Word2Vec

Syllabus

- Word2Vec Overview
- NNLM(Neural Network Language model)
- CBOW/Skip-gram
- Hierarchical Softmax and Negative Sampling

Document Representation: Bag of Words

John likes to watch movies. Mary likes too.

John also likes to watch football games.

Dictionary

{"John": 1, "likes": 2, "to": 3, "watch": 4, "movies": 5, "also": 6, "football": 7, "games": 8, "Mary": 9, "too": 10}

One-Hot

John: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

likes: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

...

too : [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

- There are 10 words in dictionary, each word corresponds to 1 **index**
- **No correlation** between the order of words in dictionary and the order in sentence

Document Representation: Bag of Words

- The document can be represented as **the sum of the vector of each word**

John likes to watch movies. Mary likes too. ➡ [1, 2, 1, 1, 1, 0, 0, 0, 1, 1]

John also likes to watch football games. ➡ [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]

- The weight of word

The **order of word** in document is not considered

- TF-IDF (Term Frequency - Inverse Document Frequency)

IDF weight $\log(1 + \frac{N}{n_t})$

Information Retrieval

[0.693, 1.386, 0.693, 0.693, 1.099, 0, 0, 0, 0.693, 0.693]

Document Representation: Bi-gram and N-gram

Index of Bi-gram

"John likes" : 1,
"likes to" : 2,
"to watch" : 3,
"watch movies" : 4,
"Mary likes" : 5,
"likes too" : 6,
"John also" : 7,
"also likes" : 8,
"watch football": 9,
"football games": 10,

John likes to watch movies. Mary likes too.

↓

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

John also likes to watch football games.

↓

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

n	Feature number
1 (unigram)	2×10^5
2 (bigram)	4×10^{10}
3 (trigram)	8×10^{15}
4 (4-gram)	16×10^{20}

Pro: Consider the order of words
Cons: Expansion of features

Bag of Words and N-gram Problem

- There is no way to see the relationship between the vectors of words

$[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$ **Drama**

$[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$ **Play**

$[0, 0, 0, 0, 0, 0, 0, 0, 1, 0]$ **Performance**

Too sparse, difficult to capture the meaning of document

- The dimension of features increases with the corpus increasing
- The dimension of features increases dramatically with the corpus increasing (N-gram)
- Sparseness problem

Bag of Words and N-gram Problem

There is no way to see the relationship between the vectors of words

Nearest words to
frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

litoria

leptodactylidae

rana

eleutherodactylus

Intuition of Word2Vec

"You shall know a word by the company it keeps"

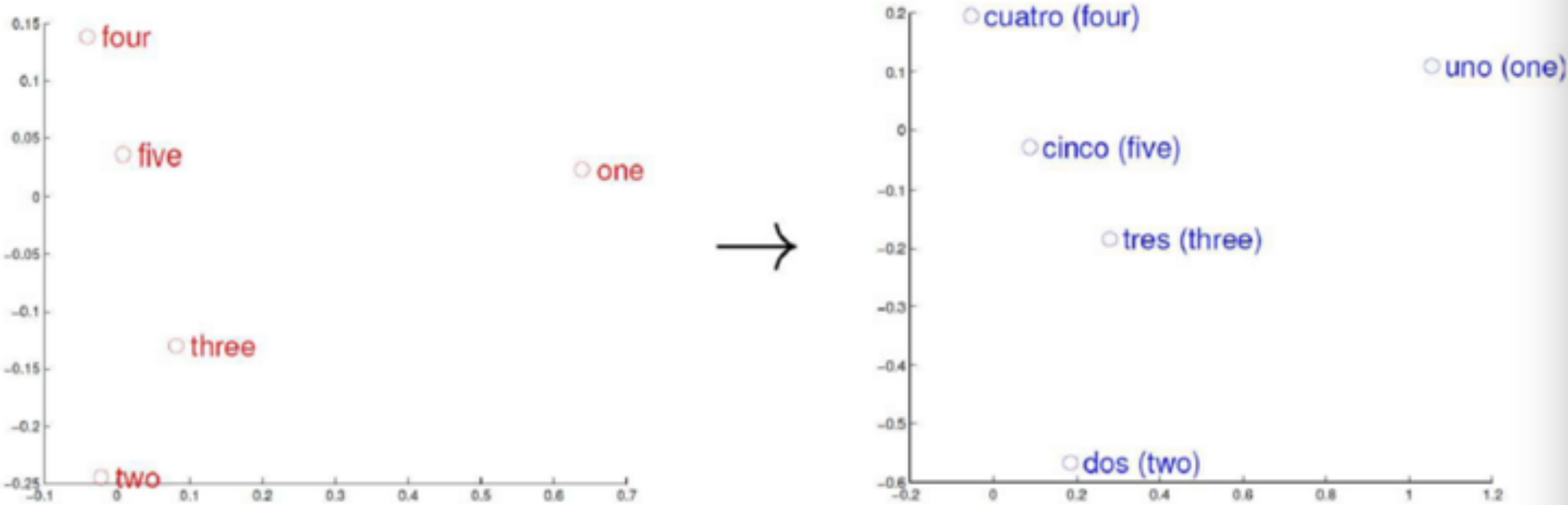
(J. R. Firth 1957: 11)

