NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 11 : Word Representation

PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- How to represent a word? Distributional Hypothesis
- Sparse vs Dense Vectors
- Reasoning with word vectors

How to represent a word?

In traditional NLP / IR, words are treated as discrete symbols.

One-hot representation

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

Vector dimension = number of words in vocabulary (e.g., 500,000)

Problems with words as discrete symbols

Example: In web search, if user searches for “Baltimore motel”, we would like to match documents containing “Baltimore hotel”. But

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

The vectors are orthogonal, and there is no natural notion of similarity between one-hot vectors!

- ***How can we define the word meaning that encodes this notion of similarity?***
- ***Can we use shorter vectors?***

Ludwig Wittgenstein

"The meaning of a word is its use in the language"

Source: https://web.stanford.edu/~jurafsky/slp3 .

Distributional Hypothesis: Let's define words by their usages

One way to define "usage":

words are defined by their environments (the words around them)

Zellig Harris (1954):

If A and B have almost identical environments we say that they are synonyms.

Source: https://web.stanford.edu/~jurafsky/slp3 .

What does recent English borrowing *ongchoi* mean?

Suppose you see these sentences:

- Ongchoi is delicious **sautéed with garlic**.
- Ongchoi is superb **over rice**
- Ongchoi **leaves** with salty sauces

And you've also seen these:

- …spinach **sautéed with garlic over rice**
- Chard stems and **leaves** are **delicious**
- Collard greens and other **salty** leafy greens

Conclusion:

- Ongchoi is a leafy green like spinach, chard, or collard greens
  - We could conclude this based on words like "leaves" and "delicious" and "sauteed"

Source: https://web.stanford.edu/~jurafsky/slp3 .

Ongchoi: *Ipomoea aquatica "Water Spinach"*

空心菜
kangkong
rau muống
…

Yamaguchi, Wikimedia Commons, public domain

Source: https://web.stanford.edu/~jurafsky/slp3 .

Defining meaning as a point in space based on distribution

Each word = a vector

Similar words are "**nearby in semantic space**"

We build this space automatically by seeing which words are **nearby in text**

Source: https://web.stanford.edu/~jurafsky/slp3 .

We define meaning of a word as a vector

Called an "embedding" because it's embedded into a space

The standard way to represent meaning in NLP

Modern NLP algorithms use embeddings as the representation of word meaning

Fine-grained model of meaning for similarity

Source: https://web.stanford.edu/~jurafsky/slp3 .

Intuition: why vectors?

Consider classical algorithms:

- With **words**, a feature is a word identity
  - Feature 5: 'The previous word was "terrible"'
  - requires **exact same word** to be in training and test (*dreadful*?)
- With **embeddings**:
  - Feature is a word vector
  - 'The previous word was vector [35,22,17…]
  - Now in the test set we might see a similar vector [34,21,14]
  - We can generalize to **similar but unseen** words!!!

Source: https://web.stanford.edu/~jurafsky/slp3 .

Encoding similarity: Distributional Vectors

Informal algorithm for constructing word spaces

- Pick the words you are interested in: **target words**
- Define a **context window**, number of words surrounding target word
  - The context can in general be defined in terms of documents, paragraphs or sentences.
- Count number of times the target word co-occurs with the context words: **co-occurrence matrix**
- Build vectors out of (a function of) these co-occurrence counts

Let's take an example

Small Dataset

An automobile is a wheeled motor vehicle used for transporting passengers .
A car is a form of transport , usually with four wheels and the capacity to carry around five passengers .
Transport for the London games is limited , with spectators strongly advised to avoid the use of cars .
The London 2012 soccer tournament began yesterday , with plenty of goals in the opening matches .
Giggs scored the first goal of the football tournament at Wembley , North London .
Bellamy was largely a passenger in the football match , playing no part in either goal .

distributional matrix = targets X contexts

	wheel	transport	passenger	tournament	London	goal	match
automobile	1	1	1	0	0	0	0
car	1	2	1	0	1	0	0
soccer	0	0	0	1	1	1	1
football	0	0	1	1	1	2	1

Distributional Vectors: Computing Similarity

	wheel	transport	passenger	tournament	London	goal	match
automobile	1	1	1	0	0	0	0
car	1	2	1	0	1	0	0
soccer	0	0	0	1	1	1	1
football	0	0	1	1	1	2	1

Using simple vector product

automobile . car = 4

automobile . soccer = 0

automobile . football = 1

car . soccer = 1

car . football = 2

soccer . football = 5

Context Weighting via association measures

basic intuition

word1	word2	freq(1,2)	freq(1)	freq(2)
dog	small	855	33,338	490,580
dog	domesticated	29	33,338	918

Association measures are used to give more weight to contexts that are more significantly associted with a targer word.

