ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Lesson 11:
Deep learning for NLP

Word embedding

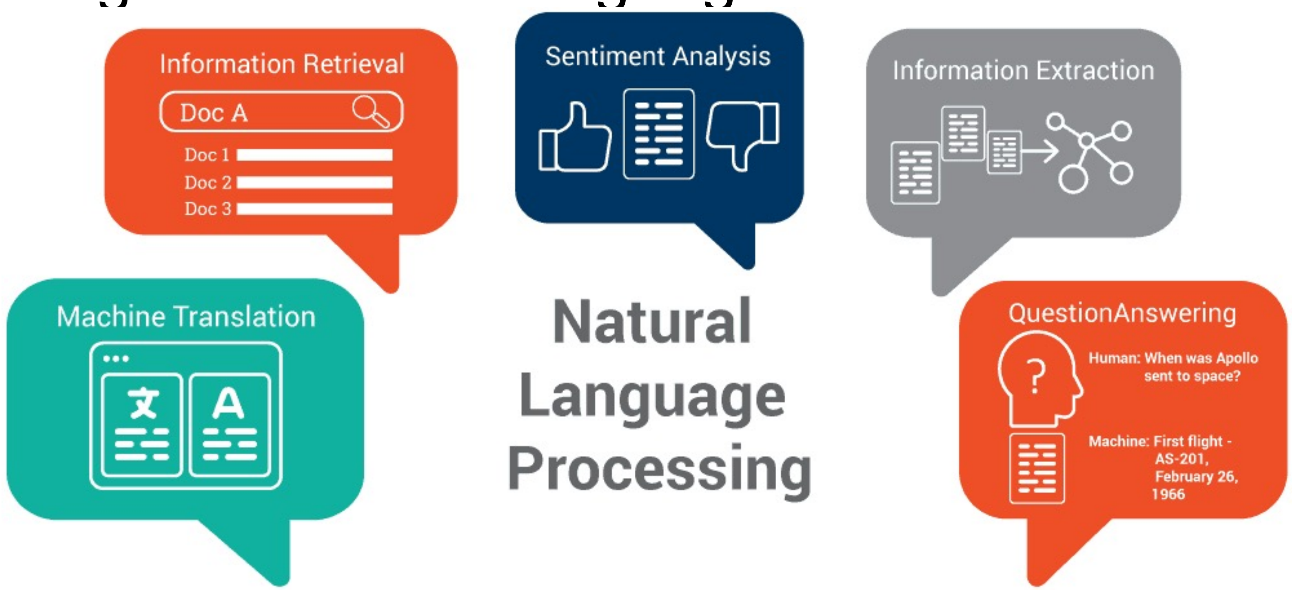
Outline

- Overview of natural language processing
- Word and text representation
- Pre-trained NLP models

Overview of natural language processing

What is natural language processing (NLP)

- Natural language processing is a branch of artificial intelligence that deals with the interaction between computers and human languages.
- The purpose of natural language processing is to enable computers to read, understand, and derive meaning from human language.



NLP tasks

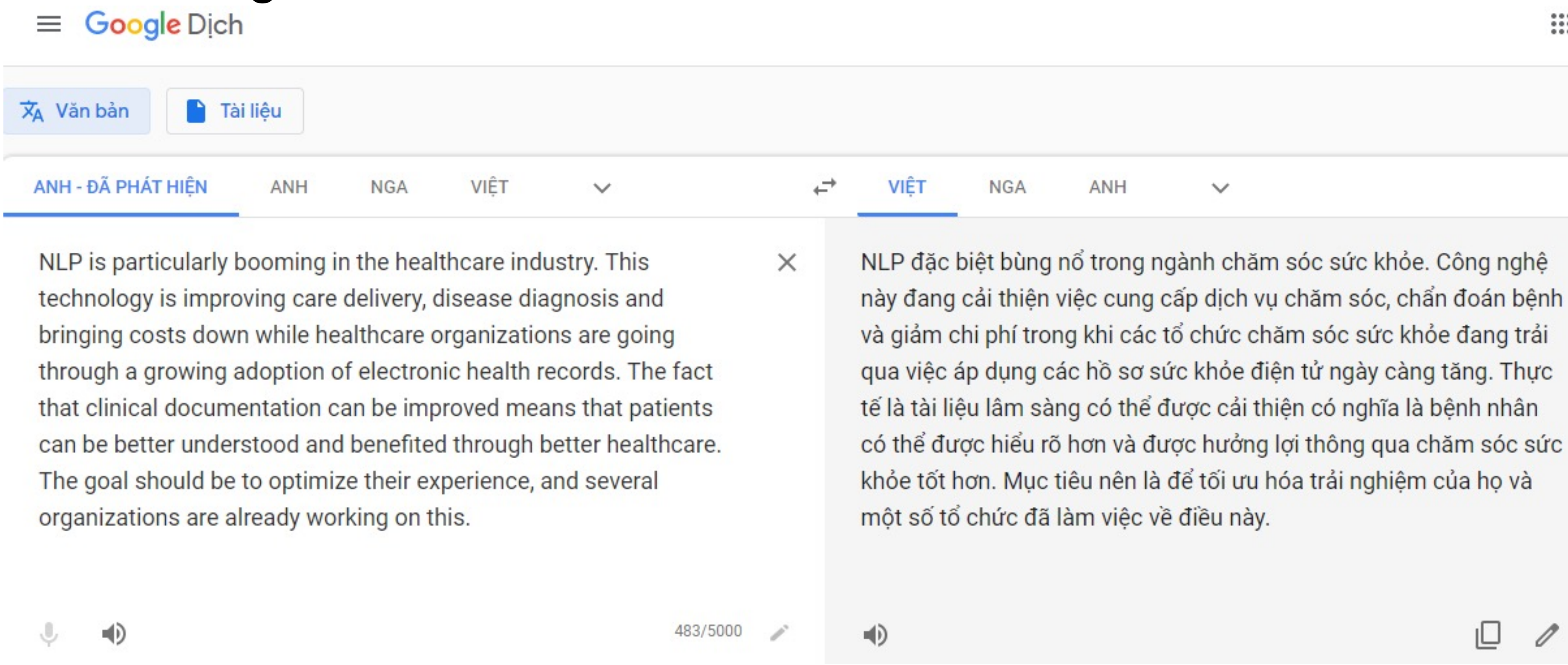
- **Morphology:** how words are constructed, word prefixes and suffixes
- **Syntax:** the relationship of grammatical structure between words and phrases
- **Semantics:** meaning of words, phrases, and expressions
- **Discourse:** the relationship between expressions or sentences
- **Pragmatic:** utterance purposes, how to use language in communication
- **World Knowledge:** knowledge about the world, latent knowledge