government debt problems turning into banking crises as has happened in

saying that Europe needs unified banking regulation to replace the hodgepodge

Word2Vec Overview

The similarity of vector distribution



Word2Vec Overview

Final target: Word2Vec can be used as input for machine learning/ deep learning model

$$V_{King} - V_{Queen} + V_{Women} = V_{Man}$$

$$V_{Paris} - V_{France} + V_{German} = V_{Berlin}$$

NNLM (Neural Network Language model)

- The optimisation process of language model ->
  the process of generate Word2Vec

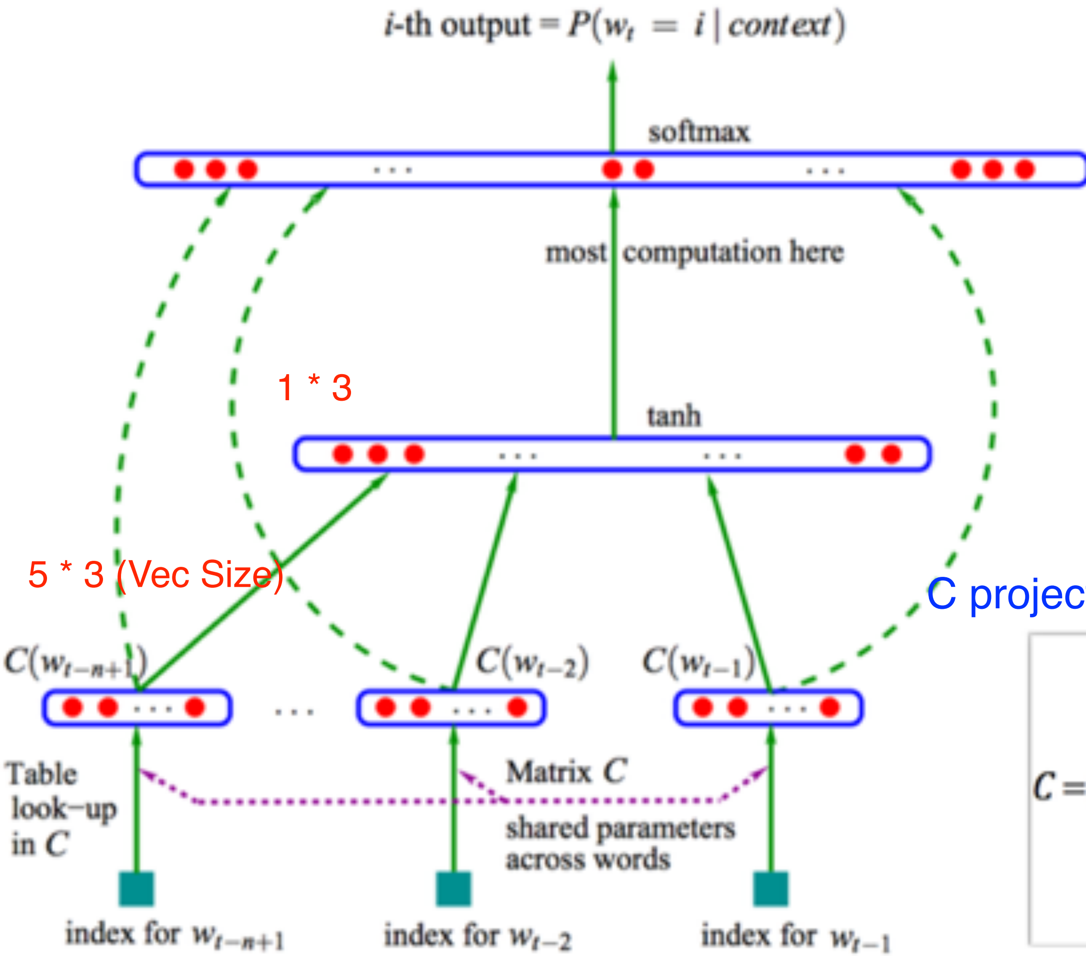
Objective Function $L(\theta) = \sum_t \log P(w_t | w_{t-n+1}, \dots w_{t-1}).$

