

# Lecture 7

WORD REPRESENTATION LEARNING

# Contents

- Word Representation
  - Word Vectorization
  - Word Embedding
- Word2Vec
- Other models
  - GLoVE
  - FastText
  - ELMo
- Building a representation word model
- Application: Plagiarism Detection

# Review - Text Document Vectorization Approaches

# The meaning of a word

- Meaning:
  - the idea that is represented by a word, phrase, etc.
  - the idea that a person wants to express using words, phrases, etc.
  - the idea that is expressed in a work of writing, art, etc.
- Some commonest linguistic ways of thinking of meaning:

signifier (symbol)  $\Leftrightarrow$  signified (idea or thing)

= denotational semantics

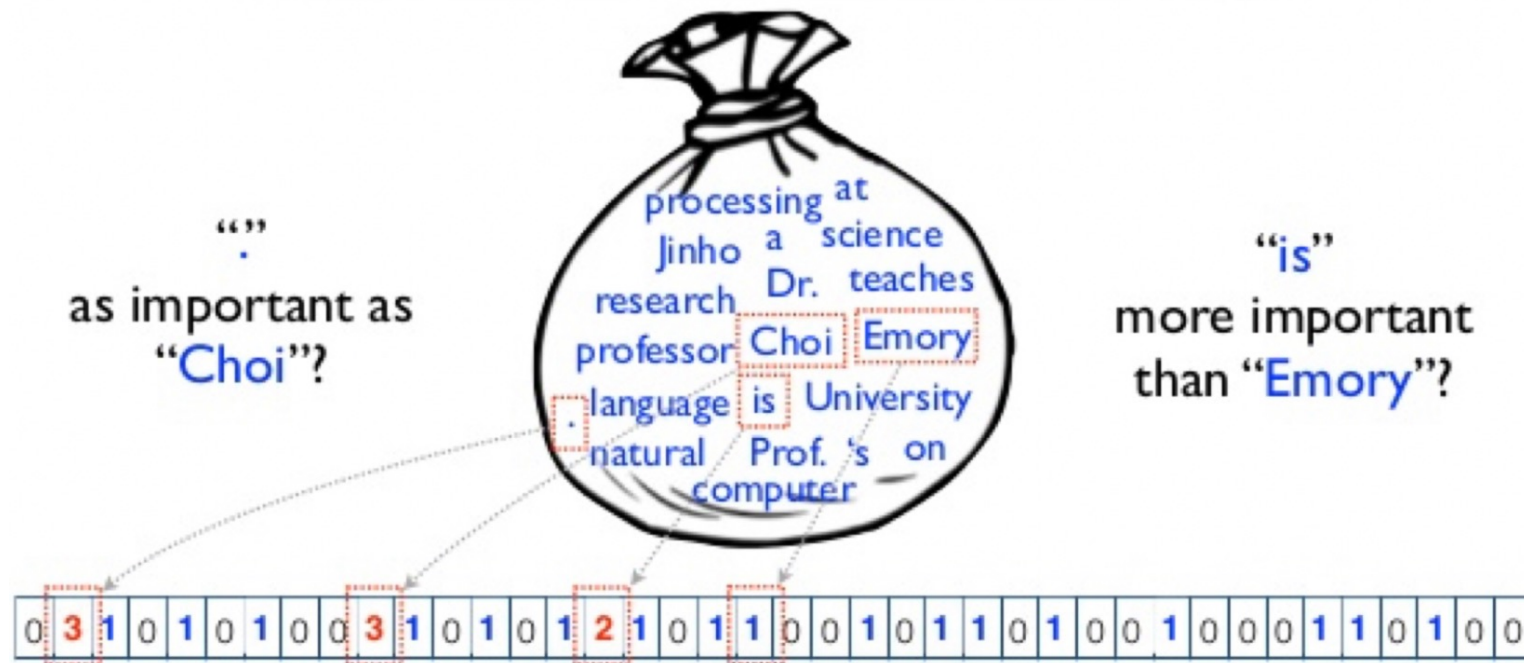
tree  $\Leftrightarrow$  { , , , ... }

# Discrete Vector Representation

- Bag-of-words model
- Co-occurrence matrix
- TF-IDF
- One-hot encoding vector

# Bag-of-words Model

Jinho Choi is a professor at Emory University .  
Prof. Choi teaches computer science .  
Dr. Choi's research is on natural language processing .