- The less frequent the target and context element are, the higher the weight given to their co-occurrence count should be.
  $\Rightarrow$ Co-occurrence with frequent context element *small* is less informative than co-occurrence with rarer *domesticated*.
- different measures - e.g., Mutual information, Log-likelihood ratio

Pointwise Mutual Information (PMI)

$$PMI(w_1, w_2) = log_2 \frac{P_{corpus}(w_1, w_2)}{P_{corpus}(w_1) P_{corpus}(w_2)}$$

$$P_{corpus}(w_1, w_2) = \frac{freq(w_1, w_2)}{N} \qquad P_{corpus}(w) = \frac{freq(w)}{N}$$

Statistically measure whether two words co-occur frequently (relative to their global frequencies)

Even simpler: Can use documents as context

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
against	0	0	0	1	0	0	3	2	3	0
age	0	0	0	1	0	3	1	0	4	0
agent	0	0	0	0	0	0	0	0	0	0
ages	0	0	0	0	0	2	0	0	0	0
ago	0	0	0	2	0	0	0	0	3	0
agree	0	1	0	0	0	0	0	0	0	0
ahead	0	0	0	1	0	0	0	0	0	0
ain't	0	0	0	0	0	0	0	0	0	0
air	0	0	0	0	0	0	0	0	0	0
aka	0	0	0	1	0	0	0	0	0	0

Context weighting: documents as context

Indexing function F: Essential factors

- **Word frequency (f_{ij}):** How many times a word appears in the document? $F \propto f_{ij}$
- **Document length ($|D_i|$):** How many words appear in the document? $F \propto \frac{1}{|D_i|}$
- **Document frequency (N_j):** Number of documents in which a word appears. $F \propto \frac{1}{N_j}$

Indexing Weight: tf-Idf

- $f_{ij} * log(\frac{N}{N_j})$ for each term, normalize the weight in a document with respect to L_2-norm.

Sparse versus dense vectors

One-hot (or distributional) vectors (*includes tf-idf, PMI, etc.*)

- **long** (length |V|= 20,000 to 50,000)
- **sparse** (most elements are zero)

Alternative: learn vectors which are

- **short** (length 50-1000)
- **dense** (most elements are non-zero)

Source: https://web.stanford.edu/~jurafsky/slp3 .

Sparse versus dense vectors

Why dense vectors?

- Short vectors may be easier to use as **features** in machine learning (fewer weights to tune)
- Dense vectors may **generalize** better than explicit counts
- Dense vectors may do better at capturing synonymy:
  - *car* and *automobile* are synonyms; but are distinct dimensions
    - a word with *car* as a neighbor and a word with *automobile* as neighbor should be similar, but aren't
- **In practice, they work better**

Source: https://web.stanford.edu/~jurafsky/slp3 .

Common methods for getting short dense vectors

Singular Value Decomposition (SVD)

- A special case of this is called LSA – Latent Semantic Analysis

“Neural Language Model”-inspired models

- **Word2vec (skipgram, CBOW), GloVe**

Alternative to these "static embeddings":

- Contextual Embeddings (ELMo, BERT)
- Compute distinct embeddings for a word in its context
- Separate embeddings for each token of a word

Source: https://web.stanford.edu/~jurafsky/slp3 .

Word2vec

Instead of **counting** how often each word *w* occurs near "*apricot*"

- Train a classifier on a binary **prediction** task:
  - Is *w* likely to show up near "*apricot*"?

We don't actually care about this task

- But we'll take the learned classifier weights as the word embeddings

Big idea: **self-supervision**:

- A word c that occurs near apricot in the corpus acts as the gold "correct answer" for supervised learning
- No need for human labels
- Bengio et al. (2003); Collobert et al. (2011)

Source: https://web.stanford.edu/~jurafsky/slp3 .

Distributional Representation

- Take a vector with several hundred dimensions (say 1000).
- Each word is represented by a distribution of weights across those elements.
- So instead of a one-to-one mapping between an element in the vector and a word, the representation of a word is spread across all of the elements in the vector, and
- Each element in the vector contributes to the definition of many words.

Reasoning with Word Vectors

- It has been found that the learned word representations in fact capture meaningful syntactic and semantic regularities in a very simple way.
- Specifically, the regularities are observed as constant vector offsets between pairs of words sharing a particular relationship.

Case of Singular-Plural Relations

If we denote the vector for word i as x_i, and focus on the singular/plural relation, we observe that

$$x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families} \approx x_{car} - x_{cars}$$

and so on.

Reasoning with Word Vectors

Perhaps more surprisingly, we find that this is also the case for a variety of semantic relations.

