Lecture 2 Word Embedding and Representation

Word Meaning based Representation

1. Definition: Meaning

   the ideas that it represents and which can be explained using other words

   the thoughts and ideas that are intended to be expressed by it

2. Resources: WordNet(NLTK)

   calculating the similar words based on lemmas

   `from nltk.corpus import wordnet as wn`

   `pose={'n':'noun','v':'verb','s':'adj(s)','a':'adj','r':'adv'}`

   `for synset in wn.synsets("good"):`

   `print("{}: {}".format(pose[synset[synset.pos()]],", ".join([l.name() for l in synset.lemmas()])))`

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
...
adverb: well, good
adverb: thoroughly, soundly, good
```

Calculating similar synsets based on hypernyms

```
from nltk.corpus import wordnet as wn
```

```
panda=wn.synset("panda.n.01")
```

```
hyper=lambda s:s.hypernyms()
```

```
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

3. Problems

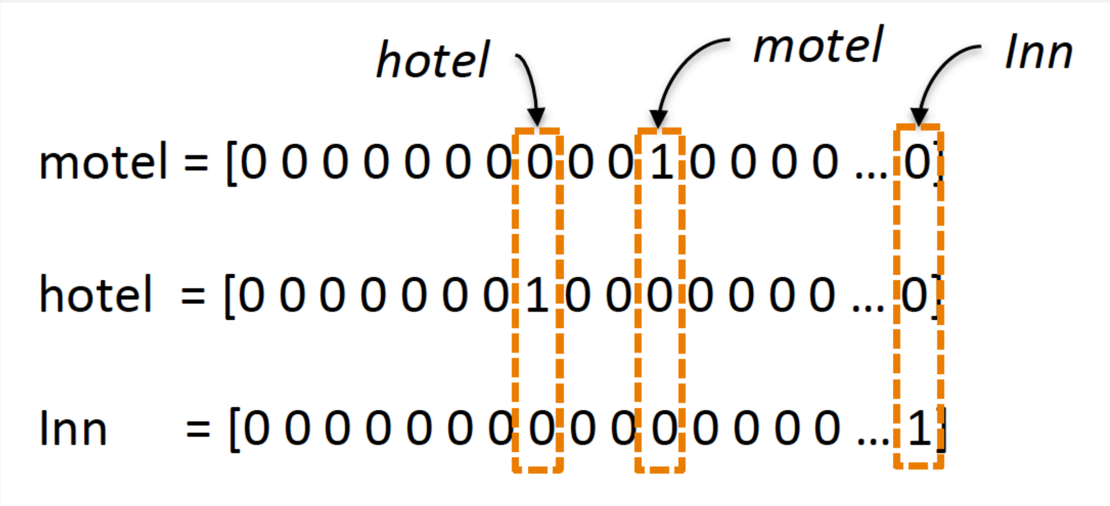
a. missing nuance: synonyms is not correct all the time
b. missing new meanings of words: impossible to keep up-to-date
c. subjective
d. need human labor to create and adapt
e. difficult to compute accurate word similarity

Count based Word representation

Sparse representation: high-dimensional features

One-hot Encoding

1. every word is represented by a binary vector with index 0 and 1
2. vecotor dimention =(1,len(vocaubulary))



3. *Manual Coding one-hot*

Sentence Tokenization

```
import nltk
nltk.download('punkt')
word_data = "I enjoy studying natural language processing"
token = nltk.word_tokenize(word_data)
print (token)
```

```
word2index={}
for voca in token:
    if voca not in word2index.keys():
      word2index[voca]=len(word2index)
print(word2index)
```
Token Indexing

```
def one_hot_encoding(word, word2index):
      one_hot_vector = [0]*(len(word2index))
      index=word2index[word]
      one_hot_vector[index]=1
      return one_hot_vector
```
One-hot Encoding

Problems

a. No word similarity: "sydeny hotel" (supposed)->"sydney inn"

b. Inefficiency : vector dimension =len(vocab)

   word only be represent with 1, others are 0s

Bag of words(BOW)