# Window-based co-Occurrence Matrix

- Window length = 1  
(more common: 5-10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - 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

# TF-IDF

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$  = number of occurrences of  $i$  in  $j$

$df_i$  = number of documents containing  $i$

$N$  = total number of documents

	Document 1	Document 2	Document 3	Document 4	Document 5	Document 6	Document 7	Document 8
Term(s) 1	10	0	1	0	0	0	0	2
Term(s) 2	0	2	0	0	0	18	0	2
Term(s) 3	0	0	0	0	0	0	0	2
Term(s) 4	6	0	0	4	6	0	0	0
Term(s) 5	0	0	0	0	0	0	0	2
Term(s) 6	0	0	1	0	0	1	0	0
Term(s) 7	0	1	8	0	0	0	0	0
Term(s) 8	0	0	0	0	0	3	0	0

← Word Vector (Passage Vector)

Document Vector



# One-hot Vector

Dictionary D= { I; cat; dog; have; a}

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

One hot vector

Sentence s = "I have a dog"

Region  
Size = 2

1	0	0
0	0	0
0	0	0
0	1	0
0	0	1
0	0	0
0	0	0
0	0	1
1	0	0
0	1	0

2D representation matrix

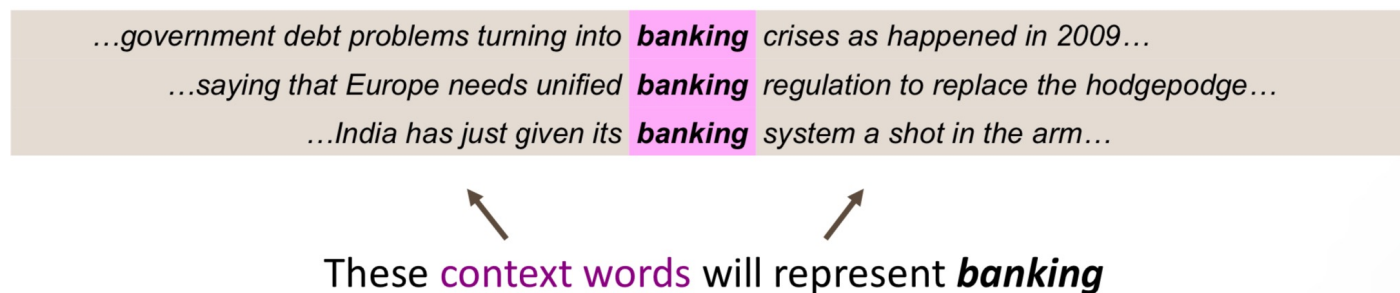
# Problems with words as discrete vectors

- Example: search for the keywords “Seattle Motel” will result also “Seattle Hotel”
- However:  $\text{motel} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]$   
 $\text{hotel} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]$
- These two vectors are orthogonal, without any notion of similarity for one-hot vectors
- Solution: **encode the similarity inside the word vector**

# Word Vector Representation

# Representing word by their context

- Distributional semantic: ***A word's meaning is given by the words that frequently appear close-by***
- When a word  $w$  appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window)
- We will use the many contexts of  $w$  to build up a representation of  $w$



# Word vectors

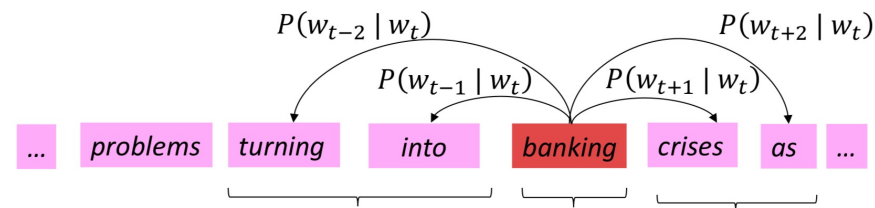
- Each word is represented by a dense vector which is chosen so that it's similar to vectors of words that appear in similar contexts
- The similarity is measured by the vector dot (scalar) product

$$\begin{array}{l} \textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \end{array} \qquad \begin{array}{l} \textit{monetary} = \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix} \end{array}$$

# Word Embedding Model: word2vec

The main idea:

- Start with random vectors
- Iterate through each word position in the whole corpus
- Try to predict surrounding words using word vectors  $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$

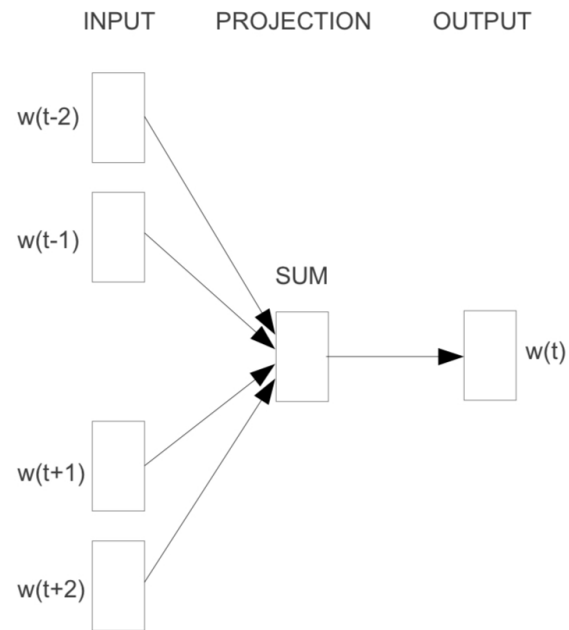


- Learning: update vectors so they can predict actual surrounding words better
- The word embedding model try to learn word vectors that capture well word similarity and meaningful directions in a word space!

