第六章（part 1）词嵌入概念与方法

1.1 词嵌入概念

1.2 主要词嵌入方法原理

1.3 基于Gensim的调用

回顾：词袋模型

基本词袋模型：基于频率统计，包括TF-IDF、n-gram

- 文档表示成高维、稀疏向量，样本稀疏，导致“维度灾难”
- 词袋模型的局限性：忽略了语义、结构、顺序、以及上下文信息

比如：

句1：”a clustering algorithm is used to group documents”;
句2：“text dataset is categorized by an unsupervised approach”
句3：“a dog is chasing a black cat”

句1和句2的意思非常接近。
但由于用了不同的词来表达，基于词袋模型句1和其他两句话的相似度都为0。

原因：每个词独立地对应一个维度，不能表示词和词之间的相关性（语义、上下文）。

词嵌入（word embedding）

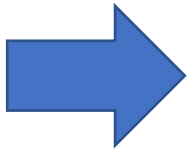
词表征学习：把词表示成一个向量，每个分量对应一个潜在的概念。

特点：

1. 通过模型学习得到
2. 表示成低维、连续（稠密）的词向量，
3. 可以反映出词之间的语义、上下文的相关性

比如：

document: 0.1 0.3 0.5 0.6
Text: 0.2 0.2 0.6 0.5
dog: 0.03 0.02 0.1 0.01



“document” 和“text” 对各个概念的相关性分布很接近。而”dog”很不一样。

词嵌入

通过大型语料库的学习，得到的词嵌入向量可以描述词之间的不同关联性或相似性，包括：

句法关系，如

- V(small)-V(smallest) = V(big)-V(biggest) 形容词原型-最高级的关系
- V(quick)-V(quickly)=obvious-obviously 形容词-副词的关系

语义关系，如

- V(men)-V(king)= V(women)-V(queen) 男-女性别关系
- V(France)-V(Paris)=V(Germany)-V(Berlin) 城市-国家的隶属关系

可以描述语义细节的能力使得这些词向量可以被用于很多基本的自然语言处理应用，比如机器翻译。

基于统计 vs 基于预测

- 基于统计的方法，如隐语义检索Latent Semantic Indexing（LSI）
  基于词与其他词之间的共现频率(co-occurrence)得到词向量；

- 基于预测的方法，如基于神经网络的语言模型

以下论文给出了更多分析和对比：
Don’t count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors,” by Baroni et al

Word2Vec

中心词（target）和上下文(context words)之间的关联性

用窗口来得到上下文

Window size

比如： “the quick brown fox jumps over the **lazy** dog”,

- 以”lazy”作为中心词，当上下文窗口(context window) 大小为1时那么对应的上下文窗口内的词为“the”和”dog”。
- 以“fox”作为中心词，当上下文窗口(context window) 大小为2时那么对应的上下文窗口内的词为”quick”,“brown”,”jumps”和”over”。

[1]“Distributed Representations of Words and Phrases and their Compositionality” by Mikolov et al.

[2]“Efficient Estimation of Word Representations in Vector Space” by Mikolov et al

移动窗口，得到所有的{中心词-上下文}

小练习

假设window size =2，给出下面例子的所有(target-context words)。

格式：target-{context word1, context word2…}

To tackle the dataset shift problem

最开始几个词为中心词时，左边窗口，和最后几个词为中心词时的右边窗口为空，应该怎么处理?

用虚假词"PAD"代替。

所有(target-context words)
“To tackle the dataset shift problem”

Window-size=2

先对所有词进行索引，得到

(PAD,0), (to, 1), (tackle, 2), (the, 3),(dataset,4), (shift,5), (problem,6)

To-{PAD,PAD, tackle, the}

Tackle-{PAD, To, the dataset}

the-{to, tackle, dataset, shift}

dataset-{tackle, the, shift, problem}

shift-{the, dataset, problem, PAD}

problem-{dataset, shift, PAD, PAD}

1, {0,0,2,3}

2,{0,1,3,4}

3,{1,2,4,5}

4,{2,3,5,6}

5,{3,4,6,0}

6,(4,5,0,0)