- Use asymmetric sliding window (size = n-1)
- Sum the probabilities using sliding window based on each word in corpus
- The probabilities need to fitful the normalization formulas:

$$\sum_{w \in \{vocabulary\}} P(w | w_{t-n+1}, \cdots, w_{t-1}) = 1$$

Use neural network to get the probability

NNLM Structure



1* 5 (Dictionary Size)

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

(n-1) preceding words: One-hot

Use linear projection to present word by dense vector

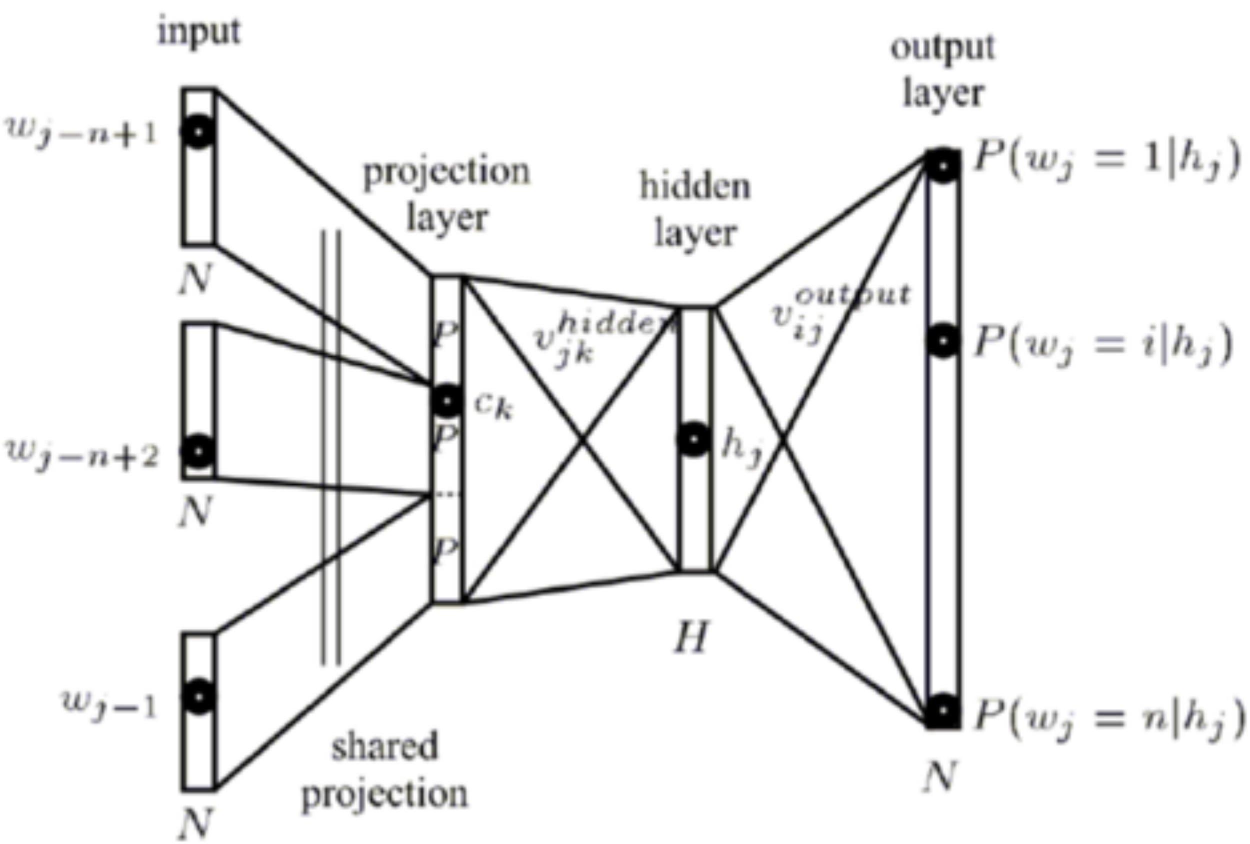
Output layor: Softmax

Weight optimization: BP + SGD

C projection matrix: dense representation of words

$$C = (w_1, w_2, \cdots, w_V) = \begin{pmatrix} (w_1)_1 & (w_2)_1 & \cdots & (w_V)_1 \\ (w_1)_2 & (w_1)_2 & \cdots & (w_V)_2 \\ \vdots & \vdots & \ddots & \vdots \\ (w_1)_D & (w_1)_D & \cdots & (w_V)_D \end{pmatrix}$$

