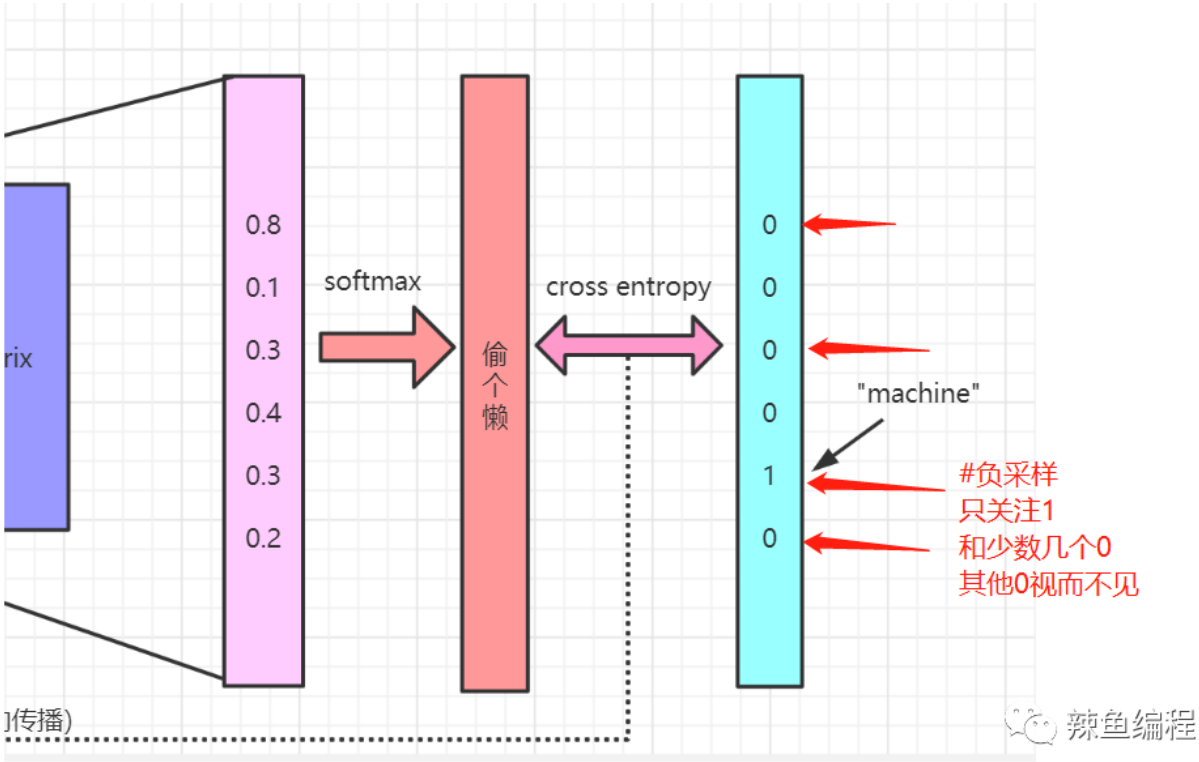
这样子在少数量级的数据上可以跑了，但大数据集的时候就跑不动啦

四、一些近似求解

当我们的文本数据很大的时候，比如这个互联网的wiki文档里面的单词，那vocab_size就会很大，也就是下面的粉红色、红色、蓝色会很高很高很高。

这样计算softmax的时候，太费时间了！所以对于word2vec有两种近似求解，一种是负采样，另一种是层次哈夫曼。层次哈夫曼就是利用计算机里面

讲到的那个哈夫曼树，将问题从o（N） 降到 o（log N），而负采样就是如下图所示，只用1和少数几个0，更新权重，其他的就当看不见
O（N) 到 O（K） ，k是负采样的数量。



五、简单码一下三层网络

```
import tensorflow as tf
x = tf.placeholder(tf.float32, shape=(None, vocab_size))
y_label = tf.placeholder(tf.float32, shape=(None, vocab_size))


EMBEDDING_DIM = 5


W1 = tf.Variable(tf.random_normal([vocab_size,EMBEDDING_DIM]))
b1 = tf.Variable(tf.random_normal([EMBEDDING_DIM]))
hidden = tf.add(tf.matmul(x,W1),b1)


W2 = tf.Variable(tf.random_normal([EMBEDDING_DIM,vocab_size]))
b2 = tf.Variable(tf.random_normal([vocab_size]))
prediction = tf.nn.softmax(tf.add(tf.matmul(hidden, W2), b2))


sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)


cross_entropy_loss = tf.reduce_mean(-tf.reduce_sum(y_label * tf.log(predictio
train_step = tf.train.GradientDescentOptimizer(0.1).minimize(cross_entropy_lo


batch_size = 10000


for _ in range(batch_size):
    sess.run(train_step,feed_dict={x:x_train,y_label:y_train})
    print("loss is:", sess.run(cross_entropy_loss,feed_dict={x:x_train,y_labe
```

> 这篇主要讲原理，就不贴完整代码了