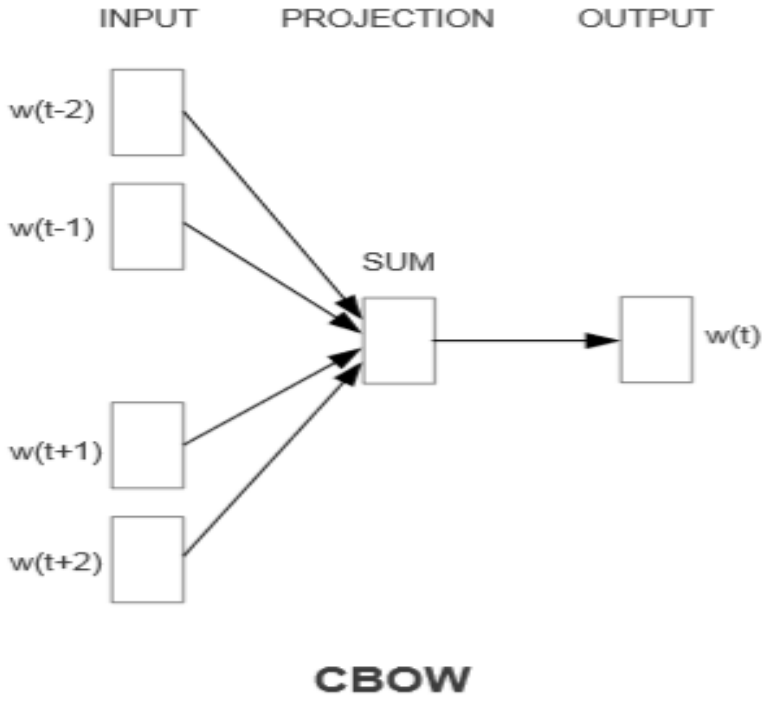
Word2Vec-两种方法

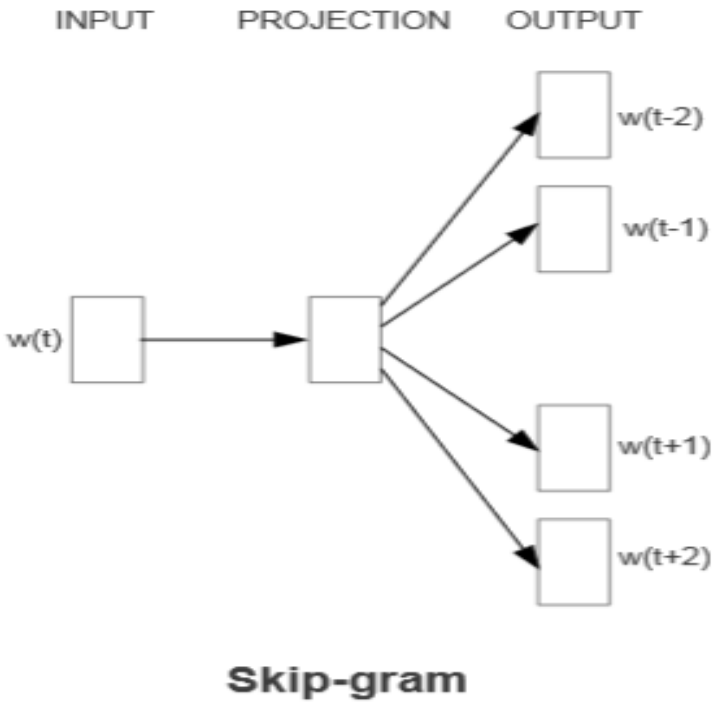
- Continuous Bag of Words (CBOW)

通过周边的词来预测中心词

Skip-gram

用中心词来预测周边词





[1]"Distributed Representations of Words and Phrases and their Compositionality" by Mikolov et al.

[2]"Efficient Estimation of Word Representations in Vector Space" by Mikolov et al

CBOW训练样本表示

建立（输入、输出）关系对，其中上下文作为输入/数据，中心词作为标签。

每个样本为：窗口内的2n 个context word（假设n为窗口大小），标签为对应的中心词。
每个context word表示为长度为vocab_size（字典大小） 的one-hot向量。

所以每个样本为一个矩阵$X_{n_c \times n_t}$，而不是一个向量。

"the quick brown fox jumps over the **lazy** dog"

假设"lazy","the","dog"三个词的索引/id分别是 7,1,8

构成的样本为：$X = \left[\quad \right]$

对应的标签 $y = [\quad]$

n_c :2×Window_size
n_t: vocab_size

分成多少个类别?