NLP applications

- Speech recognition
- Text mining
  - Text clustering
  - Text classification
  - Text summarization
  - Topic modeling
  - Question answering
- Language tutoring
  - Grammar/Spelling Correction
- Machine translation

Machine translation

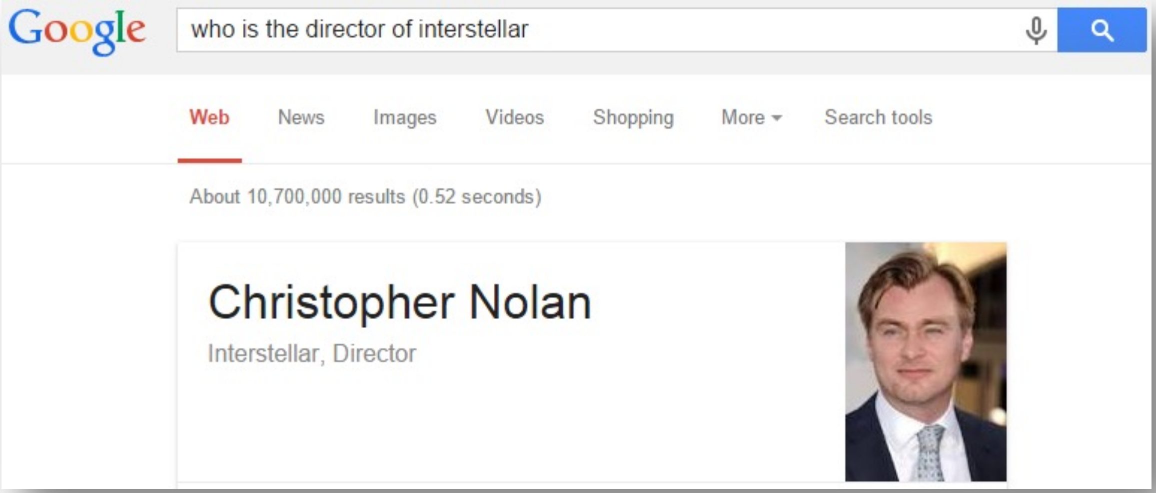
- Google translate



Conversation systems

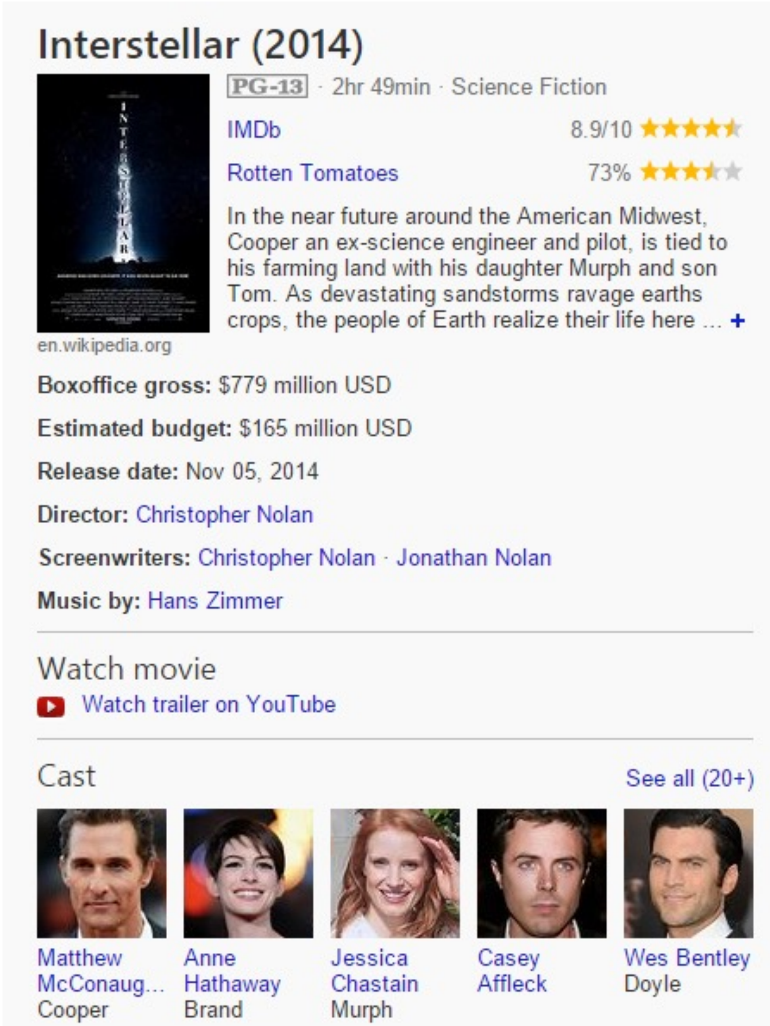
- Chatbot, virtual assistant, automatic Q&A

Apple's siri system



Google search

Information extraction



Google Knowledge Graph

Wiki Info Box

Tokenization

- Split text into sentences and words

There was an earthquake near D.C. I’ve even felt it in Philadelphia, New York, etc.

There + was + an + earthquake + near + D.C.

I + ve + even + felt + it + in + Philadelphia, + New + York, + etc.