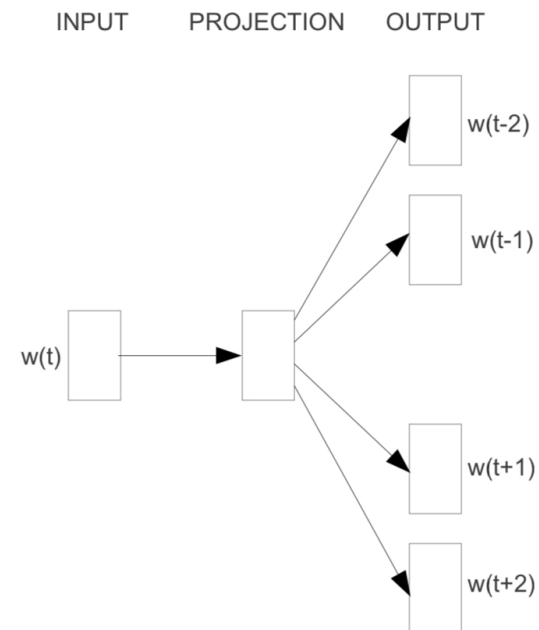
Word2vec maximizes objective function by putting similar words nearby in space



# Two variant approaches for word2vec



**CBOW**



**Skip-gram**




# Skip-gram vs. CBOW

- Skip-gram
  - Predict context ('outside') words (position independent) given center word
- CBOW
  - Predict center word from (bag of) context words

# Skip-gram

- Current word is used as input to a log-linear classifier
- Predict words within certain range before and after of this current word
- The normalization term is computationally expensive as:

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


A big sum over words

- Skip gram model is typically implemented with “**negative sampling**”

# Skip-gram with negative sampling

- Main idea: train binary logistic regression to differentiate a true pair (center word and a word in its context window) versus some “noise” pairs (the center word paired with a random word)
- Maximize the objective function:

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

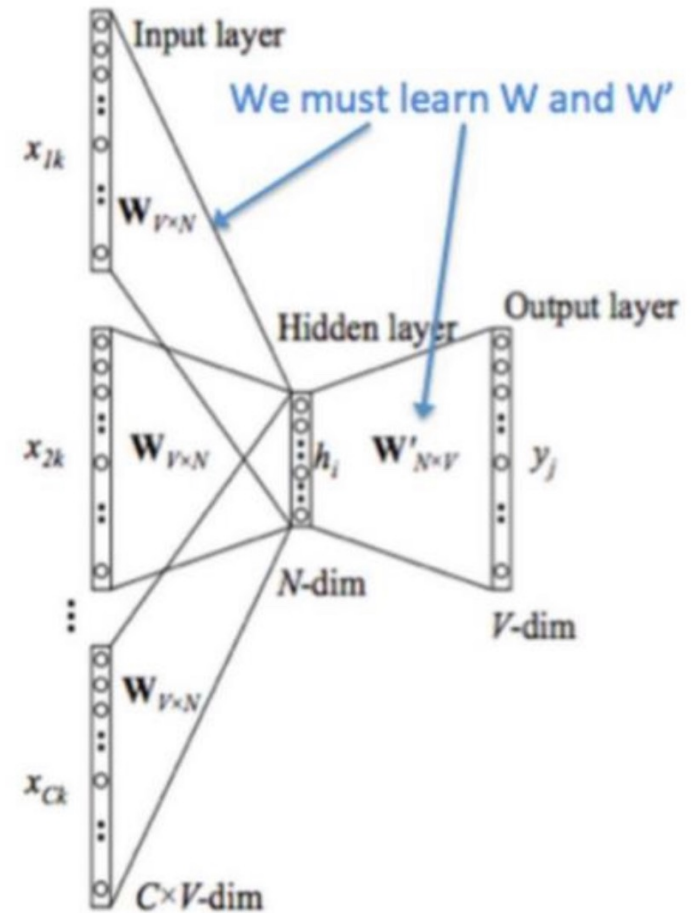
- We take k negative samples, maximize the probability that real outside word appears, minimize the probability that random words appear around center word

