word2vec

Abstract

- 1. what is word2vec?
- 2. previous work
- 3. architectures of Mikolov’s
- 4. result and comparison
- 5. Hyper parameters and tricks
- 6. extensions

What is word2vec?

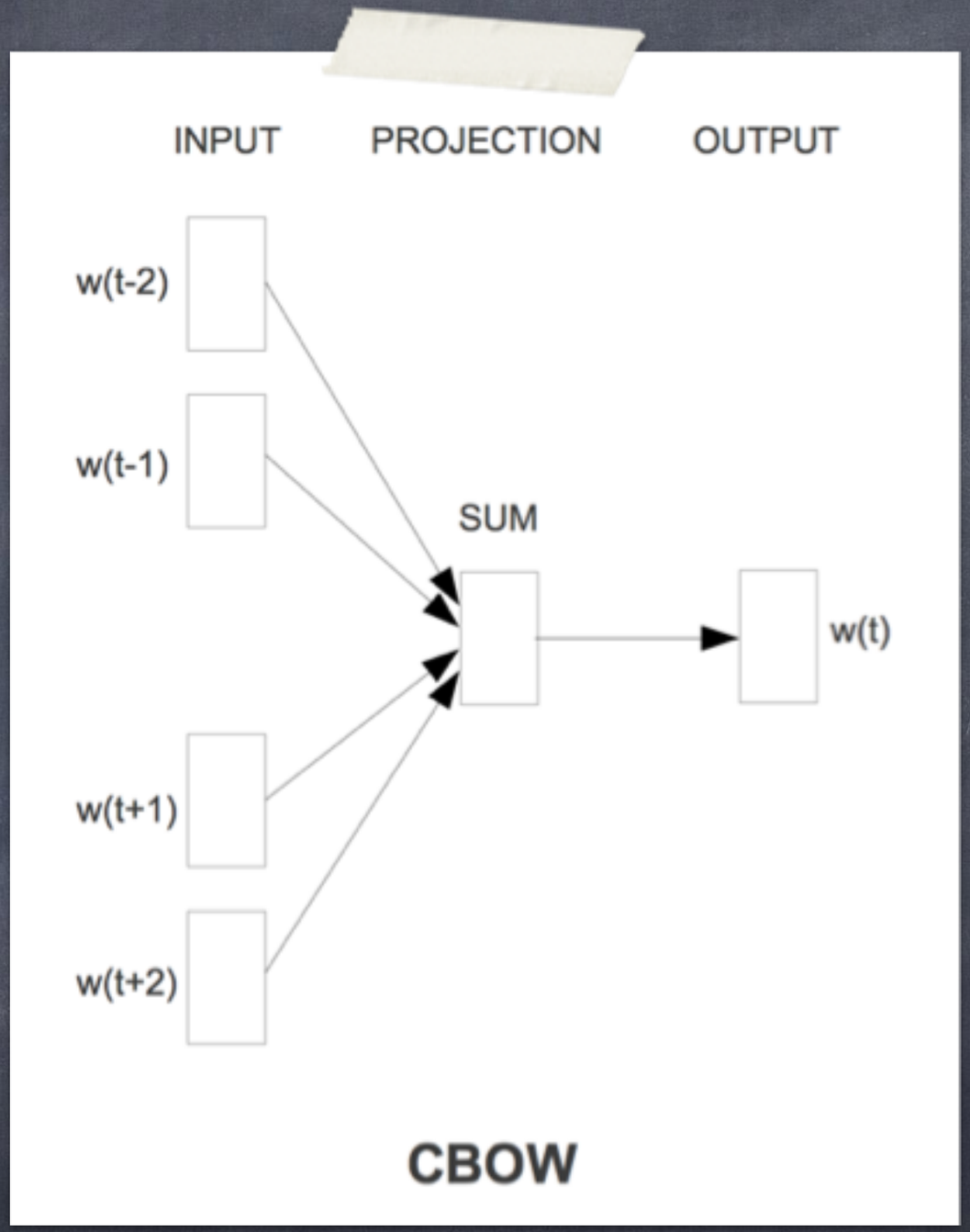
- 1. continuous vector representations of words
- 2. capture syntactic and semantic word similarities
- 3. state-of-art performance and high efficiency

previous work

- 1. Learning representations by back-propagating errors(Hinton, 1986)
- 2. NNLM(Bengio 2003)
- 3. RNNLM: time-delayed connections within hidden layer, short term memory

