推荐系统从零单排系列(四)—Word2Vec理论与实践(上)

原创 可爱又迷人的反派角色宁宁 机器学习荐货情报局 2019-05-11

> 【导读】Word2Vec是Embedding中非常基本的模型，训练出来的词向量不仅能保持语义与语法上的相关性，并且可以实现类似代数运算的能力。除了在NLP中作为基本网络模块，在推荐系统等领域中也是应用广泛，相信聪明的你肯定看到过类似的介绍，但是又没有深入、系统的学习过。没关系，让我们现在开始来一步一步的解决这个问题。

One-Hot encoding

最基本的也是最简单的把word转换成vector的办法就是通过计数word在文档中出现的次数，这样的表达方式称为 one hot 或者 count vectorizing 。假设词典共有V个词，那么每一个单词都有一个V维度的向量来表示，向量中只有一个位置为非0（可以是1，也可以是出现次数），表示该词的编号，其余为0。这样的表示很容易带来维度灾难，而且表示非常的稀疏，一个普通的词典中可能就包含上百万个词。借助 sklearn 可以轻松的实现该表达。

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
corpus = [
    'Text of first document.',
    'Text of the second document make longer.',
    'Number three.',
    'This is number four.',
]

# store CountVectorizer sparse matrix in X
# The column of matrix is words, rows are documents
X = vectorizer.fit_transform(corpus) #(4, 13)
print(vectorizer.get_feature_name())
print(X.toarray())