Part-of-Speech tagging

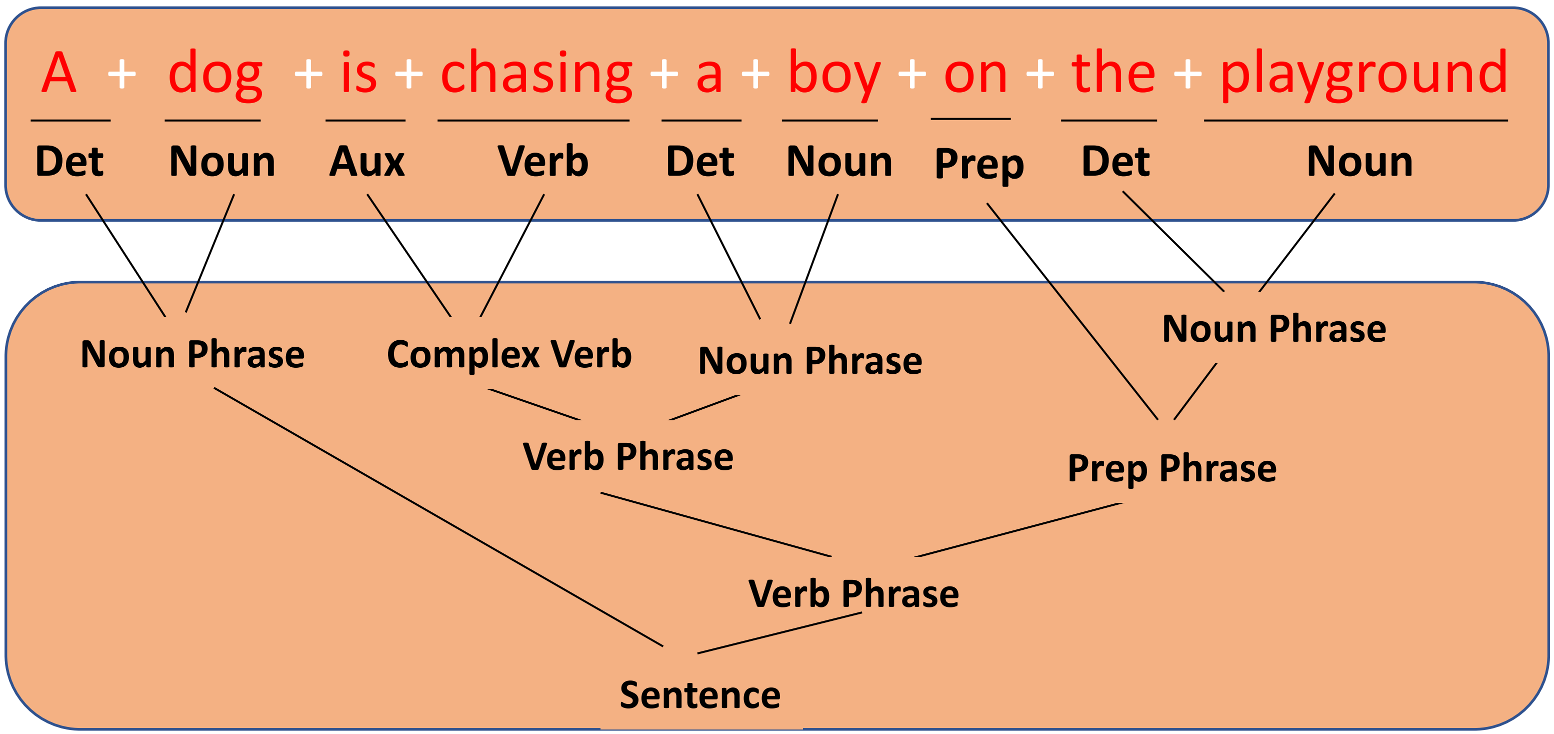
- Identify the word type of every word in the text

A + dog + is + chasing + a + boy + on + the + playground

A	dog	is	chasing	a	boy	on	the	playground
Det	**Noun**	**Aux**	**Verb**	**Det**	**Noun**	**Prep**	**Det**	**Noun**

Syntactic parsing

- Grammatical analysis of a given sentence according to grammar rules



Named entity recognition (NER)

- Search and classify text elements into predefined categories such as names of people, organizations, places, times, quantities, monetary values, etc.

Its initial Board of Visitors included U.S. Presidents Thomas Jefferson, James Madison, and James Monroe.

Its initial Board of Visitors included U.S. Presidents Thomas Jefferson, James Madison, and James Monroe.

Organization, **Location**, **Person**

Relation extraction

- Identify relationships between entities
  - Semantic analysis at a shallow level

Its initial Board of Visitors included U.S. Presidents Thomas Jefferson, James Madison, and James Monroe.

1. Thomas Jefferson Is_Member_Of Board of Visitors
2. Thomas Jefferson Is_President_Of U.S.

Semantic parsing

- Deep level semantic analysis

Its initial Board of Visitors included U.S. Presidents Thomas Jefferson, James Madison, and James Monroe.

$\exists x$ (Is_Person(x) & Is_President_Of(x,'U.S.') & Is_Member_Of(x,'Board of Visitors'))

Word and text representation

Word and text representation

- WordNet: a dictionary containing lists of synonyms and hypernyms

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[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
```

e.g. hypernyms of "panda":