Good at answering analogy questions

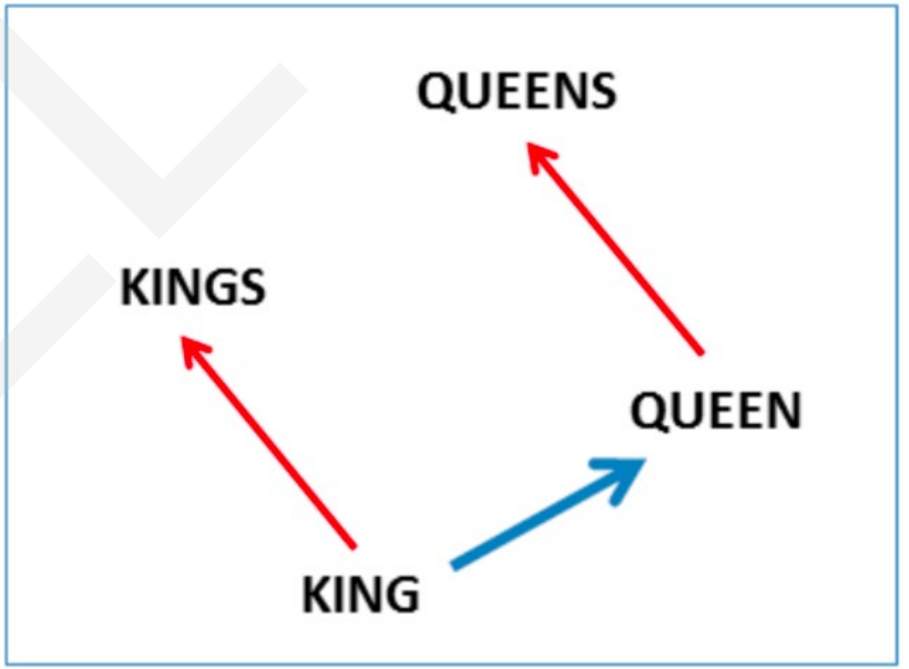
a is to b, as c is to ?

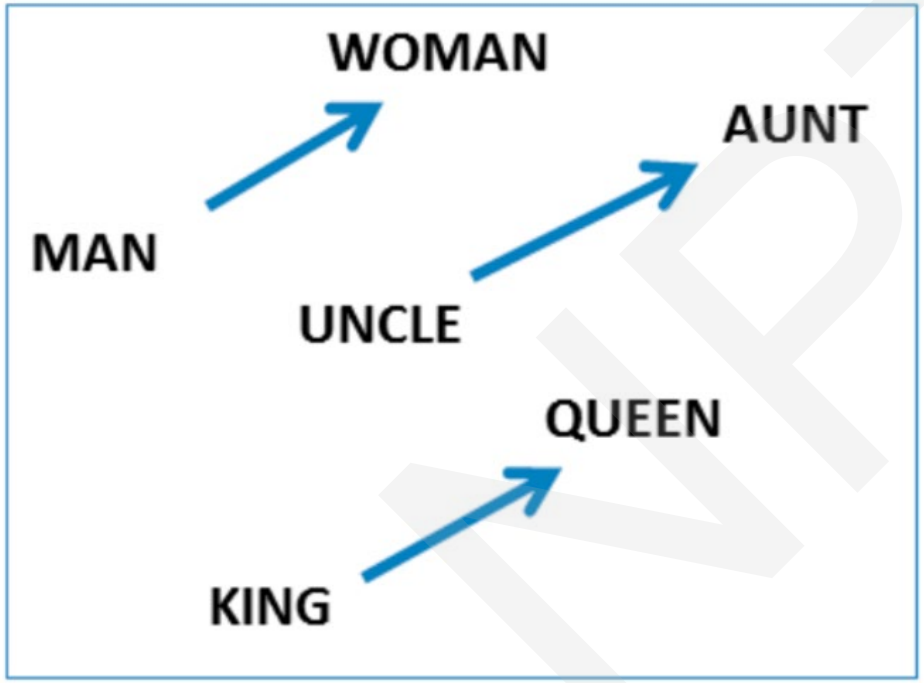
man is to *woman* as *uncle* is to ? (*aunt*)

A simple vector offset method based on cosine distance shows the relation.