# Convert a new document to count representation
vectorizer.transform(['This is a new document'])
```

得到一个非常稀疏的矩阵，每行表示一个Document，每一列表示一个Word。

Word2Vec

Word2vec是在Google Tomas Mikolov在2013年提出来的，作用同样是将word转换成vector表达，但是这样的vector是低维度的实数值。学习到的这个vector就是Embedding/词嵌入向量。

Word2Vec包括两种模型，CBOW与Skip-gram。前者利用单词上下文预测单词，后者利用单词预测上下文。本文主要关注CBOW模型。

CBOW模型

> CBOW全称是Continuous Bag Of Words, 是指利用上下文单词来预测一个词。这个上下文可以只有上文/下文，也可以上下文都有。具体是多长的上下文，是人为指定的，也称为窗口大小。

One-word Context

先来看最简单的情况，s目标是：给出当前上下文（只有一个词），求出下一个词的概率分布。也就是下一个词是字典中的某个词的概率，概率有大有小，最大的那个就是模型预测出来的下一个词。为了保证这个词真的是语料中下一个词，我们需要调整概率分布使得其概率最大，这个过程就是学习训练的过程。
此时的网络结构如下图所示。V是词典中word的数量，N表示词向量的维度/嵌入的维度。

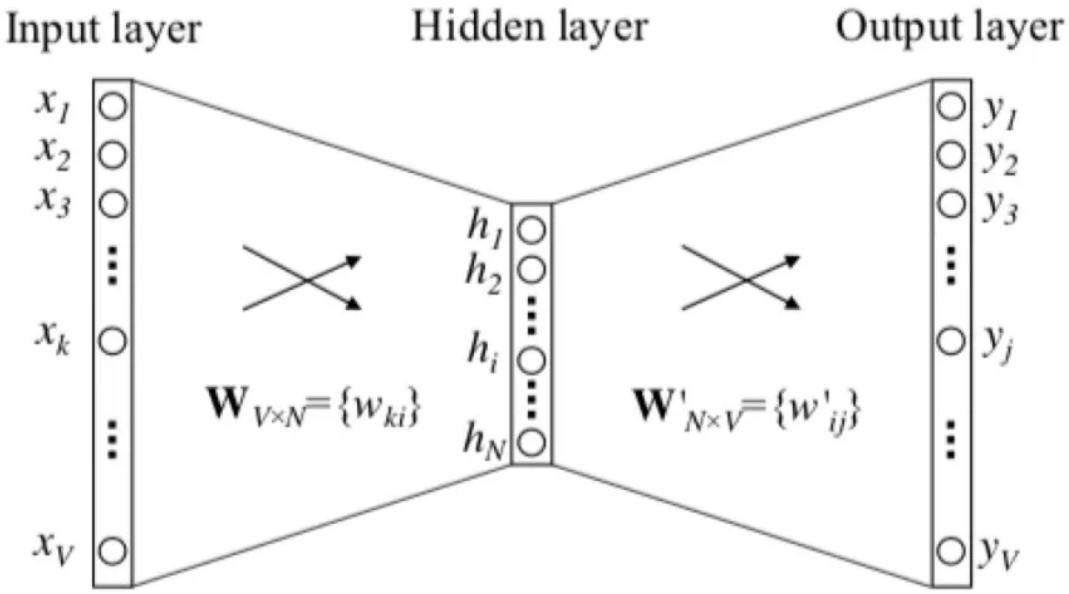


Figure 1: A simple CBOW model with only one word in the context

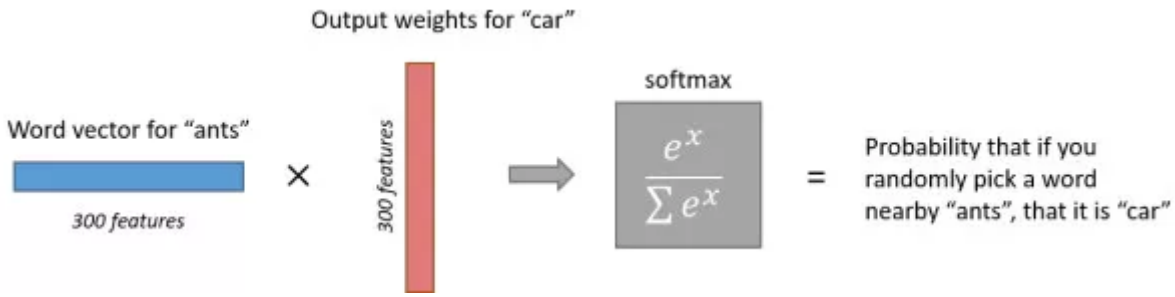
- 输入只有一个单词，用one hot的形式表示，V维度
- 中间隐藏层维度为N，也就是嵌入的维度，**没有激活函数。**
- 输出层使用*softmax*激活函数，维度为V

> 这是最基本的模型，使用softmax计算复杂度为O(V)，可以使用层次softmax 或 Negative Sampling进行优化。此处不再展开，后续会专门来写。