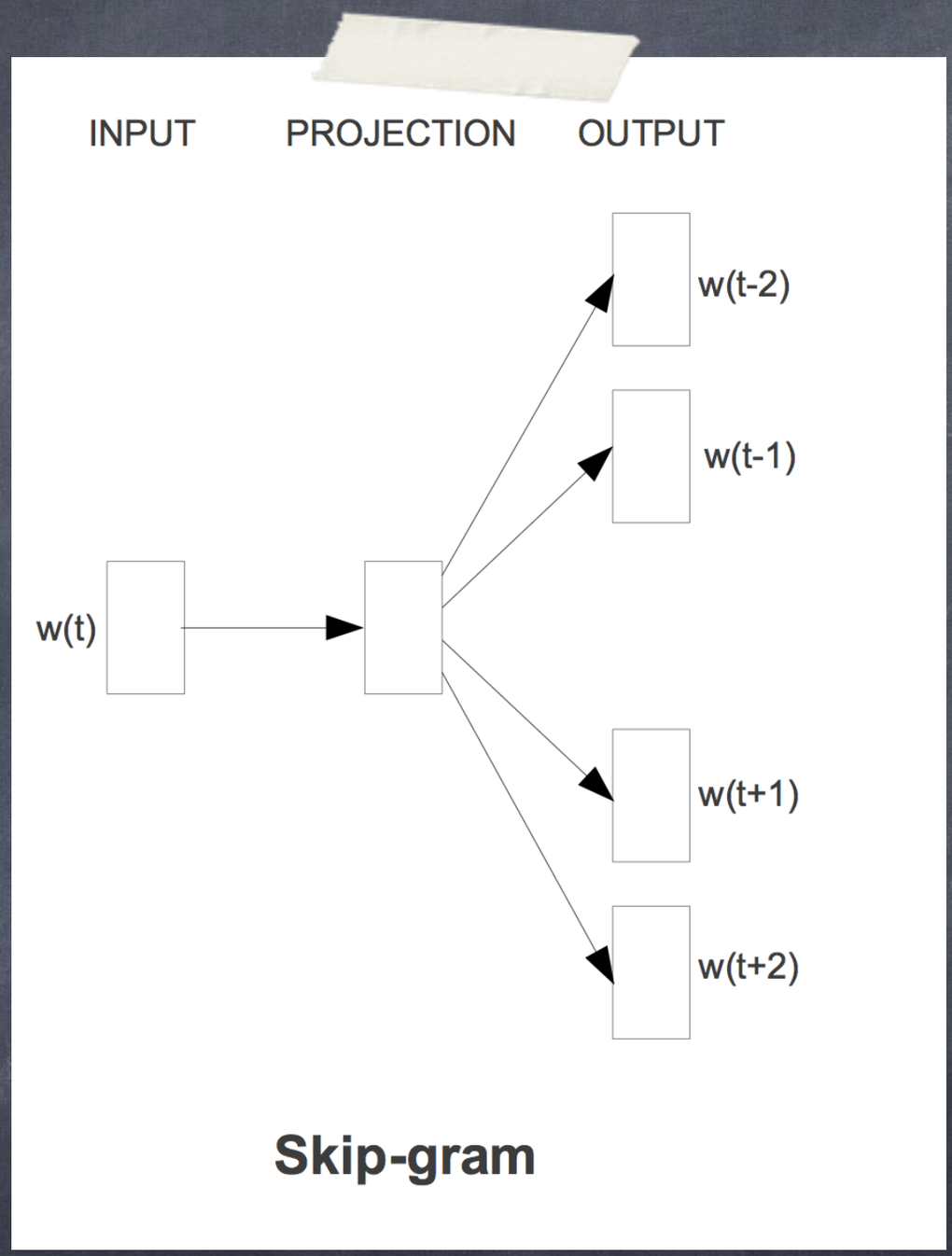
Architecture

Continuous
Bag-of-Words
Model



Architecture

Continuous Skip-gram Model



Basic Architecture

- Skip-gram:
- 1. $\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$ $p(w_O|w_I)=\frac{\exp\left({v'_{w_O}}^{\top}v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^{\top}v_{w_I}\right)}$

- 2. Represent input(vocabulary) and output(context) word vector separately, 2*V vectors(V is dict size)

- 3. stochastic gradient ascent to maximize objective function, time cost proportional to V

Efficient Architecture hierarchical softmax

- 1. Huffman Tree

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

- 2. V-1 inner node and V word have separate representation

- 3. Each word(leaf node) can be reached from the root, average L(w) is log(V), now time cost per gradient calculation?

