N-Grams (Predict based)

Language Models

- **Language Modeling** is the task of predicting the next word or character in a document.
- This technique can be used to train language models that can further be applied to a wide range of natural language tasks like text generation, text classification, and question answering.
- N-gram models are the simplest and most common kind of language model.

N-Grams 介紹

語句的表示方法(Representation)

- 對於一元模型（unigram）,每個詞都是獨立分布的

Ex. To, be, or, not, to, be

- 對於二元模型(bigram)，每個詞都與它左邊的最近的一個詞有關聯

EX. to be, be or, or not, not to, to be, …

- 三元模型(trigram): to be or, be or not, or not to, not to be

條件機率

- $P(A|B) = P(A,B)/P(A)$

$\Rightarrow P(A,B) = P(A|B)\,P(A)$

- $P(C|A,B) = P(A,B,C) / P(A,B) = P(A,B,C) / (\,P(B|A)\,P(A)\,)$

$\Rightarrow P(A,B,C) = P(A)\,P(B|A)\,P(C|A,B)$

- $P(A,B,C,D) = P(A)\,P(B|A)\,P(C|A,B)\,P(D|A,B,C)$

- $P(x_1,x_2,x_3,\ldots,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1,\ldots,x_{n-1})$

The Chain Rule in General

- $P(x1,x2,x3,\ldots,xn) = \prod^{n}_{i} P(x_n|x_1,\ldots,x_{n-1})$(累乘)

- P(W1,W2,..,Wn) = P(W1)P(W2|W1)...(Wn|Wn-1,...,W2,W1);

N-Grams 演算法

- N-Grams is a word prediction algorithm using probabilistic methods to predict next word after observing N-1 words.

貓, 跳上, 椅子

- Unigram:每個詞都是獨立分布的, 也就是對於P(A,B,C) 其中A,B,C互相之間沒有交集. 所以P(A,B,C) = P(A)P(B)P(C)

P(A="貓", B="跳上", C="椅子") = P("貓")P(“跳上”)P("椅子");

貓, 跳上, 椅子

- 對於二元模型, 每個詞都與它左邊的最近的一個詞有關聯, 也就是對於P(A,B,C) = P(A)P(B|A)P(C|B)

- P(A="貓", B="跳上", C="椅子") = P("貓")P(“跳上”|“貓”)P("椅子"|“跳上”)

ngram model

Unigram model: $$P(w^{(1)} \ldots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)})$$

Bigram model: $$P(w^{(1)} \ldots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \mid w^{(i-1)})$$

Trigram model: $$P(w^{(1)} \ldots w^{(N)}) = \prod_{i=1}^{N} P(w^{(i)} \mid w^{(i-1)}, w^{(i-2)})$$

Markov Assumption

- Simplifying assumption: (bigram)

Andrei Markov

$$P(w_n | w_{1:n-1}) \approx P(w_n | w_{n-1})$$

Wikimedia commons

Estimating bigram probabilities

- The Maximum Likelihood Estimate

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

An example (bigram)

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

- <S> =

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $\quad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $\quad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $\quad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $\quad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

應用-判斷句子組成

- 假設現在有一個語料庫，我們統計了下面的一些詞出現的數量

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

P(i|<s>)=0.25 p(Chinese|want)=0.0011
P(food|Chinese)=0-5 p(</s>|food) = 0.68
p(want|<s>) = 0.25

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

應用-判斷句子組成

- 例如，其中第一行，第二列 表示給定前一個詞是 “i” 時，當前詞為“want”的情況一共出現了827次。據此，我們便可以算得相應的頻率分佈表如下。

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

應用-判斷句子組成

- S1= “<s> I want chinese food</s>”
- s2 = "<s> want i chinese food</s>”

哪個句子更合理

P(s1)=P(i|<s>)P(want|i)P(chinese|want)P(food|chinese)P(</s>|food)

=0.25×0.33×0.0011×0.5×0.68=0.000031

P(s2)=P(want|<s>)P(i|want)P(chinese|want)P(food|chinese)P(</s>|food)