1. *Representation of text*

   The occurrence of the word in the document

   — a vocabulary of known words

   — a measure of the presence of known words

2. *Discarding the order and the structure of the word*

   Only concerned on the occurence of known words in the document, not the location of known words in the document

3. *Problem*

   a. discard the order and structure will ignore the context
   b. ignore the meaning of the word,thus ignore the

Term Frequencey-Inverse Document Frequency (TF-IDF)

Representing how important a word to a document in a corpus

1. TF=BOW

   The count of how many times a word occurs in a given document

2. IDF

   The number of times a word occurs in a corpus of documents

   $\mathrm{w_{i,j}} = \mathrm{tf_{i,j}} * \log(\frac{\mathbb{N}}{\mathrm{df_i}})$

   $w_{i,j} = weight\ of\ term\ i\ in\ document\ j$

   $tf_{i,j} = number\ of\ the\ occurence\ of\ term\ i\ in\ document\ j$

   $\mathbb{N} = total\ number\ of\ documents$

   $df_i = number\ of\ document\ contain\ term\ i$

3. IDF is the way to scale up the rare word in the document and normalize the common words in the corpus

Prediction Based Word Representation

Word Embedding: low dimensional,distributed representations

Word2Vec: *representing the words similarity*

Contimuous Bag of Words (CBOW)

Predicted center word given (bag of) context words

1. Assumption:

   — Predicted the center word by around context

   — Context are selected as input by using sliding window

Center word	Context ("outside") word	
[1,0,0,0,0,0,0]	[0,1,0,0,0,0,0], [0,0,1,0,0,0,0]	Sydney is the state capital of NSW
[0,1,0,0,0,0,0]	[1,0,0,0,0,0,0], [0,0,1,0,0,0,0], [0,0,0,1,0,0,0]	Sydney is the state capital of NSW
[0,0,1,0,0,0,0]	[1,0,0,0,0,0,0], [0,1,0,0,0,0,0] [0,0,0,1,0,0,0], [0,0,0,0,1,0,0]	Sydney is the state capital of NSW
[0,0,0,1,0,0,0]	[0,1,0,0,0,0,0], [0,0,1,0,0,0,0] [0,0,0,0,1,0,0], [0,0,0,0,0,1,0]	Sydney is the state capital of NSW
[0,0,0,0,1,0,0]	[0,0,1,0,0,0,0], [0,0,0,1,0,0,0] [0,0,0,0,0,1,0], [0,0,0,0,0,0,1]	Sydney is the state capital of NSW
[0,0,0,0,0,1,0]	[0,0,0,1,0,0,0], [0,0,0,0,1,0,0] [0,0,0,0,0,0,1]	Sydney is the state capital of NSW
[0,0,0,0,0,0,1]	[0,0,0,0,1,0,0], [0,0,0,0,0,1,0]	Sydney is the state capital of NSW