```
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')]
```

Disadvantages of WordNet

- Lack of nuance
  - For example, "sacrifice" means "death"
- Missing meaning for
  - New words about technology, teen language…
- Wordnet depends on the subjective thoughts of the makers
- It takes a lot of labor to create and edit
- Can't calculate similarity between two words

One-hot encoding

- Representing words as discrete symbols
- The length of the vector is equal to the number of words in the dictionary

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Problem of one-hot encoding

- Users searching for “Hanoi hotel”, we will also want to show results for “Hanoi motel”
- But these two words are orthogonal, the similarity is zero!

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Problem of one-hot encoding

- The vectors are too long
  - With daily language of about 20K words, machine translation 50K words, material science 500K words, google web crawl 13M words

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

- Solution
  - Relying on WordNet? but WordNet is not perfect and many disadvantages…
  - **Learn to encode similarities in vector representations**

Representing a word by its context

- Distributed semantics: The meaning of a word is determined by the words that often appear near it

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

(J. R. Firth 1957: 11)

- When a word appears in text, its context is the set of words that appear next to it (in a fixed-size window).
- Use different contexts of a word to build its meaning

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

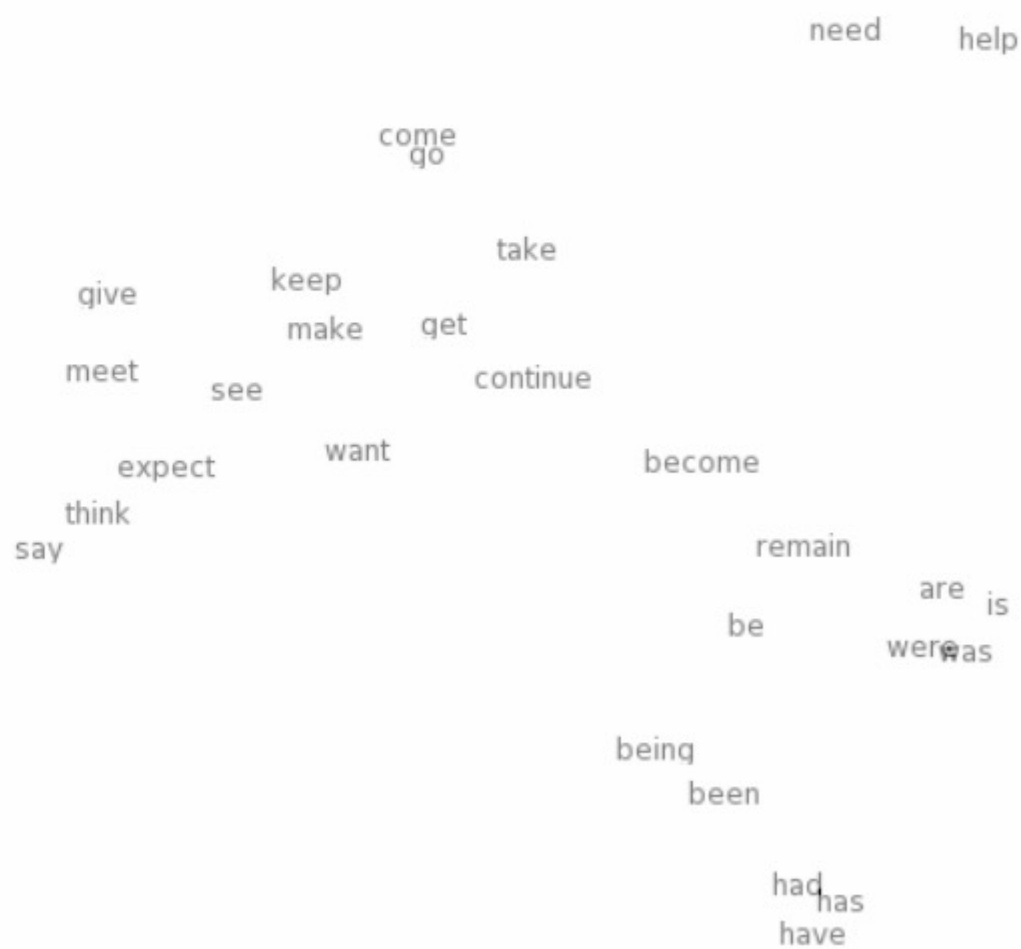
Word vector

- Each word is represented by a dense vector such that this vector is similar to the vectors representing other words that often appear in similar contexts.
- Word vectors are also known as word embeddings or word representations