- 4. gradient formula derivation is key

hierarchical softmax

- Computational efficient because of Huffman tree:
- 1. proportional to Log(V)
- 2. assigns short codes to the frequent words which results in fast training
- Explanation:
- 1. use binary classification to approximate multiple classification
- 2. inner node vector may have some semantic meaning

Negative Sampling

- Replace $\log P(w_O|w_I)$ with:

$$\log \sigma({v'_{w_O}}^\top v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^\top v_{w_I}) \right]$$

- Pn(w): $U(w)^{3/4}/Z)$
- motivation from NCE: good models differentiate data from noise
- prevent all the vectors from having the same value by disallowing some (w, c) pair(in the Explanation paper)

sub-sampling

- 1. reason: frequent words have less information
- 2. discard probability:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

- 3. contribution:
- faster training
- better vector for uncommon words(widen the sliding window)

Training process

- 1. read corpus and calculate each word's count
- 2. sort word array by word count(word index stored in hash table), pruning uncommon words
- 3. construct Huffman tree
- 3. read a sub-sentence from corpus(sub-sampling)
- 4. sliding window(window size random) within a sub-sentence
- 5. hierarchical softmax
- 6. negative subsampling

Result and Comparison

Table 4: *Comparison of publicly available word vectors on the Semantic-Syntactic Word Relationship test set, and word vectors from our models. Full vocabularies are used.*

Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	**64.5**	50.8
CBOW	300	783M	15.5	53.1	36.1
Skip-gram	300	783M	**50.0**	55.9	**53.3**

Table 5: *Comparison of models trained for three epochs on the same data and models trained for one epoch. Accuracy is reported on the full Semantic-Syntactic data set.*

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days]
			Semantic	Syntactic	Total	
3 epoch CBOW	300	783M	15.5	53.1	36.1	1
3 epoch Skip-gram	300	783M	50.0	55.9	53.3	3
1 epoch CBOW	300	783M	13.8	49.9	33.6	0.3
1 epoch CBOW	300	1.6B	16.1	52.6	36.1	0.6
1 epoch CBOW	600	783M	15.4	53.3	36.2	0.7
1 epoch Skip-gram	300	783M	45.6	52.2	49.2	1
1 epoch Skip-gram	300	1.6B	52.2	55.1	53.8	2
1 epoch Skip-gram	600	783M	56.7	54.5	55.5	2.5

Hyper parameters

- -size: vector dimension
- -window: context window size
- -sample: sub-sampling threshold
- -hs: whether using hierarchical softmax
- -negative: num of negative sample
- -min-count: pruning threshold
- -alpha: learning rate
- -cbow: whether using CBOW

Hyper parameter advices

- 1. CBOW is faster while Skip-gram has better performance, especially for uncommon words
- 2. softmax is efficient and good for uncommon wor
- 3. negative sampling is good for common words
- 4. sub-sampling leads to better performance and higher efficiency, sample between 1e-3 and 1e-5 is recommended.
- 5. vector size: the higher the better, but not often the case
- 6. window size: skip-gram: about 10, CBOW: about 5

Tricks

-
```
expTable = (real *)malloc((EXP_TABLE_SIZE + 1) * sizeof(real));
for (i = 0; i < EXP_TABLE_SIZE; i++) {
  expTable[i] = exp((i / (real)EXP_TABLE_SIZE * 2 - 1) * MAX_EXP); // Precompute the exp() table
  expTable[i] = expTable[i] / (expTable[i] + 1);                   // Precompute f(x) = x / (x + 1)
}
```

- 2、negative sampl … thod

方法二 word2vec 实现方法

1）类似于方法一，先将概率以 CDF 形式排列在一条线段上，以字符 a，b，c 出现概率分别为 1/2, 1/3, 1/6 为例，线段如下，左端点为 0，右端点为 1，中间分割点分别为 1/2,(1/2+1/3),(1/2+1/3+1/6)

|_______________a_______________|__________b__________|____c____|

2） 讲线段划分为 m 段，0 对应左端点，即上面的概率 0，m 对应右端点，即上面的概率 1，与上面的线段做一次映射，那么就知道 0-m 中任意整数所对应的字符了。