=0.25*0.0022*0.0011*0.5*0.68 = 0.00000002057

P(s1) > p(s2) => S1較合理

Python n-gram

```
from nltk.util import ngrams
nltk.download('punkt')
textfile = 'there is an apple on the desk. this is an apple'

tokens = nltk.word_tokenize(textfile)
bgs = ngrams(tokens, 3)     #tri-gram
#compute frequency distribution for all the bigrams in the text

fdist = nltk.FreqDist(bgs)
for first,second in fdist.items():
    print (first,second)
```

results

('there', 'is', 'an') 1

('is', 'an', 'apple') 2

('an', 'apple', 'on') 1

('apple', 'on', 'the') 1

('on', 'the', 'desk') 1

('the', 'desk', '.') 1

('desk', '.', 'this') 1

('.', 'this', 'is') 1

('this', 'is', 'an') 1

應用

Google
愛情
愛情公寓
愛情摩天輪
愛情語錄
愛情進化論
愛情
愛情的溫度
愛情小說
愛情電影
愛情轉移
愛情白皮書

Skip n gram

- A k-skip-n-gram is a length-n subsequence where the components occur at distance at most k from each other.
- 從一個文字來預測上下文
- It provides one way of overcoming the data sparsity problem found with conventional n-gram analysis.

Ex 1-skip-2-grams

- the rain in Spain falls mainly on the plain

- the in, rain Spain, in falls, Spain mainly, falls on, mainly the, and on plain.

Bag of word (詞袋)

- Vector space representation

1. the dog saw a cat
2. the dog chased the cat,
3. the cat climbed a tree.

Vector: ("the", "dog", "saw", "a", "cat", "chased", "climbed", "tree").

["", "dog", "", "", "", "", "", ""] ->

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

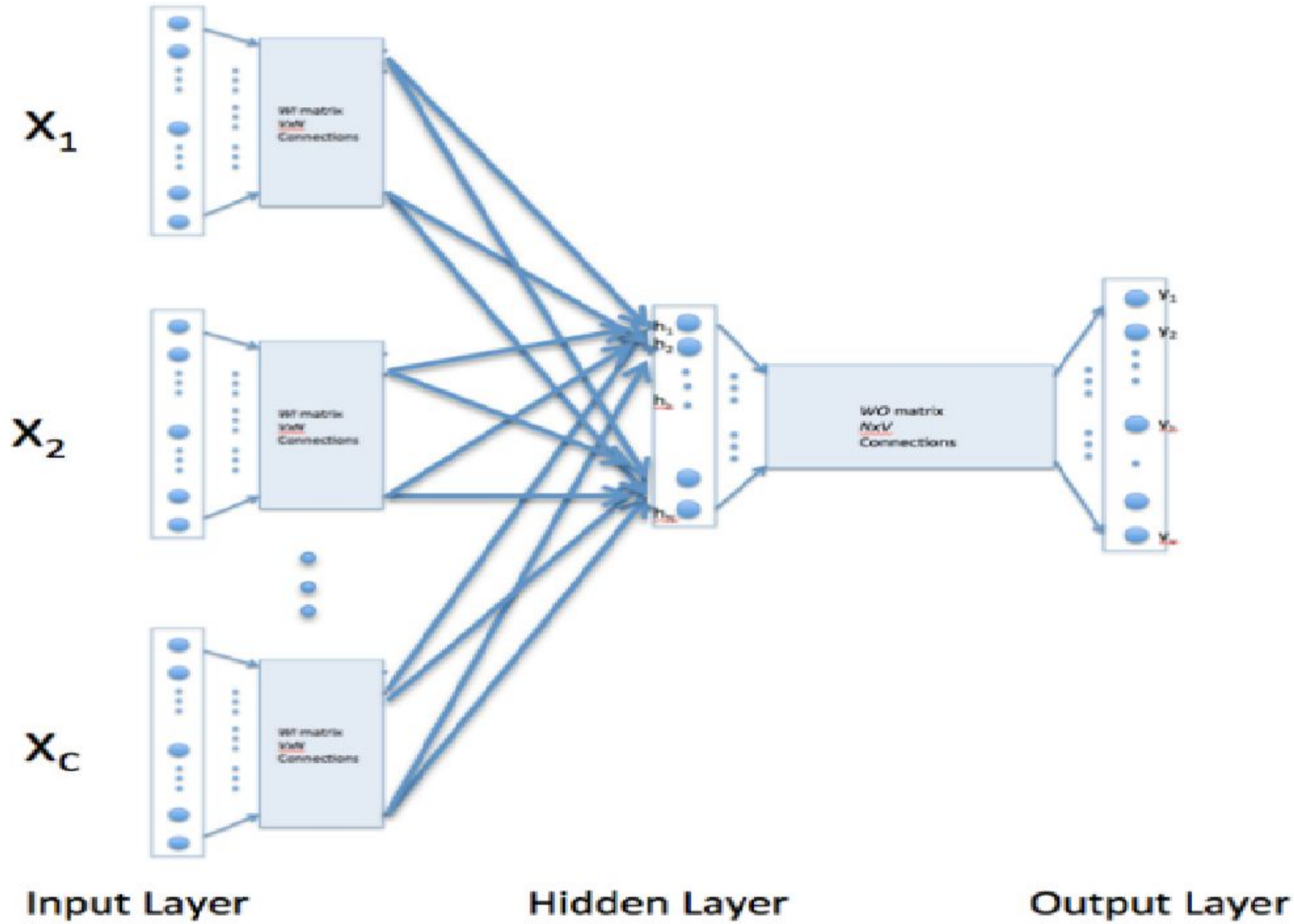
Continuous bag of word (CBOG)