Center word
Context ("outside") word

2. Neural network archetecture

   — input (context) and output (context) are one-hot vector

   — W_i : $V * N$ weight matrix between input layer and hidden layer

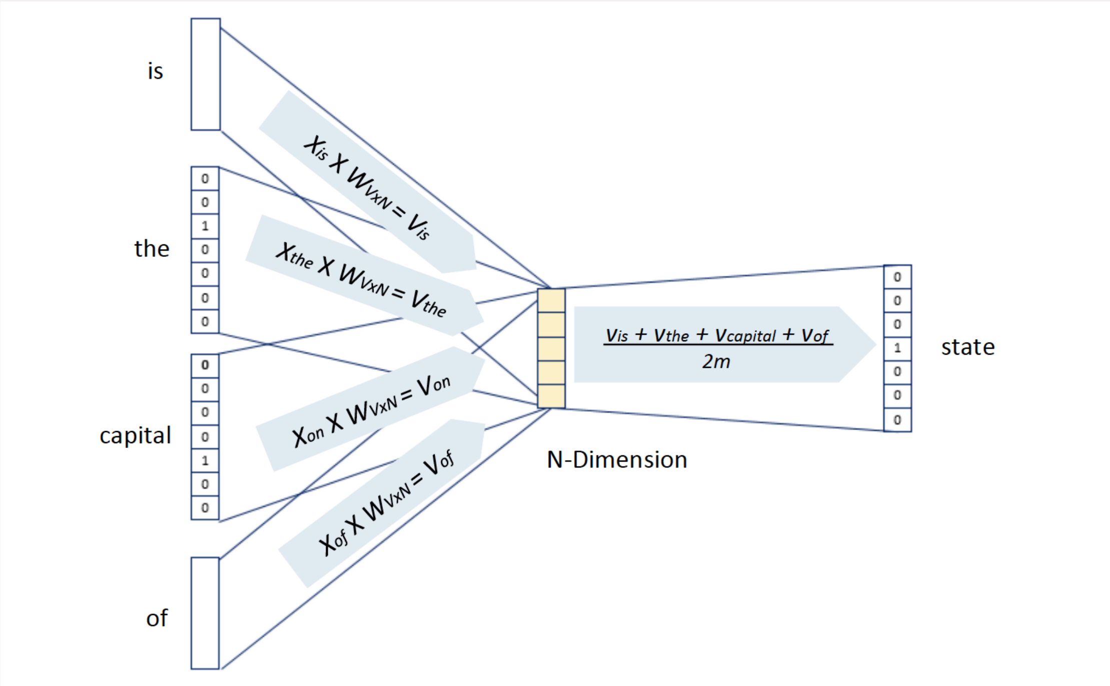
   — W_j : $N * V$ weight matrix between output layer and hidden layer

   Iuput Layer

   a. Generate one-hot: the input one hot $x_{(c-m)}, \dots, x_{(c-1)}, x_{(c+1)}, \dots, x_{(c+m)}$ for the input context of size m, where x_c is the center word. We have $C(=2m)$ one hot word vector of size $[1 * V]$. so our input layer size is$[C * V]$

b. N-dimension: multiply W_i ,return embedded words $[1 * N]$

c. Averaging: take average of these $2m[1 * N]$vectors



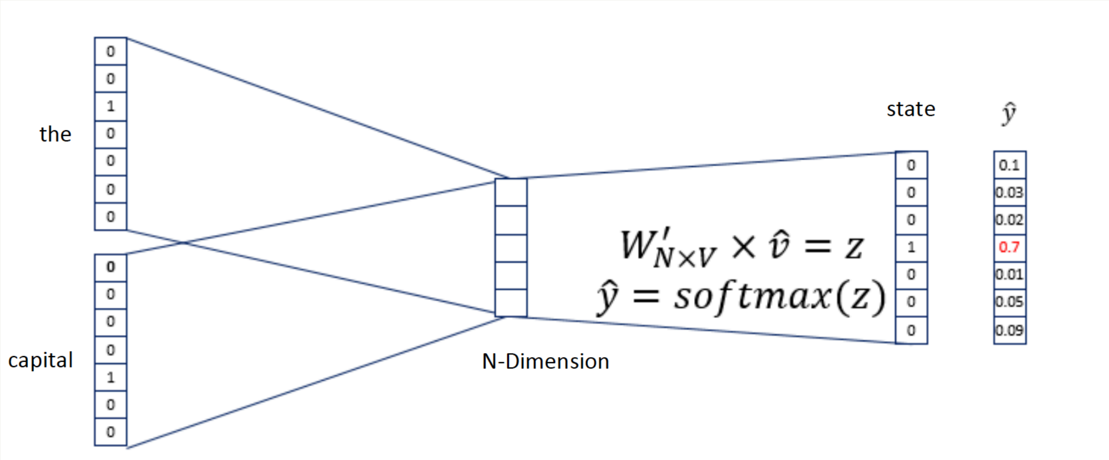
Output Layer

a. score vector: calculate the hidden layer output by multiplying hidden layer input with matrix W_j. Now we get a score vector of size$[1 * V]$. lets name it z.