# CBOW (Continuous BOW) (1)

- Predict a word using context

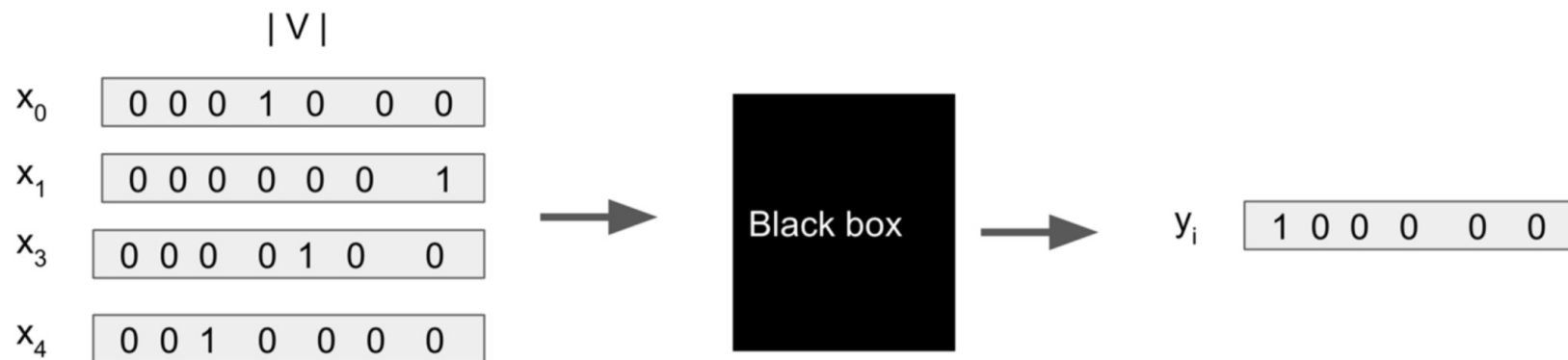
Input :  $x_0, x_1, x_3, x_4$     output :  $x_2$

“ The Cat Chills on a mat ”  
 $x_0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5$



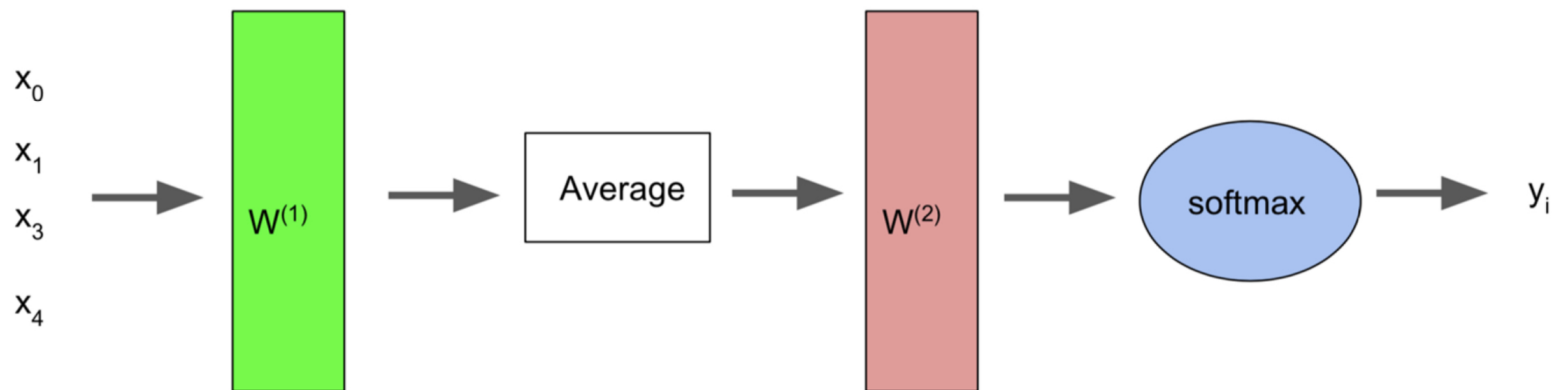
## CBOW (2)

- $|V|$  is the size of the vocabulary corpus
- $x_i$  represents the one-hot vector of the  $i^{\text{th}}$  word
- $y_i$  represents the one-hot vector of the expected word



## CBOW (3)

- $|V|$  is the size of the vocabulary corpus
- $x_i$  represents the one-hot vector of the  $i^{\text{th}}$  word
- $y_i$  represents the one-hot vector of the expected word



## CBOW (4)

$y_i$ 

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

$Z$ 

32	14	23	0.22	2	14	55	19
----	----	----	------	---	----	----	----

$y^\wedge$ 

0.7	0.1	0.02	0.08	0	0	0.1
-----	-----	------	------	---	---	-----

$$y^\wedge = \text{softmax} ( Z ) \qquad \sigma(\mathbf{z})_j = \frac{e^{\tilde{z}_j}}{\sum_{k=1}^K e^{\tilde{z}_k}}$$