- 已知目標詞的上下文，來預測目標詞；

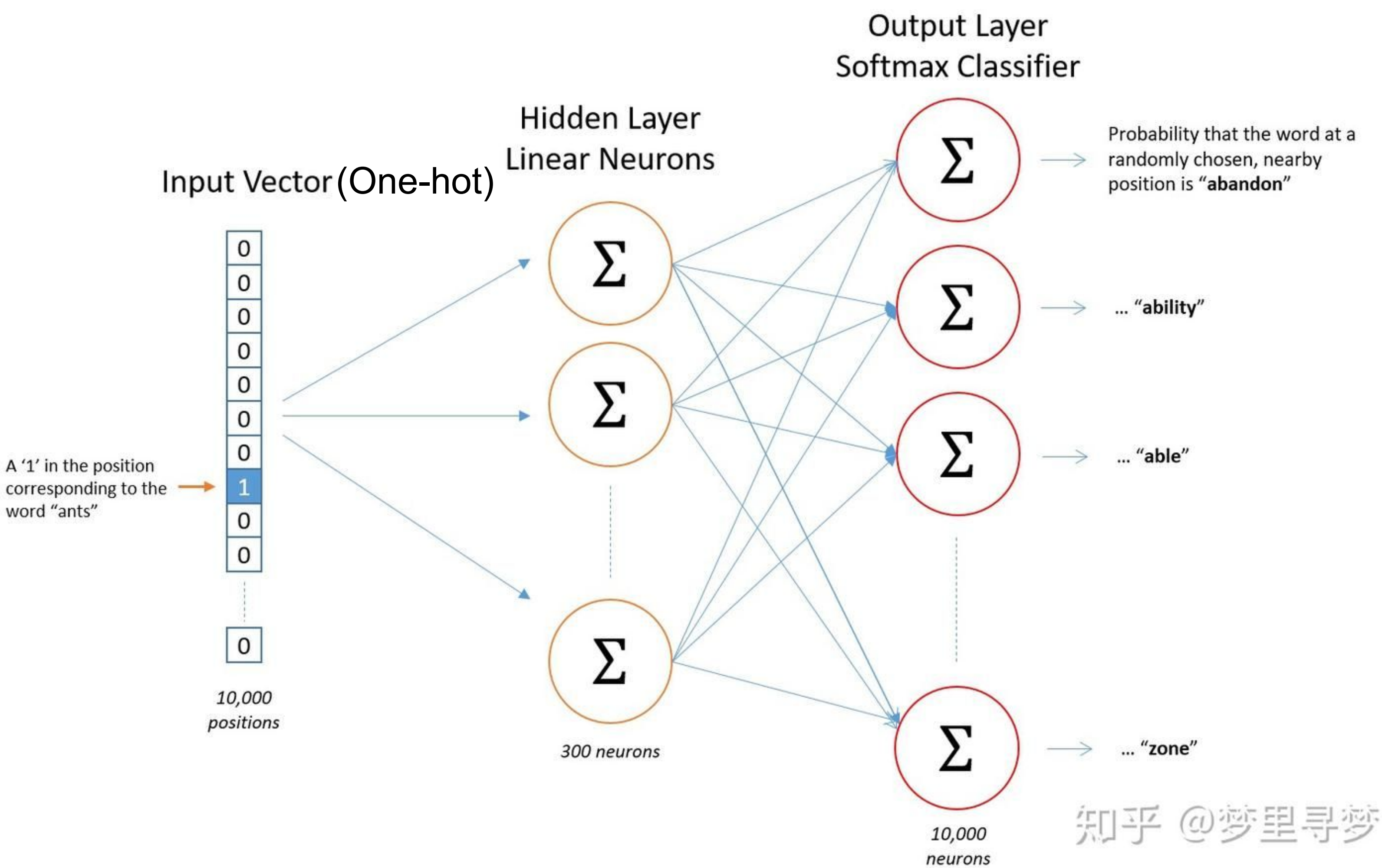
- the dog “_” the cat
- Input “the”, “dog”, “the”, “cat”
- Find “chase”