所以，是不是非常简单？输入，输出都是V维向量，中间一个隐藏层。而且，输入是one hot形式的V维度向量，输出使用softmax激活函数，隐藏层没有激活函数。
鉴于输入是one hot形式的，只有一个非零值且为1，所以相当于是在input -> hidden layer权重矩阵中选择第k行，k对应输入one hot中不为零的那一项，如下图所示。此时的隐藏层矩阵h，我们成为输入单词的**input vector**，维度为N。

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

hidden->output layer 矩阵是另外一个矩阵，它完成从隐藏层到输出层的映射。 W’ 中每一列依次与h相乘，就对应着输出层中每一个神经元的值，而输出层的每一个神经元对应着一个单词。如下图所示。



所以 W’每一列也对应着字典中一个单词，我们称该列为对应单词的**output vector**，维度为N。

> input vector和output vector是理解word2vec的关键，务必要搞清楚。聪明如你，一定可以理解啦

你可能要问了，word vector / embedding / 词嵌入向量到底在哪里那?
答案：**input vector就是对应词的嵌入向量。**

也就是说，我们最终想要的就是input -> hidden layer的权重矩阵，矩阵 shape=(V,N) ，每一行都是一个N维向量，就是对应位置的词向量。

Multi-word Context

上下文只有一个单词的情况清楚后，上下文有多个单词的情况也就非常简单了。输入由原来的一个one hot vector，变成如今的多个one hot vector。他们使用**同一个**input -> hidden layer权重矩阵，选出其中对应的行。然后对其**取平均值**作为中间隐藏层的输出。其余的和单一上下文的情况是相同的。
如下图所示：

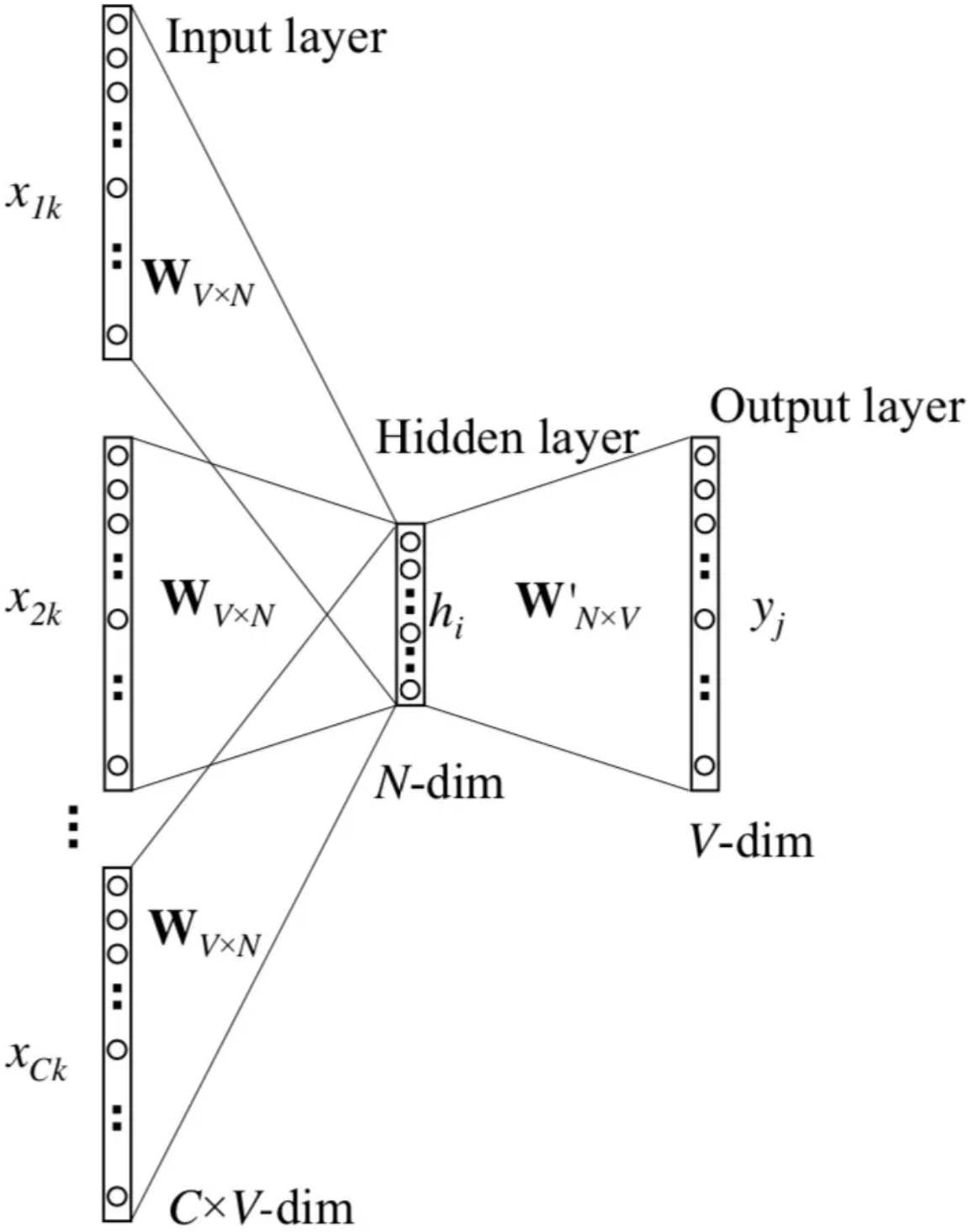


Figure 2: Continuous bag-of-word model

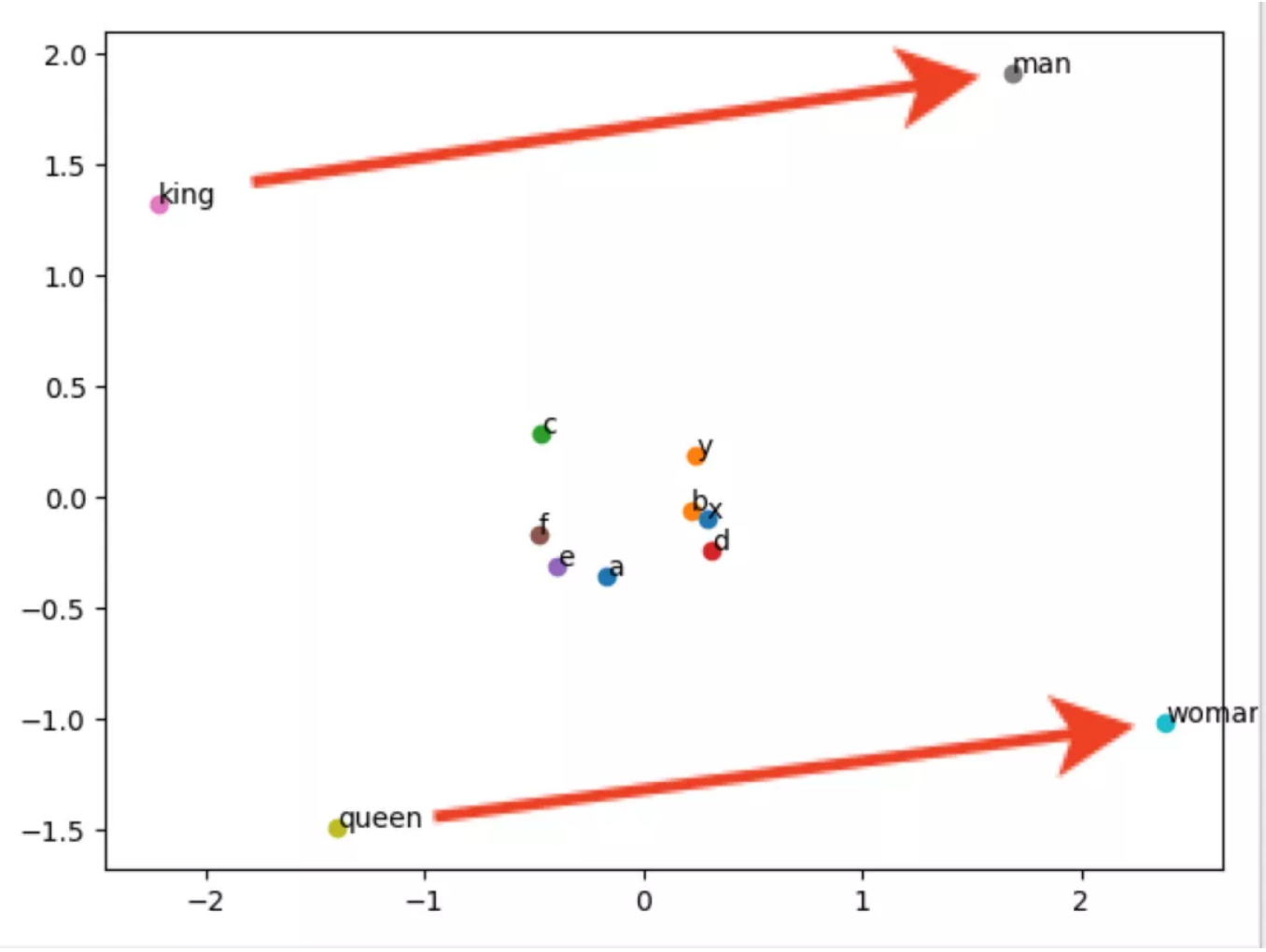
用tensorflow可以实现非常简单的One-word Context版本：