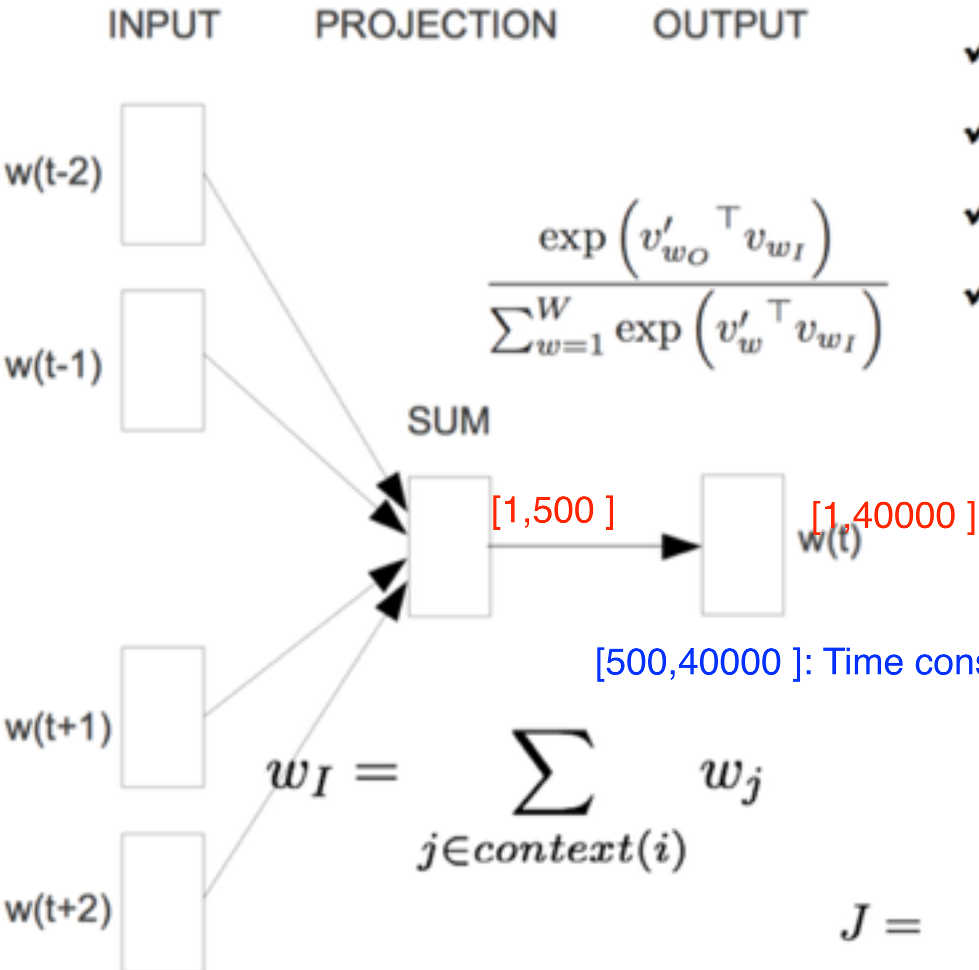
NNLM Structure



Calculation effort for each word: $N * D + N * D * H + H * V$

Word2vec: CBOW

- ✓ **No hidden layer**
- ✓ **Uses bi-directional contextual window**
- ✓ **No impacted by the orders of context words (BoW)**
- ✓ **Input layor: One-hot**
- ✓ **Projection layor: Sum of the vector**



$$\frac{\exp\left({v'_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^{\top} v_{w_I}\right)}$$