$y_i \in \mathbb{R}^{|V| \times 1}$  is the one-hot vector of the expected word

# GLOVE – Global vector for word representation

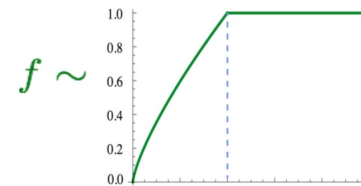
- 2 assumptions:
  - Global context (global matrix factorization)
    - Co-occurrence of words through documents
  - Local context
    - A pre-fixed size slide window
- Idea: Encoding meaning components in vector differences

A: Log-bilinear model:  $w_i \cdot w_j = \log P(i|j)$

with vector differences  $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$

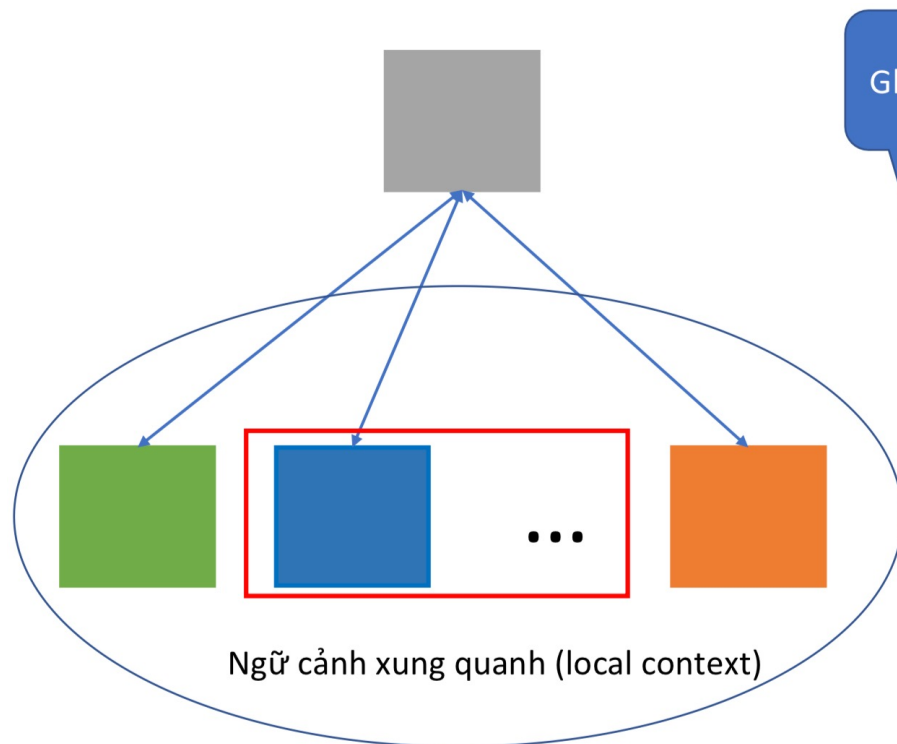
Loss: 
$$J = \sum_{i,j=1}^V f(X_{ij}) (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

- Fast training
- Scalable to huge corpora





# Word2Vec vs. Glove



Global context

Local context

$X =$

	<i>I</i>	<i>like</i>	<i>enjoy</i>	<i>deep</i>	<i>learning</i>	<i>NLP</i>	<i>flying</i>	<i>.</i>
<i>I</i>	0	2	1	0	0	0	0	0
<i>like</i>	2	0	0	1	0	1	0	0
<i>enjoy</i>	1	0	0	0	0	0	1	0
<i>deep</i>	0	1	0	0	1	0	0	0
<i>learning</i>	0	0	0	1	0	0	0	1
<i>NLP</i>	0	1	0	0	0	0	0	1
<i>flying</i>	0	0	1	0	0	0	0	1
<i>.</i>	0	0	0	0	1	1	1	0