```
"""
1. 准备训练数据
"""
# Context|Target
corpus_king_queen_symbol = ['king|a','queen|a','king|b','queen|b','king|c','queen|c','k
```

```
                    'queen|y','man|d','woman|d','man|e','woman|e','man|f','woma
                    'man|x','woman|y']

train_data = [sample.split('|')[0] for sample in corpus_king_queen_symbol]
train_label = [sample.split('|')[1] for sample in corpus_king_queen_symbol]

vocabulary = (list(set(train_data) | set(train_label)))
vocabulary.sort()

one_hot_encoder = OneHotEncoder()
one_hot_encoder.fit(np.reshape(vocabulary, (-1, 1)))

X = one_hot_encoder.transform(np.reshape(train_data, (-1, 1))).toarray()
y = one_hot_encoder.transform(np.reshape(train_label, (-1, 1))).toarray()

"""
2. 构建模型

输入是X，y
"""
N =  5
V = len(vocabulary)

inputs = Input(shape=(V, ))
x = Dense(N, activation='linear', use_bias=False)(inputs)
predictions = Dense(V, activation='softmax', use_bias=False)(x)

model = Model(inputs=inputs, outputs=predictions)
model.summary()

"""
3. 训练模型
"""
model.compile(optimizer=keras.optimizers.Adagrad(0.07),
              loss='categorical_crossentropy',
              metrics=['accuracy', 'mse'])
model.fit(X, y,
          batch_size=1,
          epochs=1000
          )

"""
4. 验证/可视化结果
"""
weights = model.get_weights()
embeddings = np.array(weights[0])
assert (embeddings.shape == (V, N))
word_vec = dict((word, vector) for word, vector in zip(vocabulary, embeddings))

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(embeddings)
print(X_pca)

fig, ax = plt.subplots()
for i in range(len(X_pca)):
    team = X_pca[i]
    ax.scatter(team[0], team[1])
```

```
    ax.annotate(vocabulary[i], (team[0], team[1]))
plt.show()
```

训练之后，可以看到经典的 (king - man) + woman = queen 如下图所示：



完整代码见github: https://github.com/gutouyu/ML_CIA/blob/master/Embedding/Word2Vec.py

预告：接下来三篇文章我们将依次介绍Word2Vec的剩余部分，包括：Skip-gram模型、Hierarchical Softmax、Negative sampling。跟着小编一步一步彻底搞懂Word2Vec，敬请期待~

往期回顾

推荐系统从零单排系列(三)--再谈亚马逊Item-based推荐系统
推荐系统从零单排系列(二)--Item-Based协同过滤算法
推荐系统从零单排系列(一)--Deep Neural Network for YouTube Recommendations