$$banking = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Word vector

$$\textit{expect} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

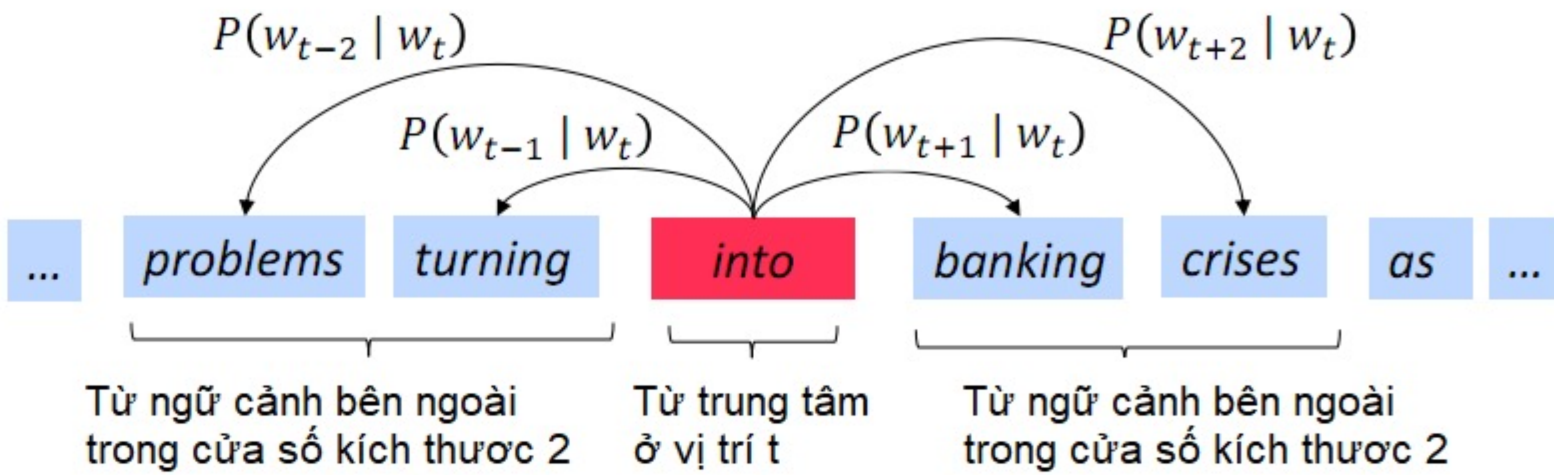


Word2vec

- Word2vec (Mikolov et al. 2013) is a method to learn the word representation
- **Idea**
  - Using a large set of documents (corpus)
  - Each word in a fixed vocabulary is represented by a vector .
  - Traverse each position t in the text, each containing **the central word c and the outer context words o**
  - Use the similarity of the representation vectors **c** and **o** to calculate the probability that o occurs when c is present (or vice versa).
  - Tweak the word vectors to maximize this probability

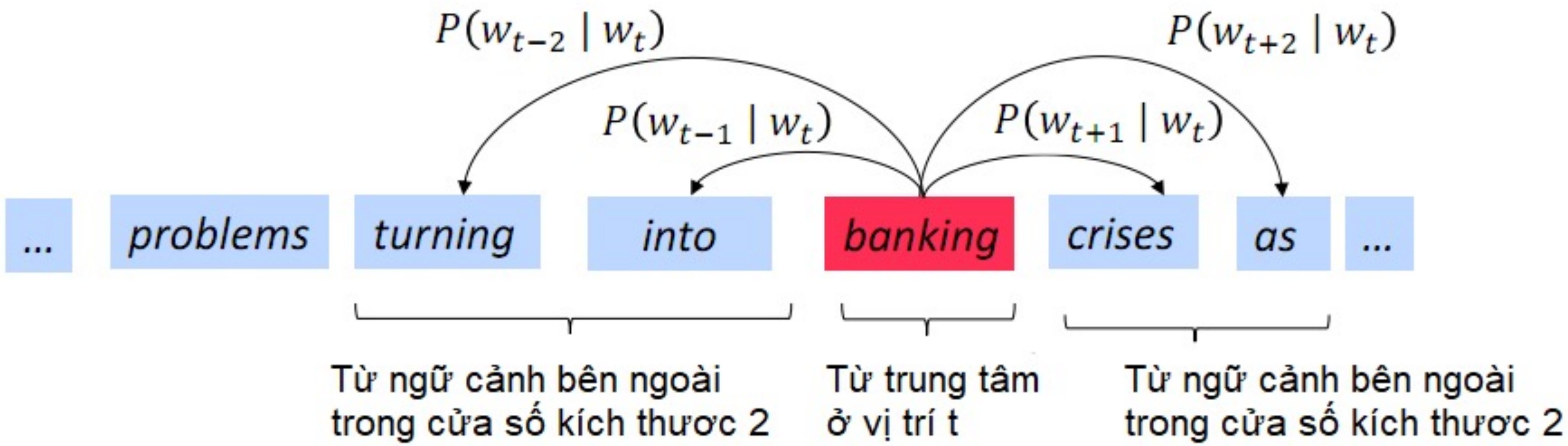
Word2vec

- Example, $P(w_{t+j} \mid w_t)$ in a window of size 2



Word2vec

- Example, $P\left(w_{t+j} \mid w_t\right)$ in a window of size 2



Word2vec: objective function

- Likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{t+j} \mid w_t; \theta)$$

- Loss function:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P(w_{t+j} \mid w_t; \theta)$$

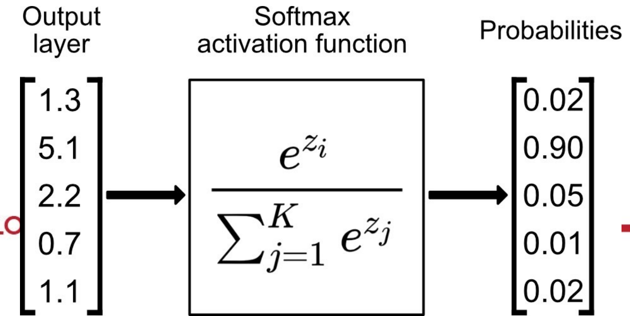
Word2vec

- How to calculate $P(w_{t+j} \mid w_t; \theta)$?

- We will use two vectors for each word w:
  - v_w when w is the central word
  - u_w when w is the outer context word
- Then with the central word c and the outer context word o we have:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Do you recall Softmax?



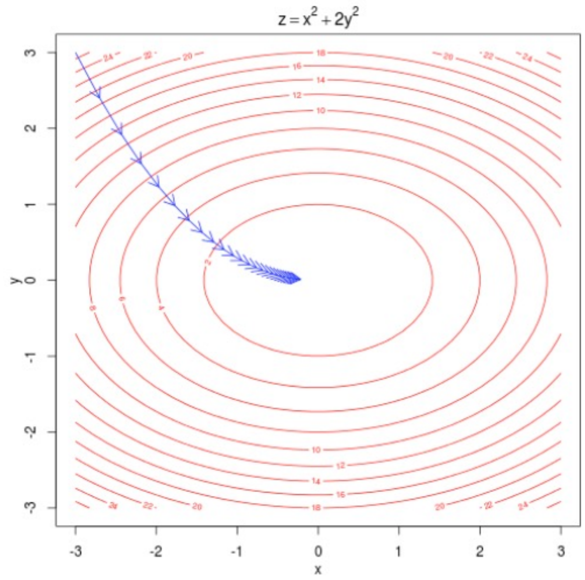
Word2vec

- Model parameters:

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_{a} \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_{a} \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

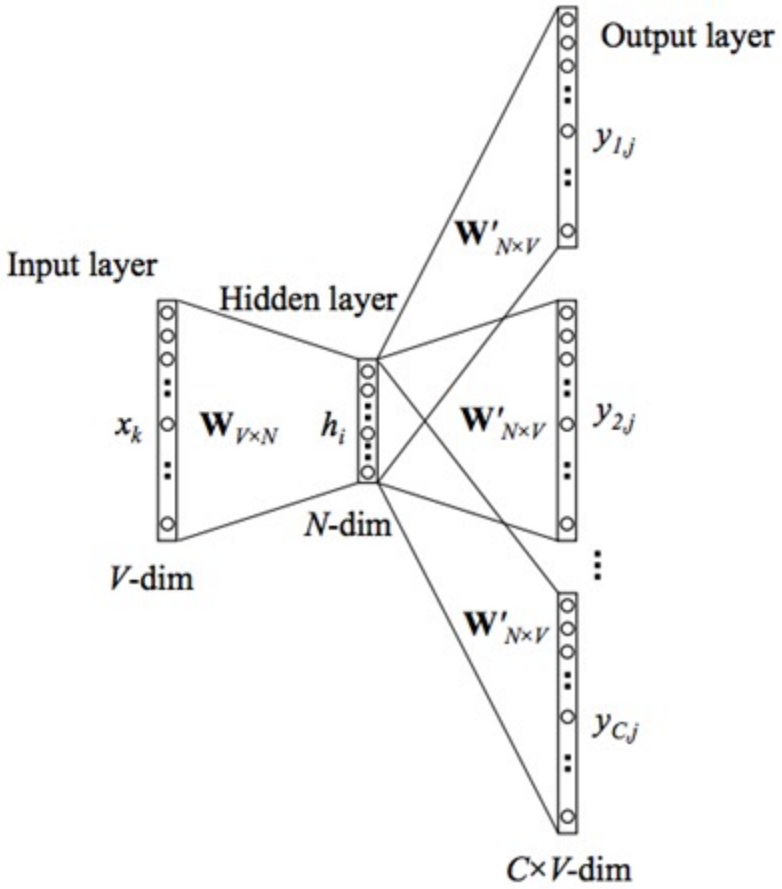
- Training by SGD:

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$$