整个训练集表示成什么?

CBOW基本实现原理

n_c :2×Window_size
n_t: vocab_size
r: embed_dim

每个样本矩阵$X_{n_c \times n_t}$

↓ 降维(W)

$$H_{n_c \times r} = X_{n_c \times n_t} W_{n_t \times r}$$

到embed_dim维空间表示

↓ 取平均

$$\boldsymbol{h}_{r \times 1} = \sum_i H_{i \times r}$$

降维后的平均向量

↓ “升维“(G)

$$\hat{\boldsymbol{y}}_{n_t \times 1} = softmax(G_{n_t \times r} \boldsymbol{h}_{r \times 1})$$

vocab_size长度的向量，并转成概率分布

↓

计算中心词的预测与真实标签的差别-损失。

$$loss(\boldsymbol{y}_{n_t \times 1}, \hat{\boldsymbol{y}}_{n_t \times 1})$$

Word2Vec-两种方法

- Continuous Bag of Words (CBOW)

通过周边的词来预测中心词

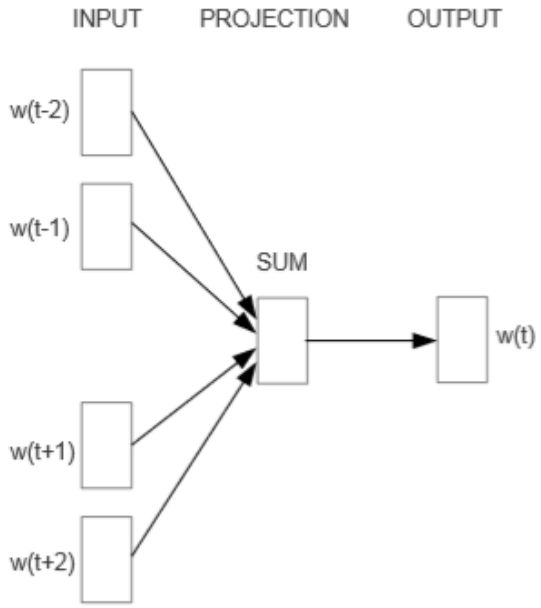
Skip-gram

用中心词来预测周边词

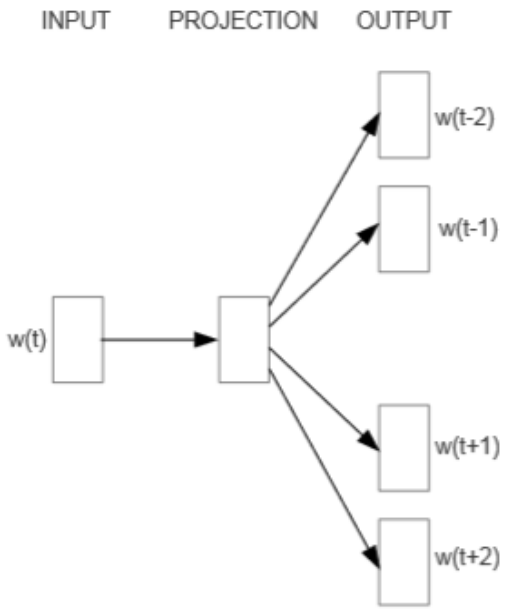
方法-CBOW:

窗口内的词在映射空间表示的平均得到中心词在该空间的映射。

取平均时并未考虑上下文中词的顺序。



CBOW



Skip-gram

- "Distributed Representations of Words and Phrases and their Compositionality" by Mikolov et al.
- "Efficient Estimation of Word Representations in Vector Space" by Mikolov et al

Skip-gram训练样本表示

根据(target, context _ words)，得到2*windowsize个由该中心词到**单个**上下文词组成的 (target, context) 对，作为正样本，表示relevant 。

“the quick brown fox jumps over the **lazy** dog”

假设”lazy”,”the”,”dog”三个词的索引/id分别是 7,1,8
构成2个正样本（”lazy”, “the”）和（”lazy”, “dog”）
构成的一个正样本为($\boldsymbol{x}_t$, $\boldsymbol{x}_{c1}$):