b. Classifier: score probability $\hat{y} = softmax(z)$

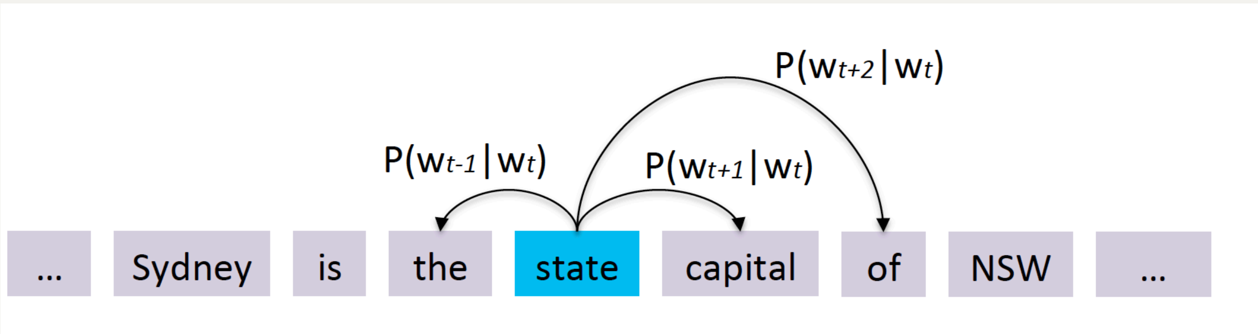
c. Evaluation(Cross-entropy): error between output and target is calculated and propagated back to re-adjust the weights

$H(\hat{y}, y) = -\sum_{j=1}^{|v|} ylog(\hat{y})$



Continuous Skip-gram:

Predicted context("outside") words (position independent) given a centre word

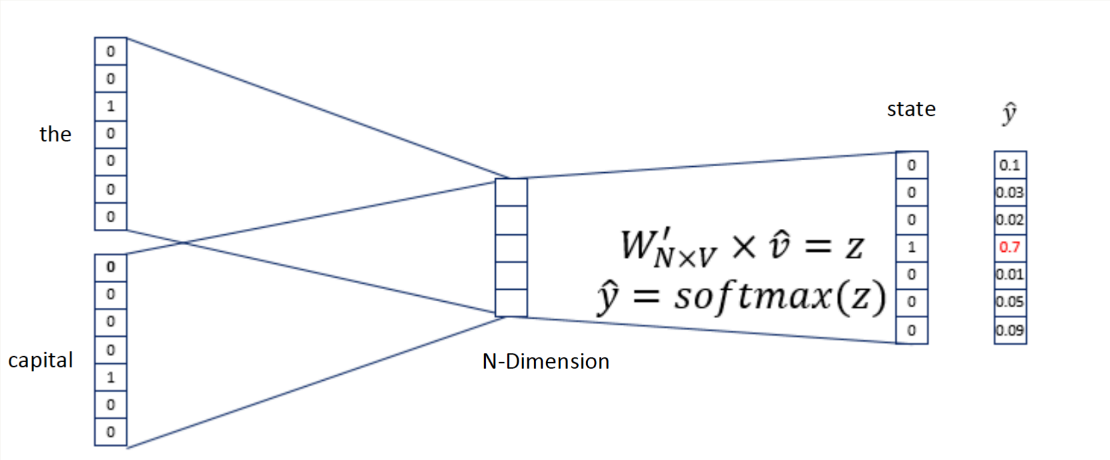


Iuput Layer

1. Generate one-hot: the input centert word x_c ,where $x_{(c-m)}, \ldots, x_{(c-1)}, x_{(c+1)}, \ldots, x_{(c+m)}$ are context words to predict. so our input layer size is $[1 * V]$
2. N-dimension: multiply W_i ,return embedded words $[1 * N]$