Vector Offset for Relations

Singular - Plural





Gender

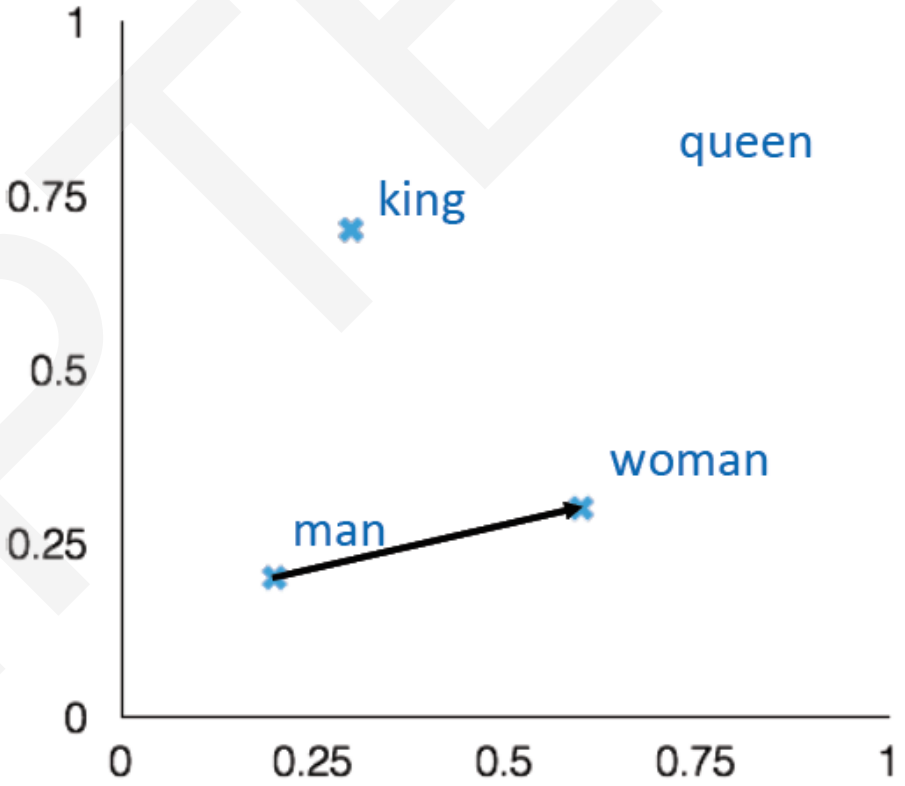
Analogy Testing

a:b :: c:?

$$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



REFERENCES

Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. [Chapter 6]


THANK YOU

FREE ONLINE EDUCATION
swayam
शिक्षित भारत, उन्नत भारत

NPTEL

NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 12 : Learning Word Representation - Part 1

PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

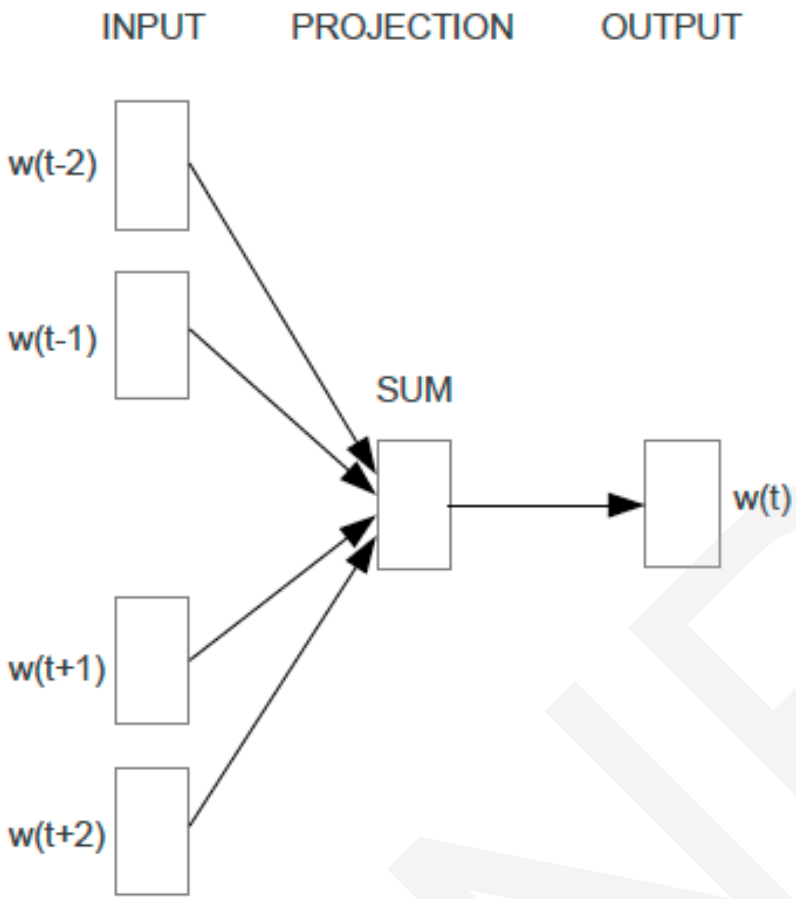
- Learning Word Vectors - Overview
- Objective Function
- Example Problem, deriving the gradients