$x_t = [$ $]$

$x_{c1} = [$ $]$

对应的标签 $y = 1$

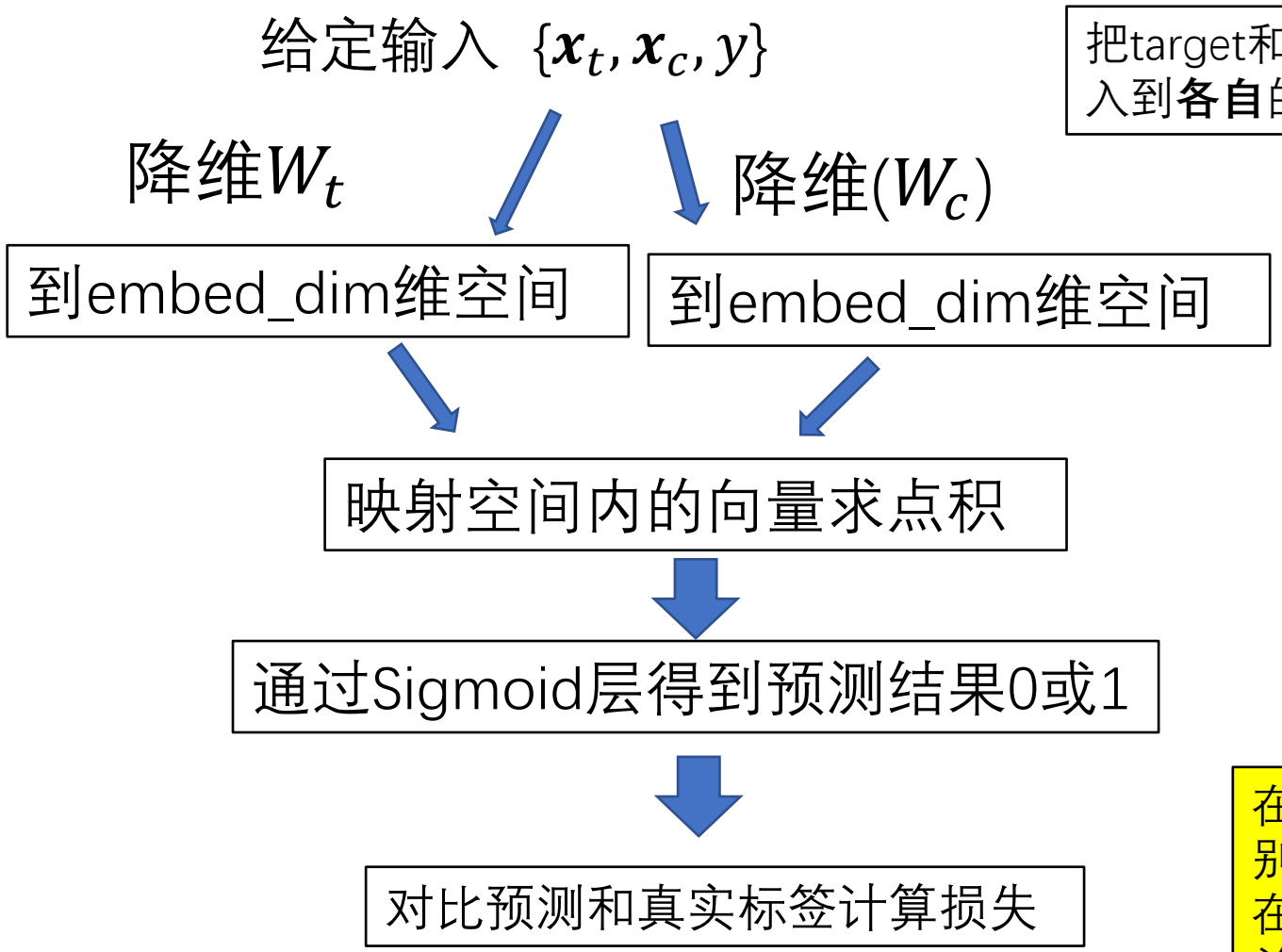
每个样本表示为:
$\{\boldsymbol{x}_t, \boldsymbol{x}_c, y\}$,
$\boldsymbol{x}_t$和$\boldsymbol{x}_t$为one-hot向量，分别对应target和context/random，
y=1或0。

分成多少个类别?

负样本如何采样？个数?