Word2vec: The skip-gram model

- Dictionary size: V
- Input layer: one-hot encoding of the central word
- k^{th} row of W_{VxN} is the central vector of the k^{th} word.
- The k^{th} column of W'_{NxV} is the context vector fo the k^{th} in V.
- Notice that each word is represented by two vectors, both randomly initialized.



Word2vec: The skip-gram model

Source Text	Training Samples
The quick brown fox jumps over the lazy dog.	(the, quick) (the, brown)
The **quick** brown fox jumps over the lazy dog.	(quick, the) (quick, brown) (quick, fox)
The quick **brown** fox jumps over the lazy dog.	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown **fox** jumps over the lazy dog.	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

Word2vec: The skip-gram model

- Problem: The denominator takes a long time to calculate!

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

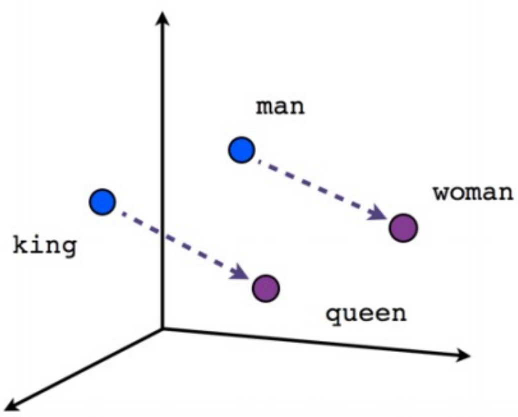
- Use negative sampling:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta)$$

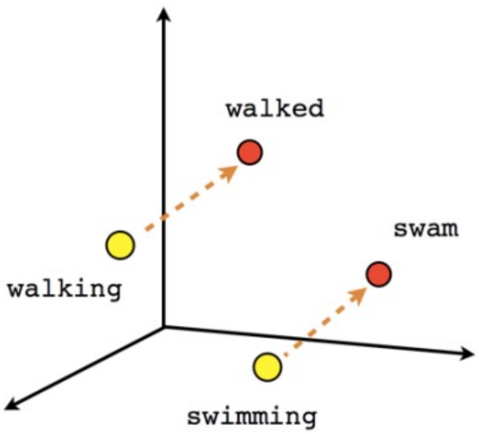
$$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{i=1}^{\kappa} \mathbb{E}_{j \sim P(w)} \left[\log \sigma\left(-u_j^T v_c\right)\right]$$

- $p(w)=U(w)^{3/4}/Z$, where U(w) is a 1-gram distribution

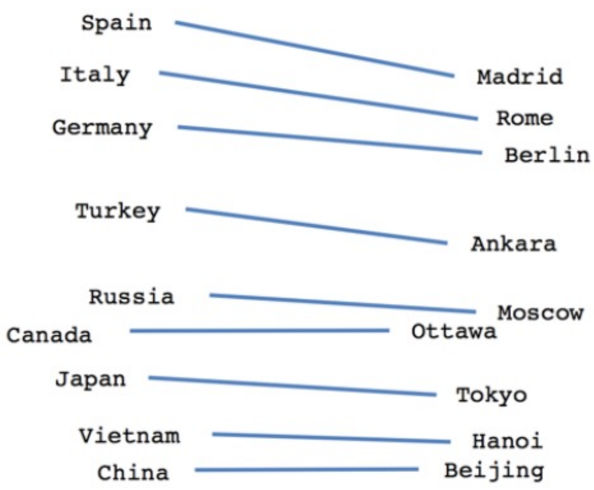
Some results of word2vec



Male-Female

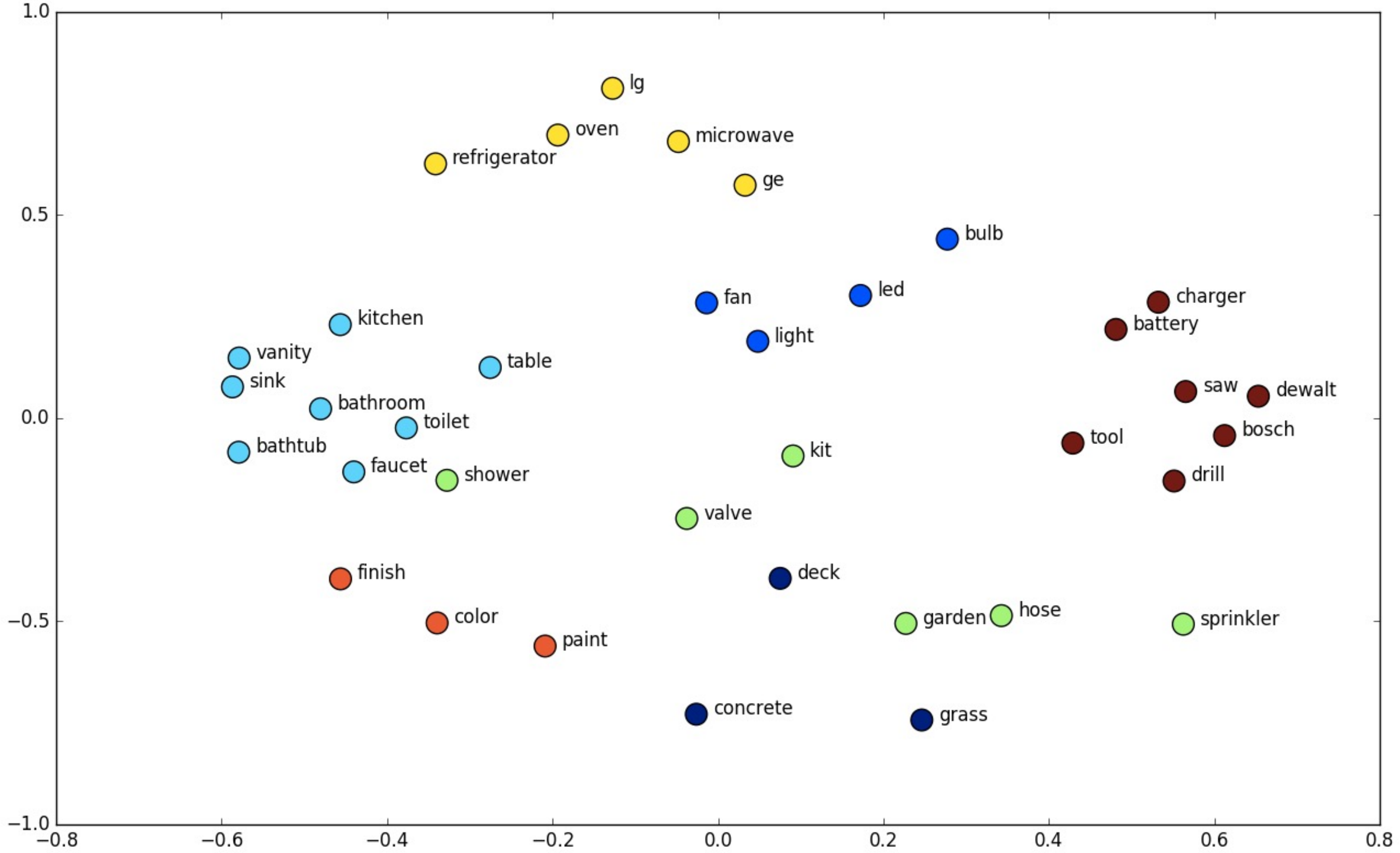


Verb tense

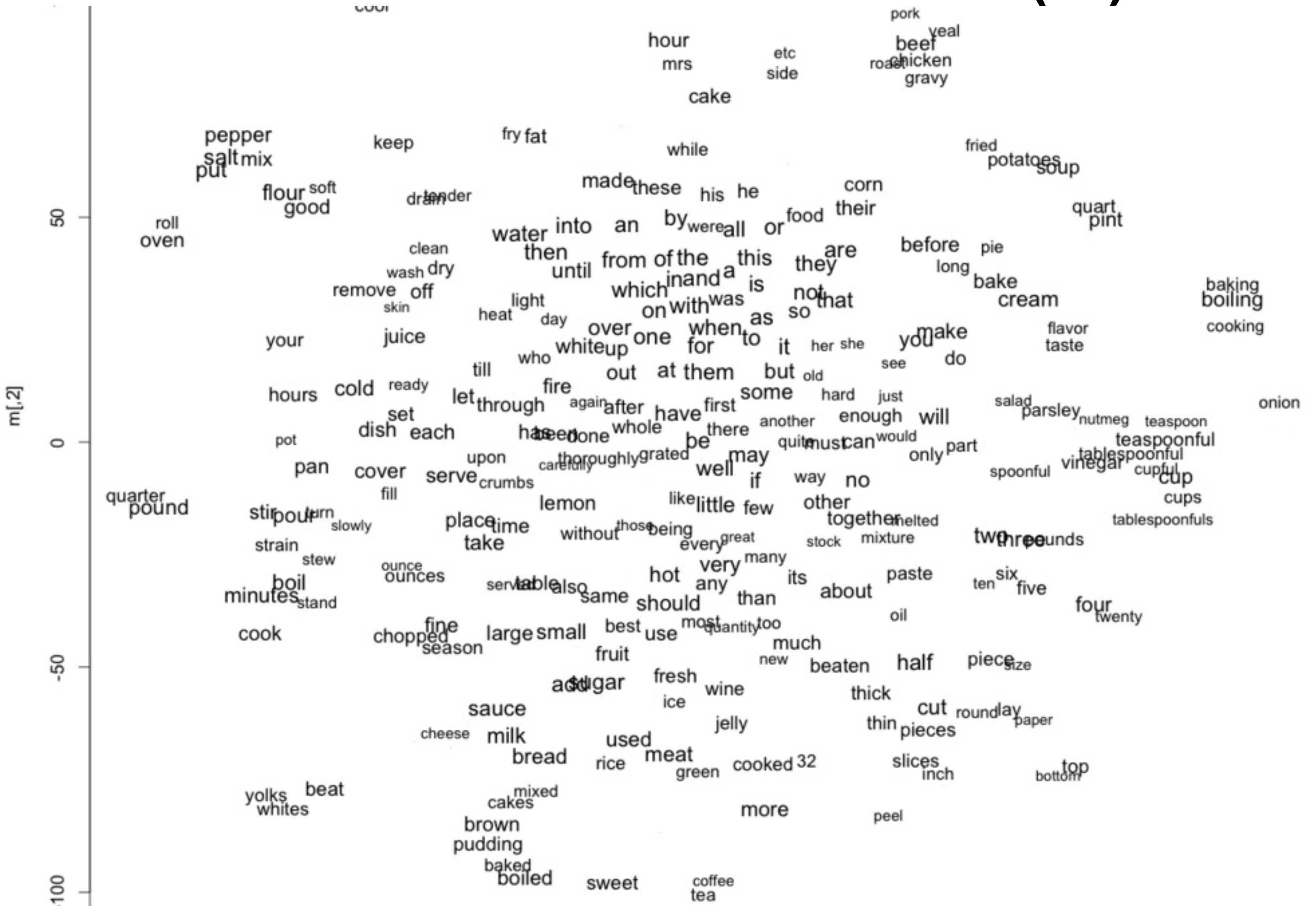


Country-Capital

Some results of word2vec (2)

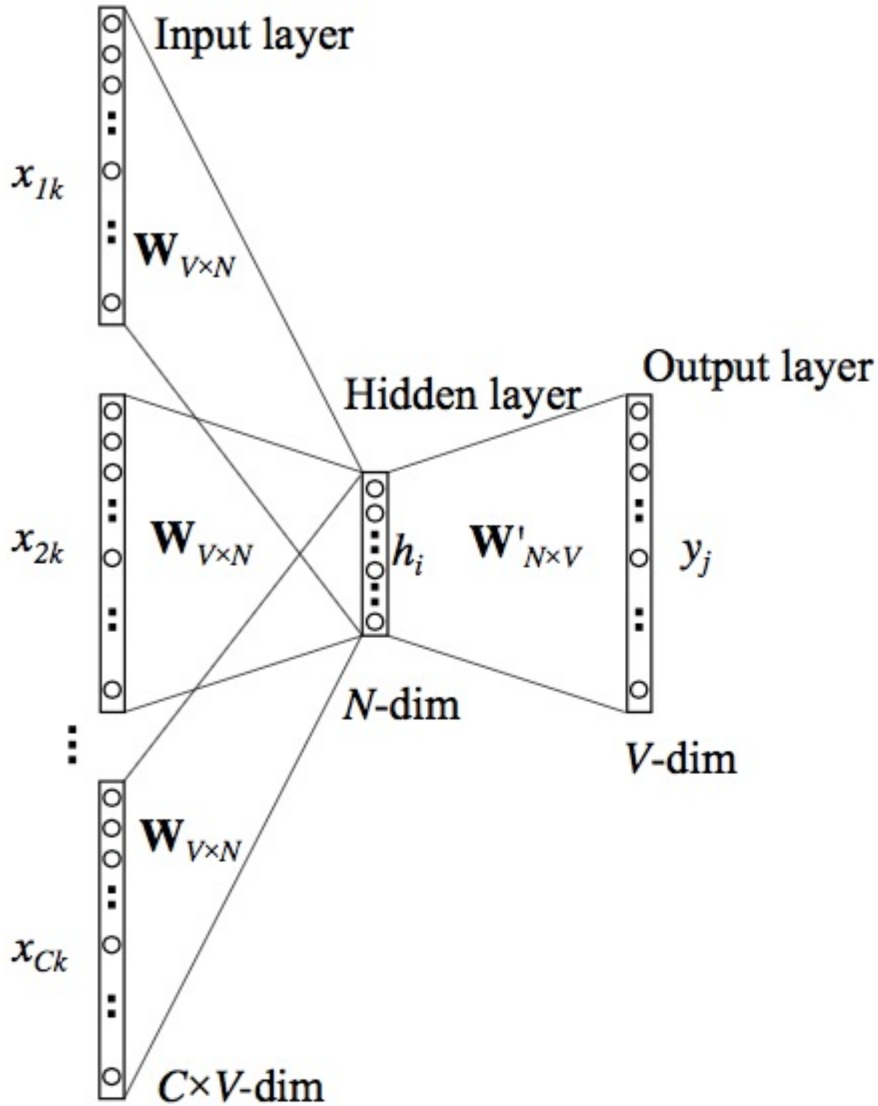


Some results of word2vec (3)



Word2vec: Continuous BOW

- Use context words to guess the central word



Words and co-occurrence vectors

Co-occurrence Matrices

- We represent how often a word occurs in a document
  - Term-document matrix
- Or how often a word occurs with another
  - Term-term matrix
  - (or word-word co-occurrence matrix or word-context matrix)