Learning Word Vectors: Overview

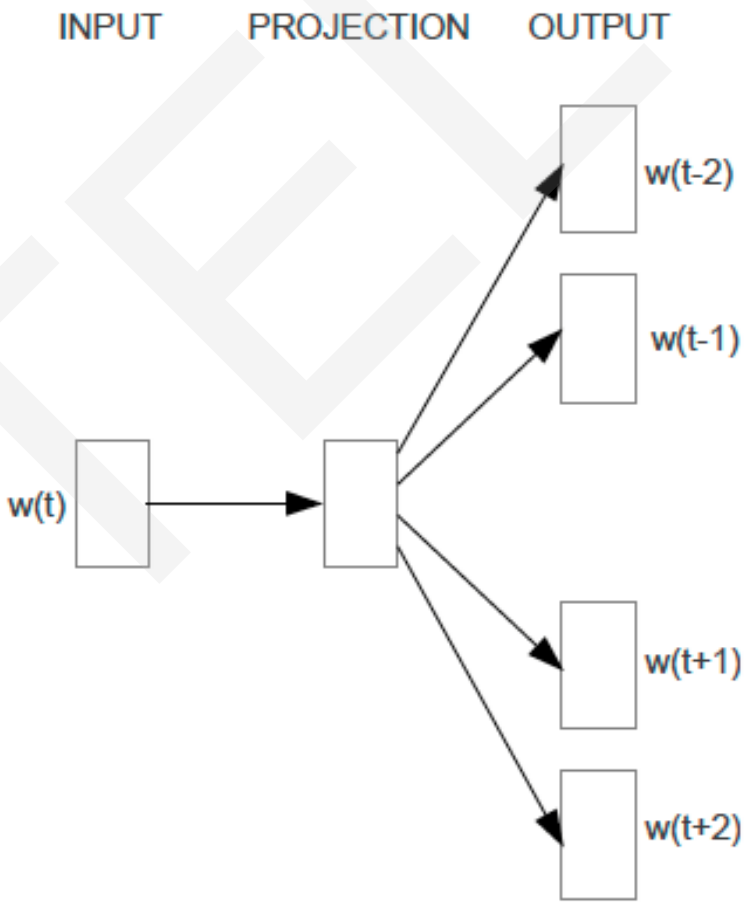
Basic Idea

- We have a large corpus of text
- Every word in a fixed vocabulary is represented by a *vector*
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability

Two Variations: CBOW and Skip-grams



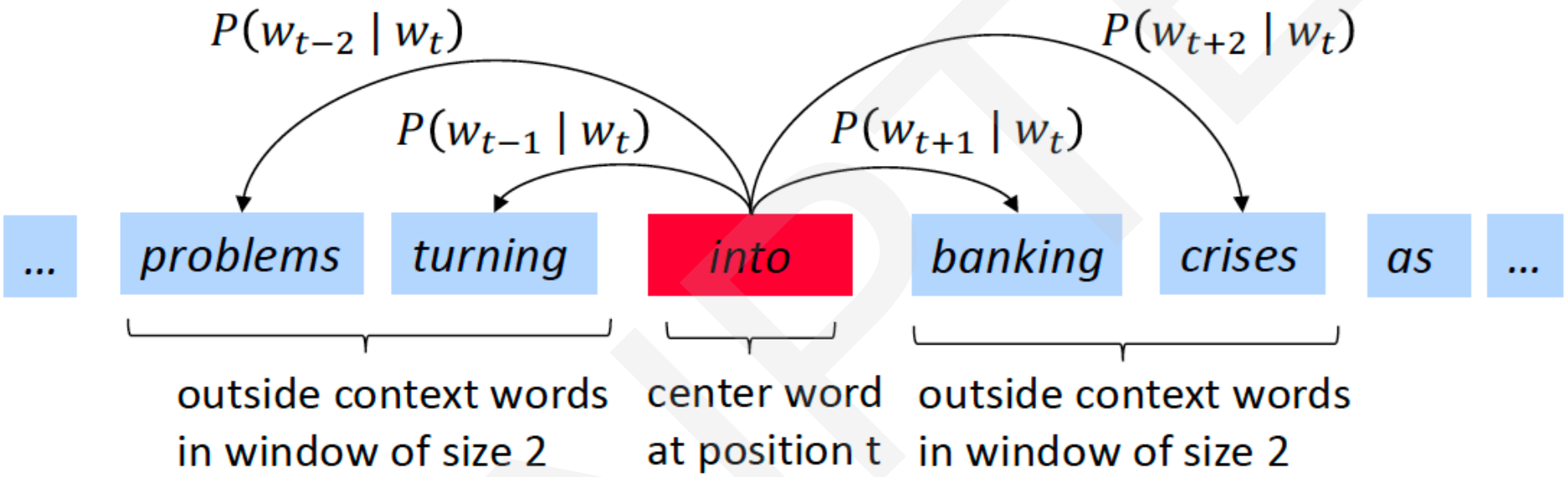
CBOW



Skip-gram

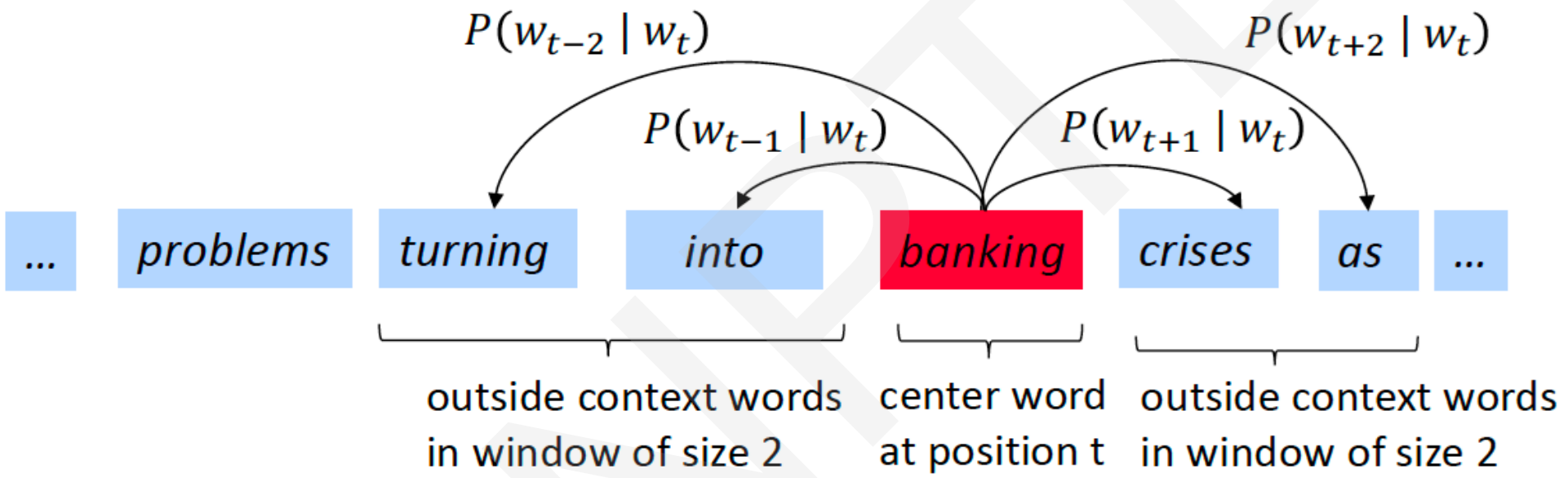
Word2Vec (Skip-gram) Overview

Example windows and process for computing $P(w_{t+j}|w_t)$



Word2Vec (Skip-gram) Overview

Example windows and process for computing $P(w_{t+j}|w_t)$



Word2Vec: objective function

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

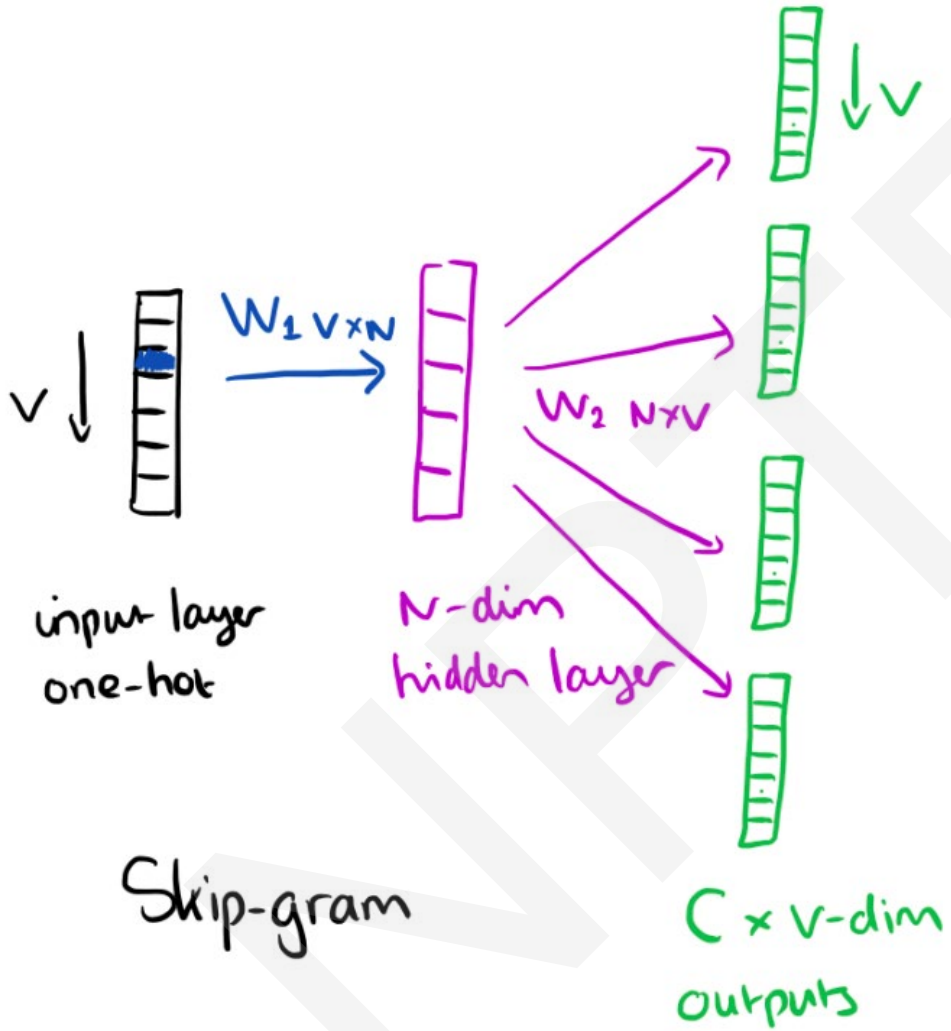
We will use two vectors per word w:

- v_w when w is a center word
- u_w when w is a context word

Then, for a center word c and a context word o

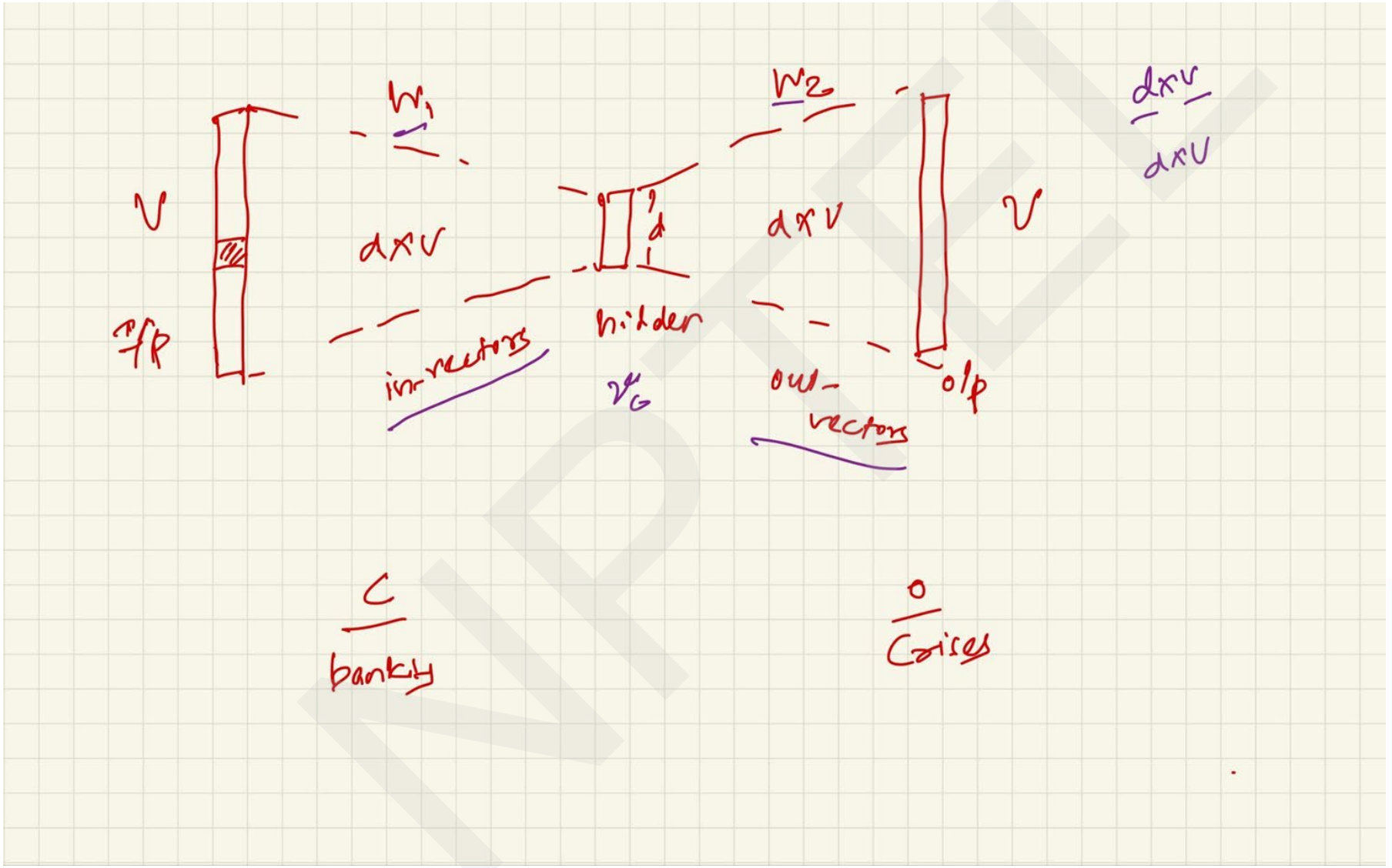
$$P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w \in V} exp(u_w^T v_c)}$$