产生(target, random) 作为负样本（$y = 0$）表示不相关。这里random表示一个从词库中随机选择的词。

Skip-gram基本实现原理



把target和context/random分别输入到**各自**的embedding 层降维

$$\boldsymbol{h}_t = W_t x_t$$
$$\boldsymbol{h}_c = W_c x_c$$

$$\hat{y} = sigmoid(\boldsymbol{h}_t^T \boldsymbol{h}_c)$$

$$loss(y, \hat{y})$$

在CBOW中是多分类（类别数目等于词汇总数），在Skip-gram是二分类（相关、不相关）

CBOW和Skip-gram输入/输出对比

“the quick brown fox jumps over the **lazy** dog”

CBOW	Skip-gram
主要步骤：在embedding 空间对所有context words的表示取平均。 输入格式：X={context words}, y=target X={“the”, “dog”}, y=“lazy” 分别把X和y用one-hot向量表示 模型输出$\hat{y}$：和y维度相同的概率分布，每个类别对应一个中心词	主要步骤：在embedding 空间对target和context word的表示计算内积 输入格式：X={target, context}, y=1或0 X={“lazy”, “dog”}, y=1 分别把X中的target和context用one-hot向量表示。 模型输出$\hat{y}$：等于1的概率

小练习

假设window size =2，给出下面例子的CBOW和Skip-gram的训练样本集。

“To tackle the dataset shift problem”

索引：(PAD,0), (to, 1), (tackle, 2), (the, 3),(dataset,4), (shift,5), (problem,6)

To-{PAD,PAD, tackle, the}
Tackle-{PAD, To, the dataset}
the-{to, tackle, dataset, shift}
dataset-{tackle, the, shift, problem}　　对应索引：4,{2,3,5,6}
shift-{the, dataset, problem, PAD}
problem-{dataset, shift, PAD, PAD}

bible数据集

```
from nltk.corpus import gutenberg
from string import punctuation
bible = gutenberg.sents('bible-kjv.txt')
#只抽取第1000到2000行
bible = bible[1000:2000]
remove_terms = punctuation + '0123456789'
norm_bible = [[word.lower() for word in sent if word not in remove_terms] for sent in bible]
norm_bible = [' '.join(tok_sent) for tok_sent in norm_bible]
norm_bible = [tok_sent for tok_sent in norm_bible if len(tok_sent.split()) > 2]

print('Total lines:', len(bible))
print('\nSample line:', bible[10])
print('\nProcessed line:', norm_bible[10])
```

对bibile-kjv进行分句，以列表赋值。
(kjv: King James version)

```
Total lines: 1000

Sample line: ['36', ':', '24', 'And', 'these', 'are', 'the', 'children', 'of', 'Zibeon', ';', 'both', 'Ajah',
',', 'and', 'Anah', ':', 'this', 'was', 'that', 'Anah', 'that', 'found', 'the', 'mules', 'in', 'the', 'wilder
ness', ',', 'as', 'he', 'fed', 'the', 'asses', 'of', 'Zibeon', 'his', 'father', '.']

Processed line: 36 24 and these are the children of zibeon both ajah and anah this was that anah that found t
he mules in the wilderness as he fed the asses of zibeon his father
```

用Gensim实现Word2Vec(更多)

注意：最新版本的gensim某些方法的具体调用可能有不同，具体请按照官方文档说明。

```
import nltk
from gensim.models import word2vec
# tokenise sentences in corpus
wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in norm_bible]
# Set values for various parameters
feature_size = 100    # Word vector dimensionality
window_context = 30          # Context window size
min_word_count = 1    # Minimum word count
sample = 1e-3    # Downsample setting for frequent words

w2v_model = word2vec.Word2Vec(tokenized_corpus, size=feature_size,window=window_context,
                              min_count=min_word_count, sample=sample, sg=1, iter=50)
# view similar words based on gensim's model
similar_words = {search_term: [item[0]
                 for item in w2v_model.wv.most_similar([search_term], topn=5)]
                       for search_term in ['god', 'egypt', 'moses','famine']}
similar_words
```

sg=1表示skipgram，默认sg=0, 即CBOW

```
{'god': ['lord', 'thee', 'thy', 'waited', 'which'],
 'egypt': ['land', 'pharaoh', 'forth', 'bondage', 'out'],
 'moses': ['aaron', 'spake', 'lord', 'commanded', 'israel'],
 'famine': ['poverty', 'plenty', 'verified', 'morever', 'pasture']}
```