- Inverse of skip-n-gram

Continuous bag of word (CBOW)

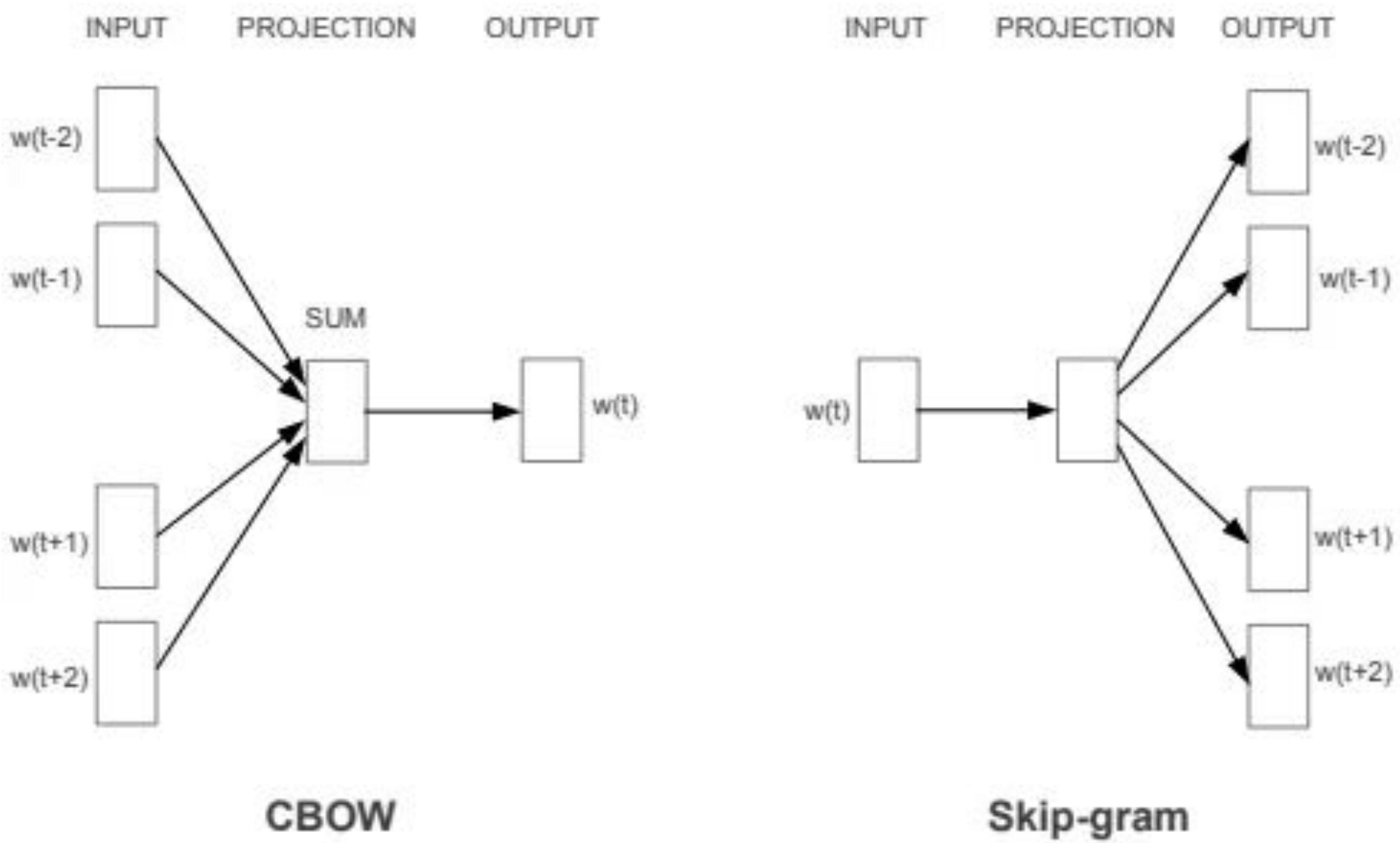


CBOW



Output Layer
Softmax Classifier
Hidden Layer
Linear Neurons
Input Vector (One-hot)
Probability that the word at a randomly chosen, nearby position is "**abandon**"
... "**ability**"
... "**able**"
... "**zone**"
A '1' in the position corresponding to the word "ants"
0
0
0
0
0
0
0
1
0
0
0
10,000 positions
300 neurons
10,000 neurons
知乎 @梦里寻梦

Skip-gram


INPUT
PROJECTION
OUTPUT
w(t-2)
w(t-1)
SUM
w(t)
w(t+1)
w(t+2)
CBOW
INPUT
PROJECTION
OUTPUT
w(t)
w(t-2)
w(t-1)
w(t+1)
w(t+2)
Skip-gram

word2vec

- 將詞語映射成一個固定維度的向量，節省空間。
- 2）詞向量可能會具備一定的語義信息，將相似的詞語放到相近的向量空間（比如香蕉和蘋果都是屬於水果，蘋果又會涉及到歧義問題），可以學習到詞語之間的關係（比如經典的男人-女人=國王-王后）。