Term-document matrix

- Each cell: count of word w in a document d:
- Each document is a count vector in $\mathbb{N}^{V}$: a column below
- Two documents are similar if their vectors are similar

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

The words in a term-document matrix

- Each word is a count vector in $\mathbb{N}^D$: a row below
- Two words are similar if their vectors are similar

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	6	117	0	0

Window based co-occurrence matrix

- Or the word-word or word-context matrix
- Instead of entire documents, use smaller contexts
  - Paragraph
  - Window of ± 4 words
- Symmetrical (regardless of left and right)
- Corpus example:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

Window based co-occurrence matrix

- Window size = 1 (normaly 5-10)
- Symmetrical (regardless of left and right)
- Corpus example:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Problem of co-occurrence matrix

- Size increases as word count increases
- Large number of dimensions, requires a lot of storage memory
- Solution
  - Dimension reduction
  - Usually 25-1000 dimensions (equivalent to word2vec)

$$\underset{|V| \times |V|}{\begin{bmatrix} \\ \\ X \\ \\ \\ \end{bmatrix}} = \underset{|V| \times k}{\begin{bmatrix} \\ \\ W \\ \\ \\ \end{bmatrix}} \underset{k \times k}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_k \end{bmatrix}} \underset{k \times |V|}{\begin{bmatrix} \quad C \quad \end{bmatrix}}$$

GloVe

- Combining word2vec and co-occurrence matrix

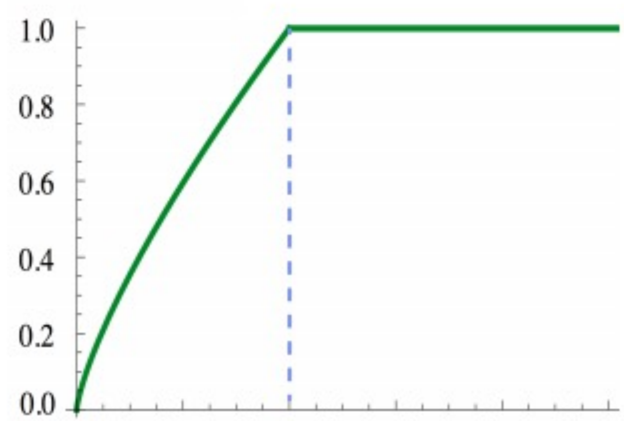
$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

- Quick training
- Scalable for large corpus
- Good performance even

with small corpus and

small vectors

$f \sim$



- Learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence

Pre-trained NLP models

Gensim

- pip install gensim