|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|

word2vec 中的具体代码如下，需要注意的是概率大小不是与词频正比，而是与词频的 power 次方成正比。

Tricks

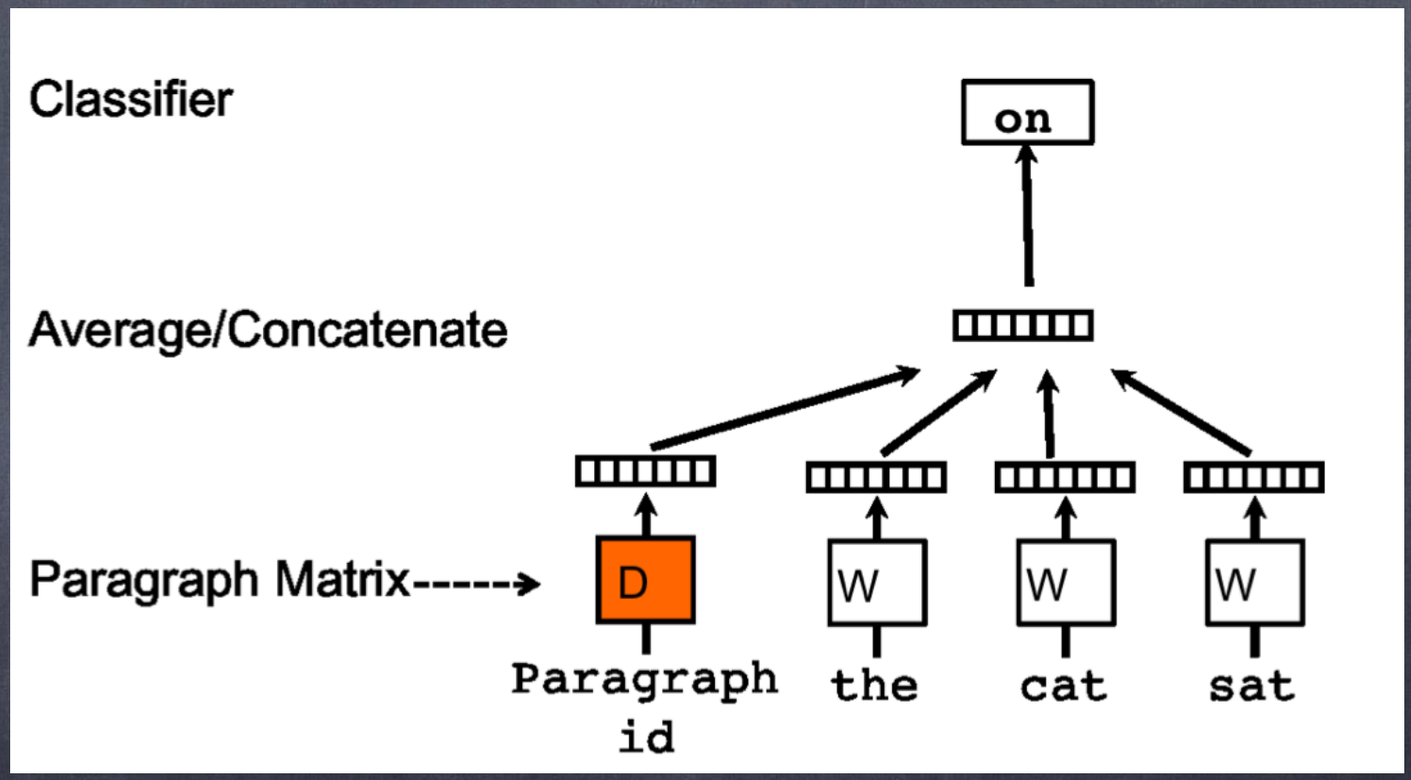
```
// Returns hash value of a word
int GetWordHash(char *word) {
  unsigned long long a, hash = 0;
  for (a = 0; a < strlen(word); a++) hash = hash * 257 + word[a];
  hash = hash % vocab_hash_size;
  return hash;
}

// Returns position of a word in the vocabulary; if the word is not found, returns -1
int SearchVocab(char *word) {
  unsigned int hash = GetWordHash(word);
  while (1) {
    if (vocab_hash[hash] == -1) return -1;
    if (!strcmp(word, vocab[vocab_hash[hash]].word)) return vocab_hash[hash];
    hash = (hash + 1) % vocab_hash_size;
  }
  return -1;
}
```

Extensions

- phrase vector
- Paragraph Vector

Paragraph Vector

- bag-of-words: ignore word orders and word semantic
- various length
- application: text classification and sentiment analysis



Thanks!