[500,40000]: Time consuming

$$w_I = \sum_{j \in context(i)} w_j$$

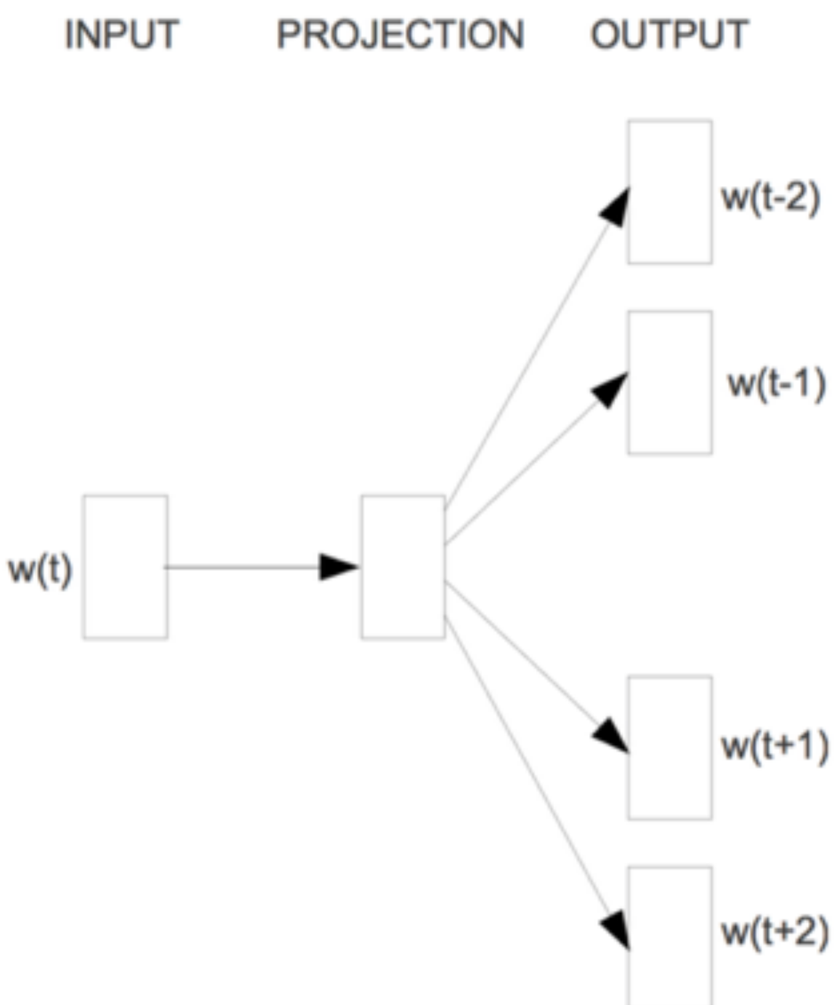
Objective Function

$$J = \sum_{w \in corpus} P(w|context(w))$$

$$J = \sum_{i \in corpus} log\left(\frac{exp(w_i^T \widetilde{w}_I)}{\sum_{k=1}^{V} exp(w_i^T \widetilde{w}_k)}\right)$$

$$J = \sum_{i \in corpus, j \in context(i)} log\left(\frac{exp(w_i^T \widetilde{w}_j)}{\sum_{k=1}^{V} exp(w_i^T \widetilde{w}_k)}\right)$$