Word2vec

- Google
- 依靠了 skip-gram 與 Continuous Bag of Word (CBOW) 的方法來實作

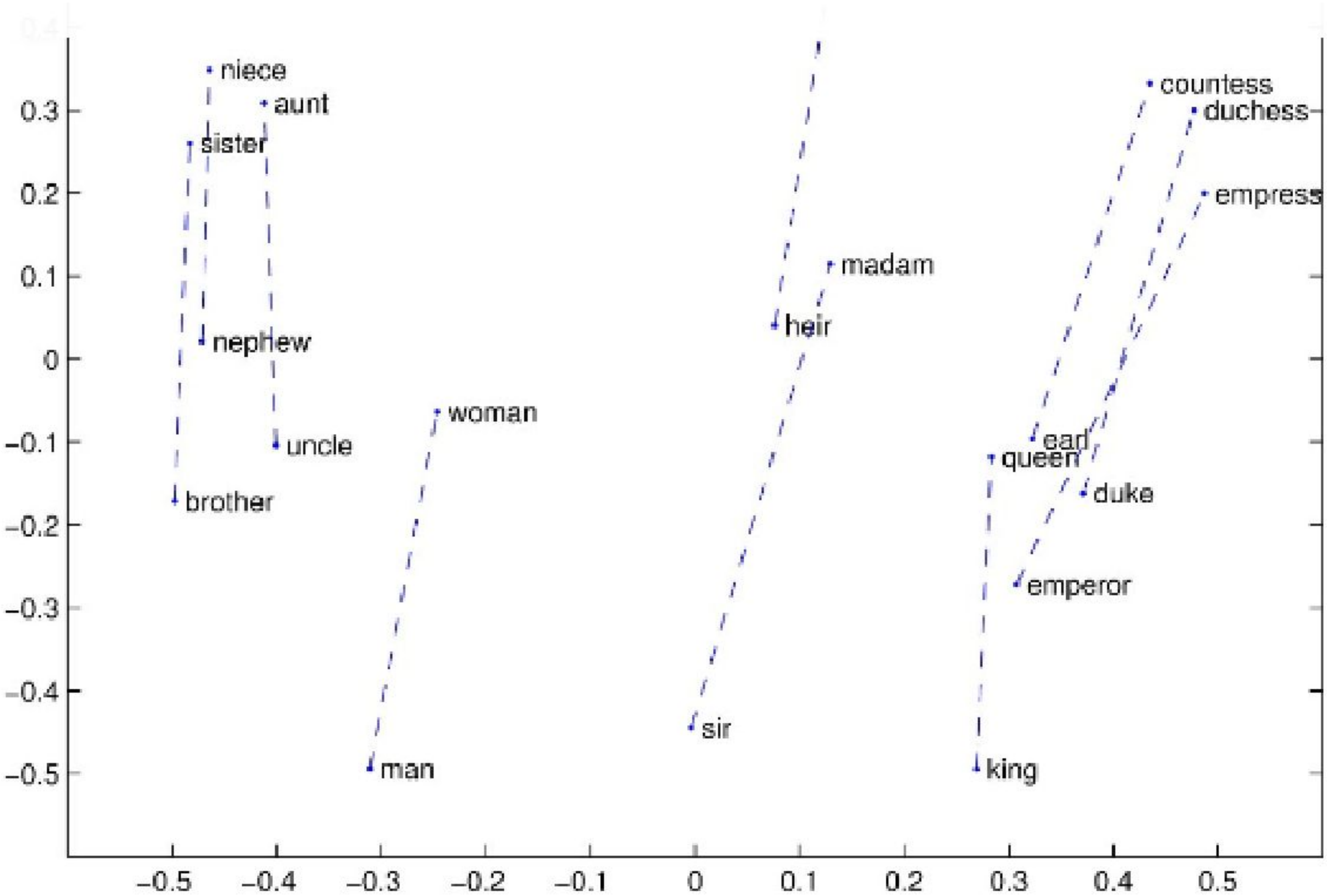
- 核心是一個極為淺層的類神經網路

- 來訓練出含有每個字詞語義的字詞向量

Word2vec

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

Word2vec



图片来自 Stanfordnlp/GloVe

word2vec

```
from gensim.models import Word2Vec
sentences = [['this', 'is', 'the', 'first', 'sentence', 'for', 'word2vec'],
['this', 'is', 'the', 'second', 'sentence'],
['yet', 'another', 'sentence'],
['one', 'more', 'sentence'],
['and', 'the', 'final', 'sentence']]
# train model
model = Word2Vec(sentences, min_count=1)
# summarize the loaded model
print(model)
# summarize vocabulary
words = list(model.wv.vocab)
print(words)
# access vector for one word
print(model['sentence'])
# save model
model.save('model.bin')
# load model
```