Ma trận đồng xuất hiện

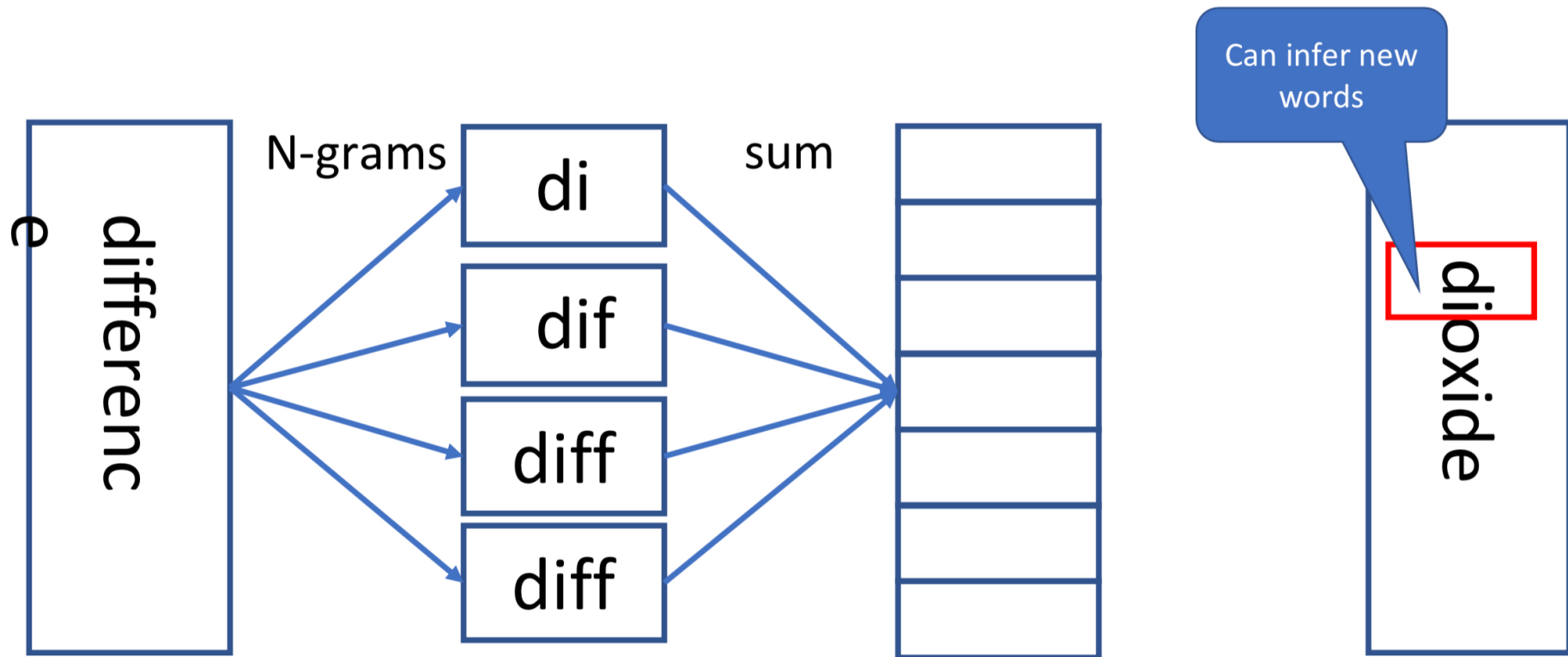
# Pros and cons

- Pros:
  - Fast training
  - Well scaled on big corpus data
  - Work well on small data
  - Early stopping
- Cons
  - Memory consuming
  - Learning rate may affect the accuracy of the model

# Fast Text

- An extension of Word2vec
- Facebook
- Multi-language support
- Sub-word
  - Based on n-grams model
  - Allow to short word learning
  - Allow to represent prefixes and postfixes of word
- Work well with rare-words

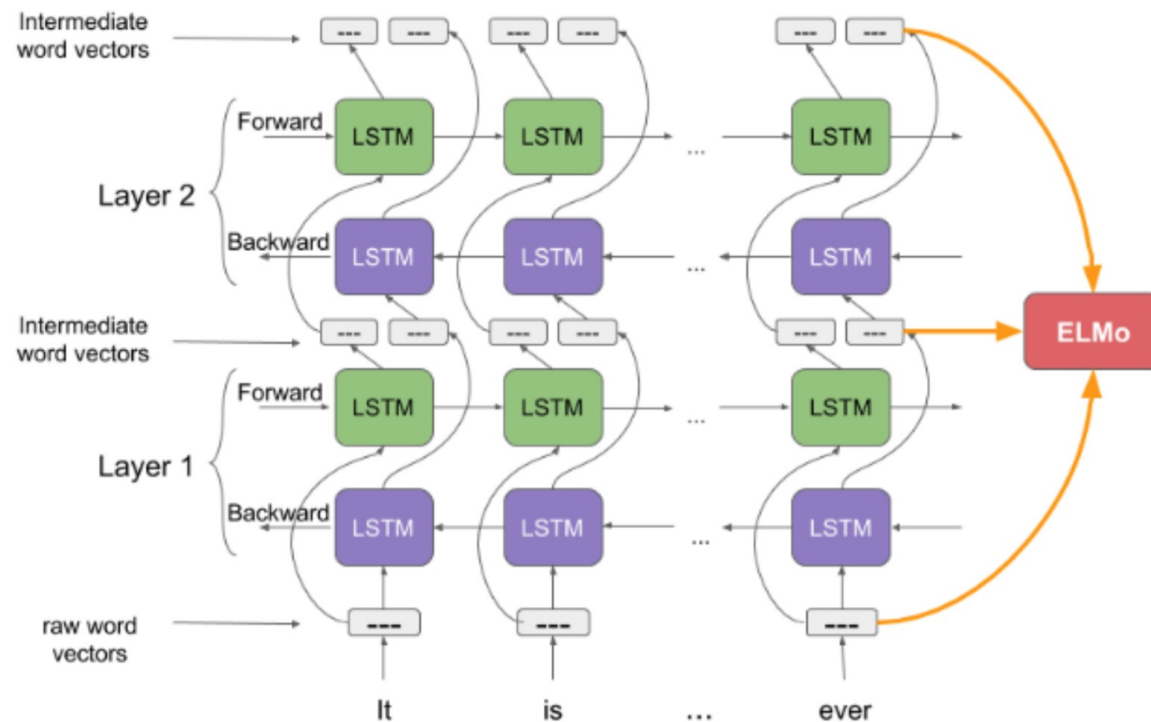
# Word vector of Fast text



# Pros and cons

- Capture rare-words thanks to sub-word representation
- Memory consuming for sub-word representations