抽取bible前面5000行的结果

用Gensim实现Word2Vec

- 训练得到的词向量存在*model.wv*（*以KeyedVectors为数据类型*）

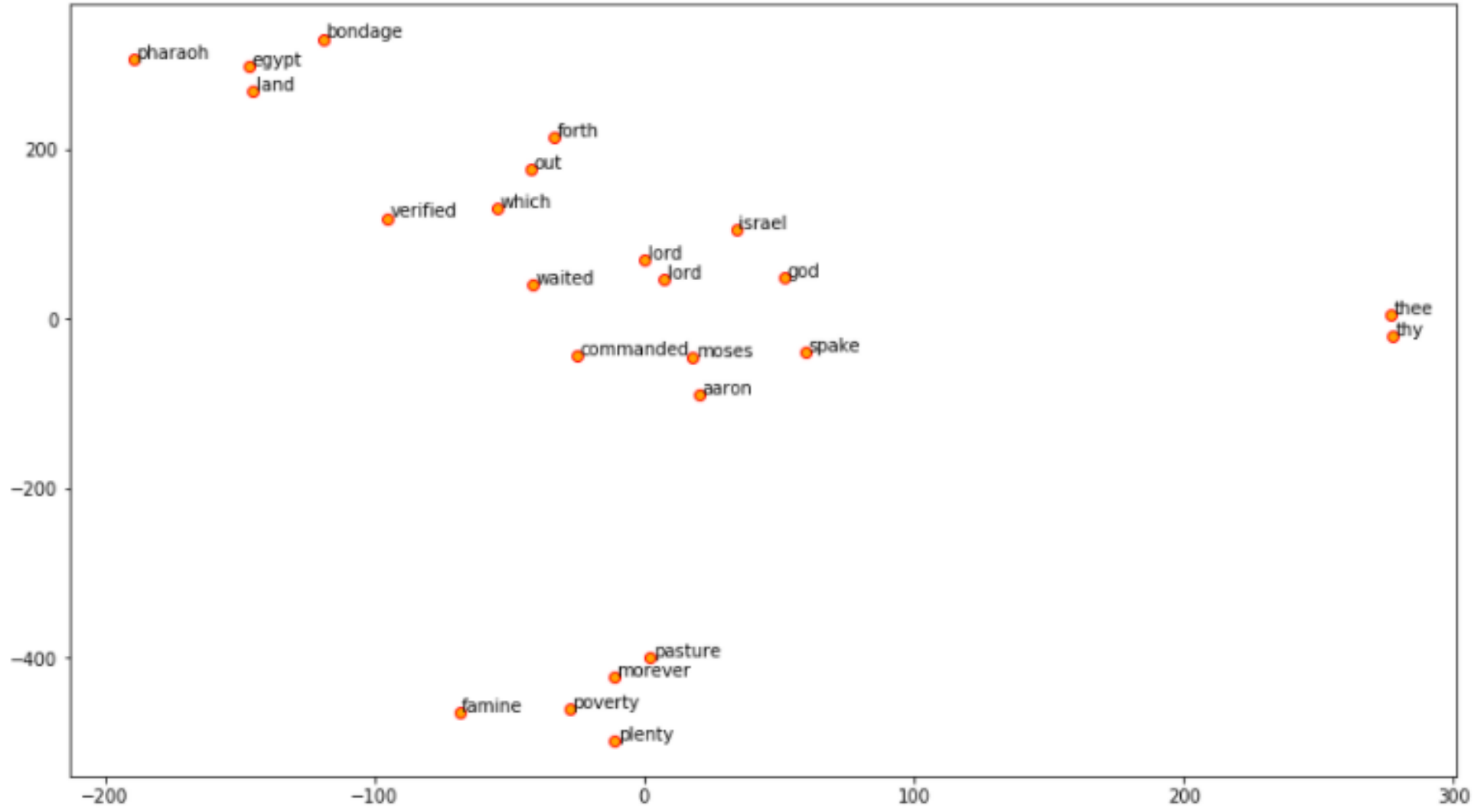
```
from gensim.models import KeyedVectors
# 把得到的词向量进行保存.
 word_vectors = model.wv
 word_vectors.save("word2vec.wordvectors")
# 从磁盘载入到内存.
wv = KeyedVectors.load("word2vec.wordvectors", mmap='r')
vector = wv['computer'] # 得到该词对应的 numpy类型向量
```