Output Layer

1. score vector: calculate the hidden layer output by multiplying hidden layer input with matrix W_j. Now we get $C(=m)$ score vector of size$[1 * V]$. lets name it z.
2. Classifier: score probability $\hat{y} = softmax(z)$
3. Evaluation: error between output and target is calculated and propagated back to re-adjust the weights



Note. ***$\hat{y}$ are all same but their target vectors are different so they all give different error vectors and Element-wise sum is taken over all the error vectors to obtain a final error vector.***

Word2vec Summary

Idaeas

1. Have a large corpus
2. Every word in a fixed vocabulary is represented by a vector
3. Go through each position t in text, which has a center word c and context words o
4. Use the similarity of the word vectors for c and o to calculate the probability of o given c

5. Keep adjusting the word vectors to maximize this probability

Problems

1. Cannot cover the morphological similarity

   — every word as an independent vector

2. Hard to conduct embedding for rare words

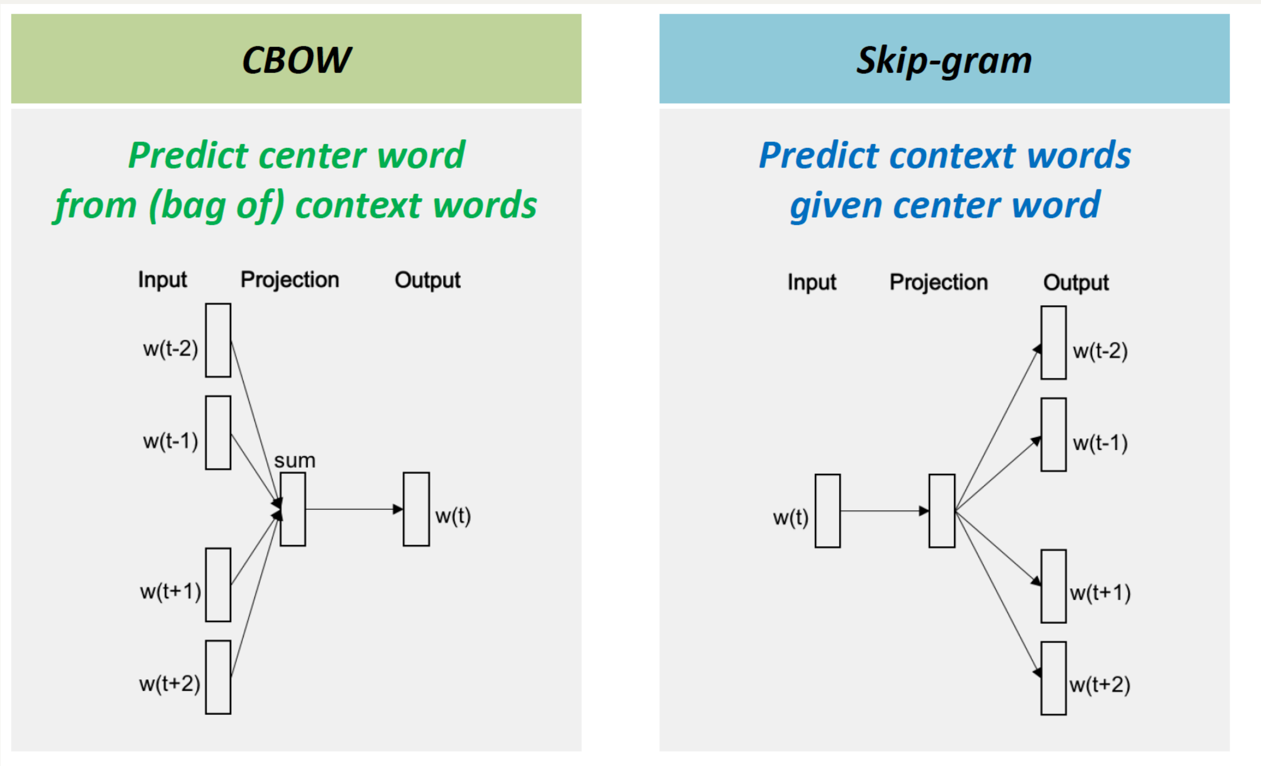
   — based on the hypothesis distrbution