Word2vec: Skip-Gram



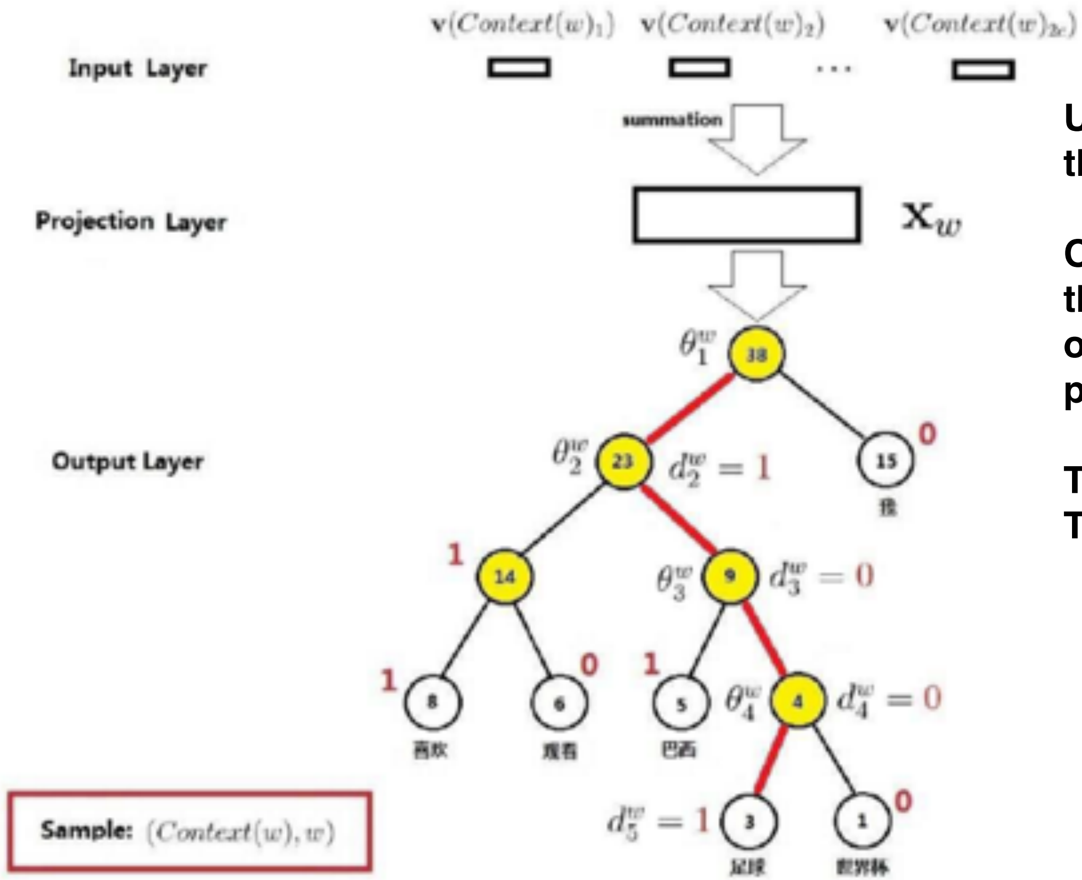
- **No hidden layer**
- **Projection mayor can be ignore**
- **The vector of each word can be input as log-linear**

Objective Function

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} log(p(w_{t+j}|w_t)$$

$$p(w_k|w_t) = \frac{exp(\widetilde{w}_k^T w_t)}{\sum_{m=1}^{V} exp(\widetilde{w}_m^T w_t)}$$

Optimisation Approach: Hierarchical Softmax



Use Huffman Tree to code the dictionary for output words

Only calculate the path (Non-leaf node): the word contribution (Binary Classificatio on each node on path, the final prob = seve prob can be shared.)