可视化词嵌入

```
from sklearn.manifold import TSNE
from matplotlib import pyplot as plt
words = sum([[k] + v for k, v in similar_words.items()], [])
wvs = w2v_model.wv[words]
tsne = TSNE(n_components=2, random_state=0, n_iter=10000, perplexity=2)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(wvs)
labels = words
plt.figure(figsize=(14, 8))
plt.scatter(T[:,0],T[:,1],c="orange",edgecolors="r")
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset points')
```

对前面的24个词（4个词以及它们各自最相似的5个）的词向量进行可视化。

可视化词嵌入


bondage
pharaoh
egypt
land
forth
out
which
verified
israel
lord
lord
god
waited
thee
thy
commanded
moses
spake
aaron
pasture
morever
famine
poverty
plenty
200
0
−200
−400
−200
−100
0
100
200
300

Glove

结合了全局矩阵分解（LSI）和局部上下文窗口（Skipgram、CBOW）

Download pre-trained word vectors

The links below contain word vectors obtained from the respective corpora. If you want word vectors trained on massive web datasets, you need only download one of these text files! Pre-trained word vectors are made available under the Public Domain Dedication and License.

- Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
- Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
- Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 300d vectors, 822 MB download): glove.6B.zip
- Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 200d vectors, 1.42 GB download): glove.twitter.27B.zip

基于word-word co-occurrence 矩阵D进行模型训练。其中每个元素dij表示word j出现在word i的上下文中的次数。

Glove: Global Vectors for Word Representation， EMNLP 2014

FastText github

- 原始的word2Vec把每个词作为独立的单位，只有出现在训练集才能学到对应的词向量。
- 把word表示成多个character n-gram，即以更小的单位来表示，然后对每个n-gram进行嵌入，得到对应向量表示；
- 一个词的表示只要对其包含的 n-grams 的向量进行相加或平均得到；
- 即使这个词没有出现在训练集，它的一部分n-gram可能出现在训练集中的其他词里面，因此就能得到该词的向量表示。

具体实现包含在：gensim.models.fasttext

```
from gensim.models.fasttext import FastText
ft_num_features = 100
ft_model = FastText(tokenized_train, size=ft_num_features, window=100,
                    min_count=2, sample=1e-3, sg=1, iter=50)
```

“Enriching Word Vectors with Subword Information” by Mikolov et al.

Gensim 中预训练好的词嵌入

```
>>> import gensim.downloader
>>> # Show all available models in gensim-data
>>> print(list(gensim.downloader.info()['models'].keys()))
['fasttext-wiki-news-subwords-300',
 'conceptnet-numberbatch-17-06-300',
 'word2vec-ruscorpora-300',
 'word2vec-google-news-300',
 'glove-wiki-gigaword-50',
 'glove-wiki-gigaword-100',
 'glove-wiki-gigaword-200',
 'glove-wiki-gigaword-300',
 'glove-twitter-25',
 'glove-twitter-50',
 'glove-twitter-100',
 'glove-twitter-200',
 '__testing_word2vec-matrix-synopsis']
```

Gensim 载入Glove预训练词嵌入

```
>>> # Download the "glove-twitter-25" embeddings
>>> glove_vectors = gensim.downloader.load('glove-twitter-25')
>>>
>>> # Use the downloaded vectors as usual:
>>> glove_vectors.most_similar('twitter')
[('facebook', 0.948005199432373),
 ('tweet', 0.9403423070907593),
 ('fb', 0.9342358708381653),
 ('instagram', 0.9104824066162109),
 ('chat', 0.8964964747428894),
 ('hashtag', 0.8859537333106995),
 ('tweets', 0.8878158330917358),
 ('tl', 0.8778461217880249),
 ('link', 0.8778210878372192),
 ('internet', 0.8753897547721863)]
```

应用

词向量->句子/文档向量

基于统计：

简单的平均

加权平均

基于语言模型：

输入到BERT等语言模型，进一步对词之间的关系、顺序等信息建模，得到句子/文档向量

词向量学习的重要启示

- 是一种无监督方法
- 虽然利用了监督学习框架，但语料库本身不包含标签信息
- 这种思想同样可以用到图像或其他数据类型上

- 基于预训练语言模型的迁移学习框架
- 在大语料库上训练好模型，在目标/下游任务上微调-突破计算资源、数据资源、隐私等局限性!