# ELMo – Embedding from Language Model



<https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/>

# ELMo - Architecture

- Using character-level CNN to represent word (raw word vectors)
- Bi-directional LSTM
- Combine forward and backward directions to represent intermediate word vectors
- Second layer using intermediate word vectors as input with the same architecture as first layer
- Final word vector:
  - Combine first representation (raw word vector) and two outputs of two layers
    - ELMo vector

# Pros and cons

- Better capture the context of word than word2vec and Glove

I *read* the book yesterday vs. Can you *read* the letter now?

- The same words may have the different representation depending on the context
- Rare-word representation (as Fast Text)
- Memory consuming

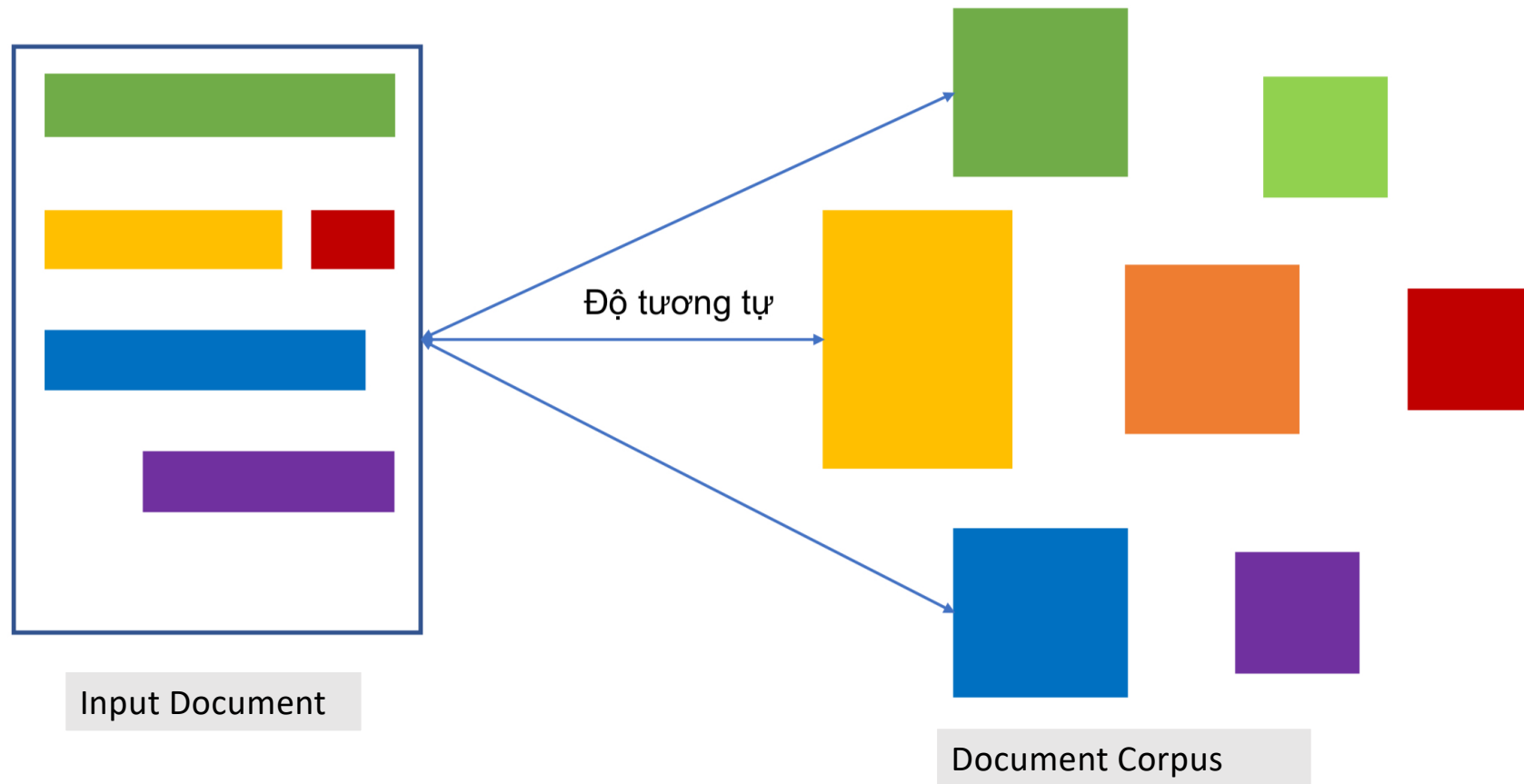


# Building word vector model

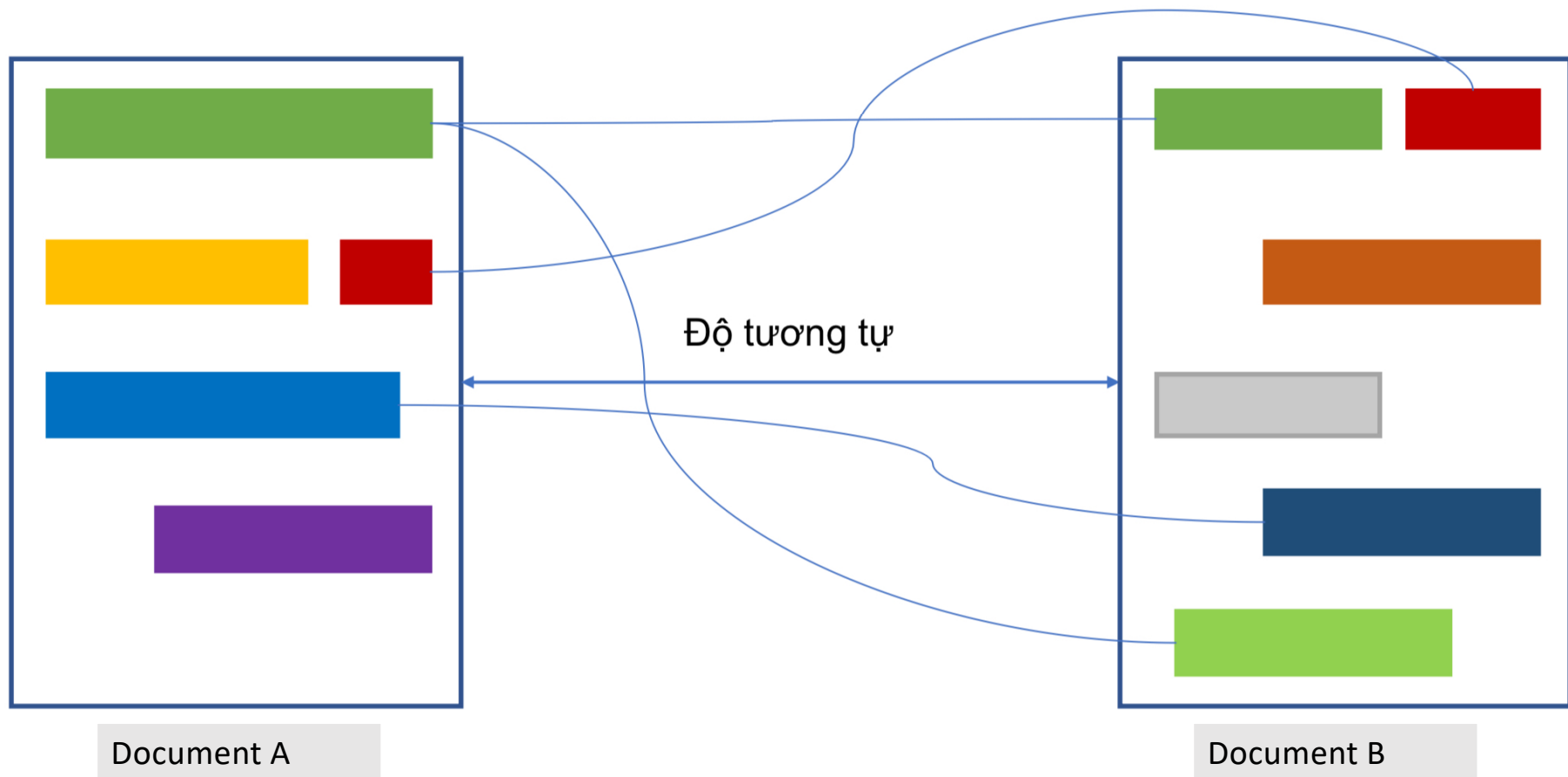
- A large text corpus
  - Wikipedia
- Choose an appropriate model
- Training model
- Using the model to solve problems of NLP

# Application - Plagiarism Detection

# Problem of Plagiarism Detection



# Similarity Comparison



# Challenges

- Comparing the similarity of different parts between two documents is much more difficult
- Long document
- Structure and vocabulary
- Focus on semantics and syntactics

# Summary

- Text Vectorization
- Word Embedding
- Word Embedding models
  - Word2vec
  - Glove
  - Fast Text
  - EMLo
- Building a model for word representation