The calculation cost : The depth of tree log2(V)

Optimisation Approach: Hierarchical Softmax

Choosing Function

$$p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma\left([\![n(w, j+1) = \mathrm{ch}(n(w,j))]\!] \cdot {v'_{n(w,j)}}^{\top} v_{w_I} \right)$$

- **Sigmoid** $\sigma(x) = 1/(1+\exp(-x))$

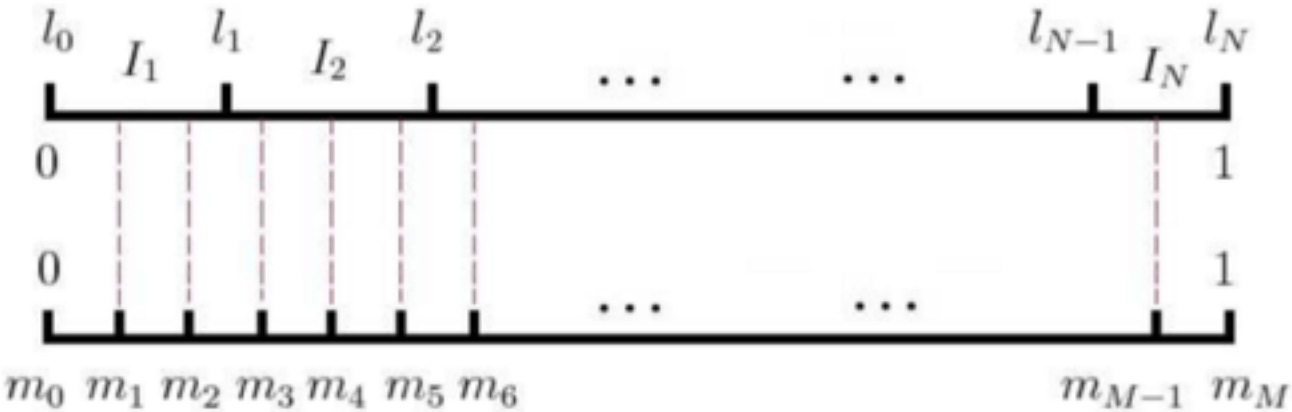
$$\sum_{w=1}^{W} p(w|w_I) = 1$$

Optimisation Approach: Negative Sampling

P(w|context(w)) : One positive sample, v-1 negative samples, sampling on negative samples

$$len(w) = \frac{\text{counter}(w)}{\sum_{u \in \mathcal{D}} \text{counter}(u)},$$

Best Value: counter(w)^(3/4)



[0,1] is divided into M= 10^8 part, a random number between [1,M-1] is generated each time for word choosing

Word2Vec: Problem

- Train based on local context window without using the information contained in global c
  currence matrix. **Solution: GLoVe**

- Unable to represent polysemous words as there is one vector of word.
  Solution: sense2Vec

Word2Vec Implementation: gensim

gensim
topic modelling for humans

Get Expert Help
- machine learning, NLP, data mining
- custom SW design, development, optimizations
- corporate trainings & IT consulting

Home | Tutorials | Install | Support | API | About

models.word2vec – Deep learning with word2vec

Initialize a model with e.g.:

```
>>> model = Word2Vec(sentences, size=100, window=5, min_count=5, workers=4)
```

Persist a model to disk with:

```
>>> model.save(fname)
>>> model = Word2Vec.load(fname)  # you can continue training with the loaded model!
```

The model can also be instantiated from an existing file on disk in the word2vec C format:

```
>>> model = Word2Vec.load_word2vec_format('/tmp/vectors.txt', binary=False)  # C text format
>>> model = Word2Vec.load_word2vec_format('/tmp/vectors.bin', binary=True)  # C binary format
```

You can perform various syntactic/semantic NLP word tasks with the model. Some of them are already built-in:

```
>>> model.most_similar(positive=['woman', 'king'], negative=['man'])
[('queen', 0.50882536), ...]

>>> model.doesnt_match("breakfast cereal dinner lunch".split())
'cereal'

>>> model.similarity('woman', 'man')
0.73723527

>>> model['computer']  # raw numpy vector of a word
array([-0.00449447, -0.00310097,  0.02421786, ...], dtype=float32)
```