- word2Vec(vocab=14, size=100, alpha=0.025) ['this', 'is', 'the', 'first', 'sentence', 'for', 'word2vec', 'second', 'yet', 'another', 'one', 'more', 'and', 'final'] [7.03078404e-04 -3.06523964e-03 8.38766922e-04 -3.42155388e-03 1.11275294e-03 -2.00818293e-03 2.82139680e-03 4.68324590e-03 -4.99570230e-03 -4.05243691e-03 3.54941934e-03 1.82797853e-03 -2.66507780e-03 -4.14388115e-03 1.31099485e-03 2.36437470e-03 6.91659341e-04 1.14680477e-03 -2.66113988e-04 -2.80059721e-05 -3.66883446e-03 8.19117238e-04 -5.66632545e-04 3.41541832e-04 2.93611060e-03 -1.75147678e-03 1.34062849e-03 4.10531508e-03 -3.09920841e-04 -3.36961634e-03 -3.30118742e-03 -4.03716043e-03 -1.00748068e-04 -7.18949595e-05 4.32796776e-03 -1.23059115e-04 -2.00851914e-03 1.16727990e-03 -3.55407386e-03 -3.76890908e-04 9.49354551e-04 4.12891665e-03 -4.55296552e-03 -3.37862666e-03 1.61578774e-03 3.97557812e-03 3.92386550e-03 -2.33172602e-03 -1.47368643e-03 -1.00871117e-03 -4.42584604e-03 -2.48288561e-04 6.55699812e-04 2.61046691e-03 3.63694923e-03 -4.83081676e-03 -1.40730327e-03 -2.92131072e-03 5.25753887e-04 6.26921188e-04 -4.71570902e-03 -4.56014750e-05 7.26238824e-04 4.83909156e-03 -3.31999431e-03 -8.77807324e-04 4.71923413e-04 4.02569817e-03 -1.67337607e-03 -1.46528066e-03 -4.94623138e-03 -1.52511254e-03 1.39585952e-03 -2.25825910e-03 -4.07036860e-04 1.10396487e-03 -2.73216097e-03 2.98726908e-03 1.18497264e-04 4.99487342e-03 -2.97104404e-03 1.49209716e-03 3.96911753e-03 -4.13467688e-03 4.63157566e-03 -2.25937692e-03 -4.12324630e-03 -3.86923319e-04 4.88799484e-03 1.83994370e-03 -4.65374254e-03 -5.91134420e-04 -3.94692505e-03 1.01323210e-04 -1.35891209e-03 3.92786569e-05 -2.13225861e-03 -4.80551505e-03 2.92222132e-03 -1.62681786e-03]

GloVe

- Stanford university
- 預先訓練好的 word vector
- 可以用 300 維的向量來表示兩百二十萬個字詞
- GloVe is based on global word co-occurrence statistics
- 可以有效解決上述的維度爆炸問題，節省了大量的運算及儲存成本。

其他

- Doc2vec
- Fasttext (facebook)

應用:情感分析

```
from textblob import TextBlob #使用textBlob
#要分析的句子
text = "I am happy today. I feel sad today."
blob = TextBlob(text)
#print 第一個句子之情感, 主觀性
#-1表最負面, 1表最正面 “I am happy today”
print(blob.sentences[0].sentiment)
#print 第2個句子之情感, 主觀性 “I feel sad today"
print(blob.sentences[1].sentiment)
```

results

- Sentiment(polarity=0.8, subjectivity=1.0)
- Sentiment(polarity=-0.5, subjectivity=1.0)

中文情感分析

```
!pip install snowNLP                    #在colab安裝snowNLP
from snownlp import SnowNLP
#u代表文本的編碼是Unicode
text = u"我今天很快樂。我今天很憤怒。"
s=SnowNLP(text)
# 用sententces方法將text斷句:
for sentence in s.sentences:
  print(sentence)
s1 = SnowNLP(s.sentences[0])      #第一個句子
print(s1.sentiments)               #print 第一個句子之情感
s1 = SnowNLP(s.sentences[1])
print(s1.sentiments)
```

results

我今天很快樂

我今天很憤怒

0.9268071116367116 (越接近1表示越正面)

0.17026607625759916 (越接近0表示偏向負面)