```
from gensim.models.word2vec import Word2Vec
from multiprocessing import cpu_count
import gensim.downloader as api

# Download dataset
dataset = api.load("text8")
data = [d for d in dataset]

# Split the data into 2 parts. Part 2 will be used later to update the model
data_part1 = data[:1000]
data_part2 = data[1000:]

# Train Word2Vec model. Defaults result vector size = 100
model = Word2Vec(data_part1, min_count = 0, workers=cpu_count())

# Get the word vector for given word
model['topic']
#> array([ 0.0512,  0.2555,  0.9393, ... ,-0.5669,  0.6737], dtype=float32)
```

Gensim

```
model.most_similar('topic')
#> [('discussion', 0.7590423822402954),
#>  ('consensus', 0.7253159284591675),
#>  ('discussions', 0.7252693176269531),
#>  ('interpretation', 0.7196053266525269),
#>  ('viewpoint', 0.7053568959236145),
#>  ('speculation', 0.7021505832672119),
#>  ('discourse', 0.7001898884773254),
#>  ('opinions', 0.6993060111999512),
#>  ('focus', 0.6959210634231567),
#>  ('scholarly', 0.6884037256240845)]

# Save and Load Model
model.save('newmodel')
model = Word2Vec.load('newmodel')

# Update the model with new data.
model.build_vocab(data_part2, update=True)
model.train(data_part2, total_examples=model.corpus_count, epochs=model.iter)
model['topic']
# array([-0.6482, -0.5468,  1.0688,  0.82  , ... , -0.8411,  0.3974], dtype=float32)
```

Gensim

- Use pretrained models in Gensim

```
import gensim.downloader as api

# Download the models
fasttext_model300 = api.load('fasttext-wiki-news-subwords-300')
word2vec_model300 = api.load('word2vec-google-news-300')
glove_model300 = api.load('glove-wiki-gigaword-300')

# Get word embeddings
word2vec_model300.most_similar('support')
# [('supporting', 0.6251285076141357),
#  ...
#  ('backing', 0.6007589101791382),
#  ('supports', 0.5269277691841125),
#  ('assistance', 0.5207136746547699),
#  ('supportive', 0.5110025405883789)]
```

Pretrained models

BERT:

- Github: https://github.com/google-research/bert
- Paper: Bidirectional Encoder Representations from Transformers

XLNet:

- Github: https://github.com/zihangdai/xlnet
- Paper: XLNet: Generalized Autoregressive Pretraining for Language Understanding

References

1. Stanford cs244n
https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attention!!!