   — dosen't embed the rare words

3. Cannot handle the OOV (Out-of-Vocabulary)

   — only work if the word is in the vocabulary

Comparison

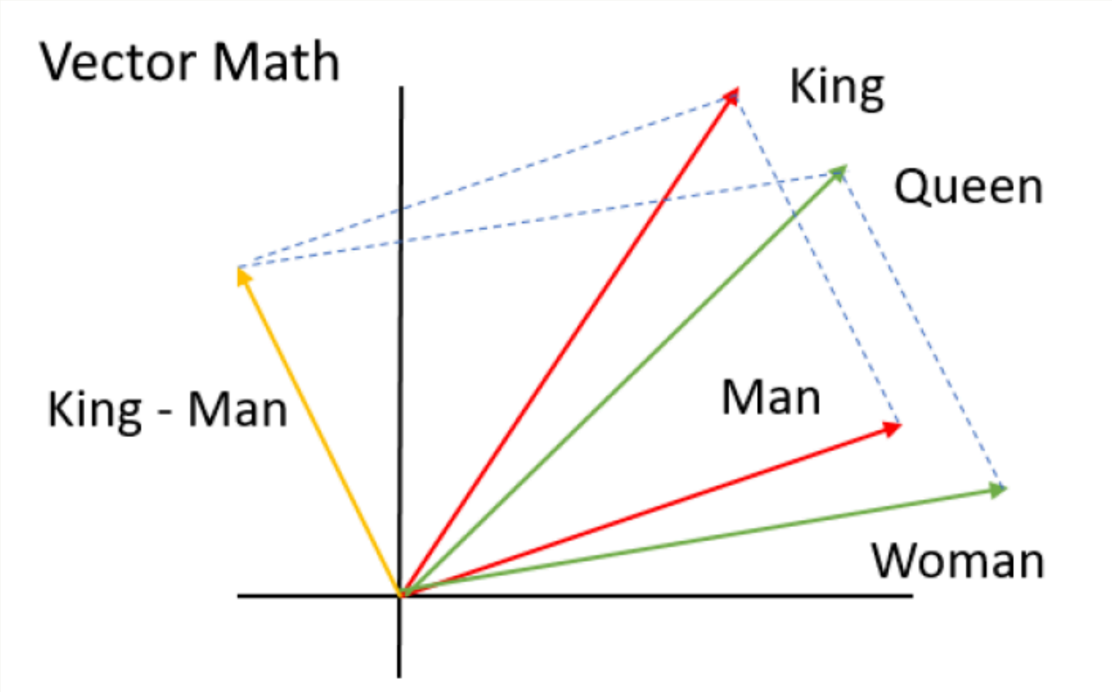


Word Embedding Evaluation

Type	How to work / Benefit
Intrinsic	Evaluation on a specific/intermediate subtask • Fast to compute • Helps to understand that system • Not clear if really helpful unless correlation to real task is established
Extrinsic	Evaluation on a real task • Can take a long time to compute accuracy • Unclear if the subsystem is the problem or its interaction or other subsystems

1. Intrisic word vector evaluation

   Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions



FastText

1. Handling the limitation 2 of the Word2Vec

2. Instead of feeding individual words into the neural network, it take breaks words into several n-grams(sub-words)

3. FastText with n-grams

   — breaking down words with n-grams

   — feed n-grams into the neural network (CBOW or Skip-gram)

   — rare words can be properly represented as it's most likely that some of their n-grams also appears in other words

Global Vectors (CloVe)

1. Handling limtation 1 of Word2Vector
2. Training on global co-occurence counts rather than on separate local context windows