Understanding P(o|c) further

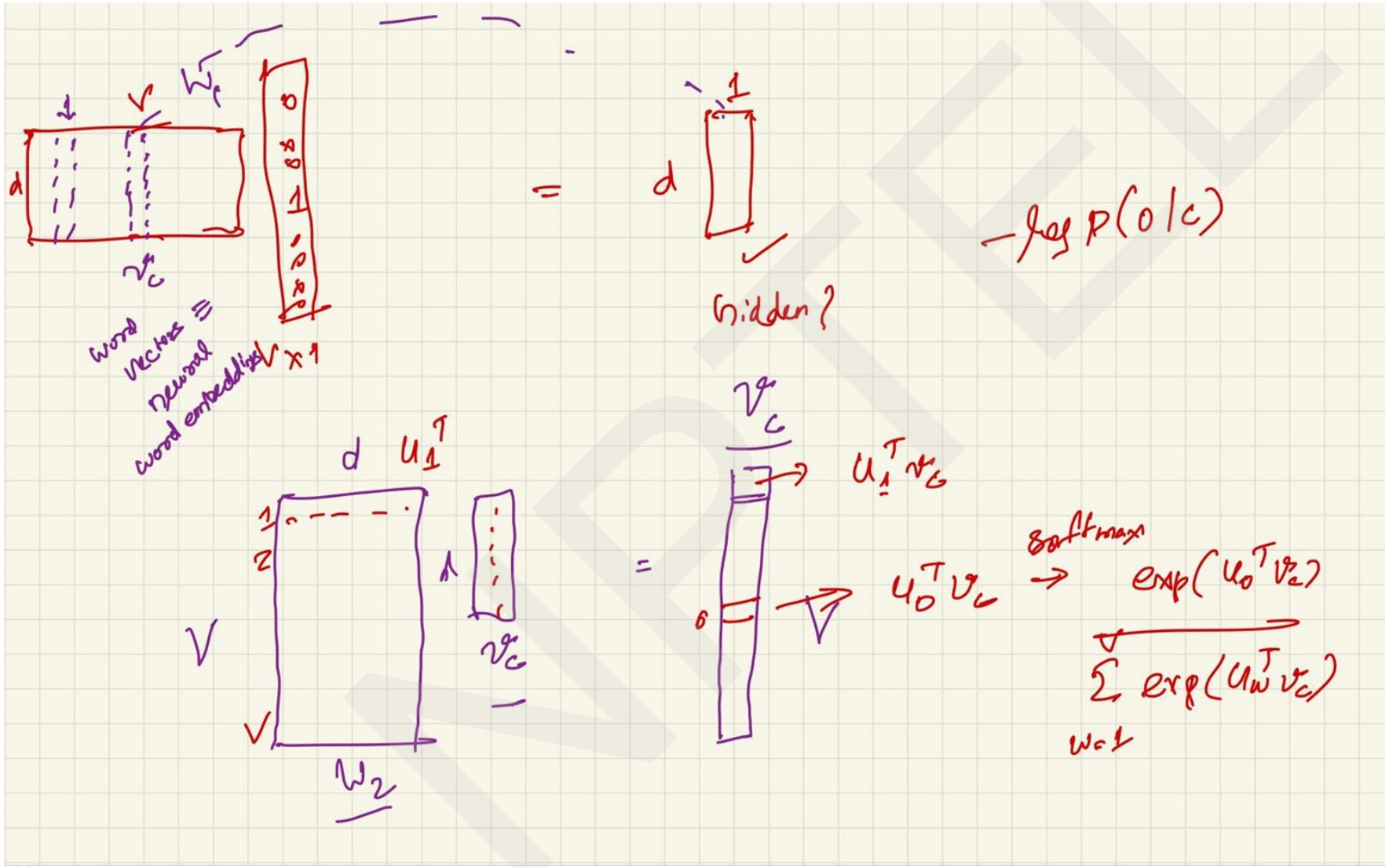


$$P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w \in V} exp(u_w^T v_c)}$$

More Explanation



More Explanation



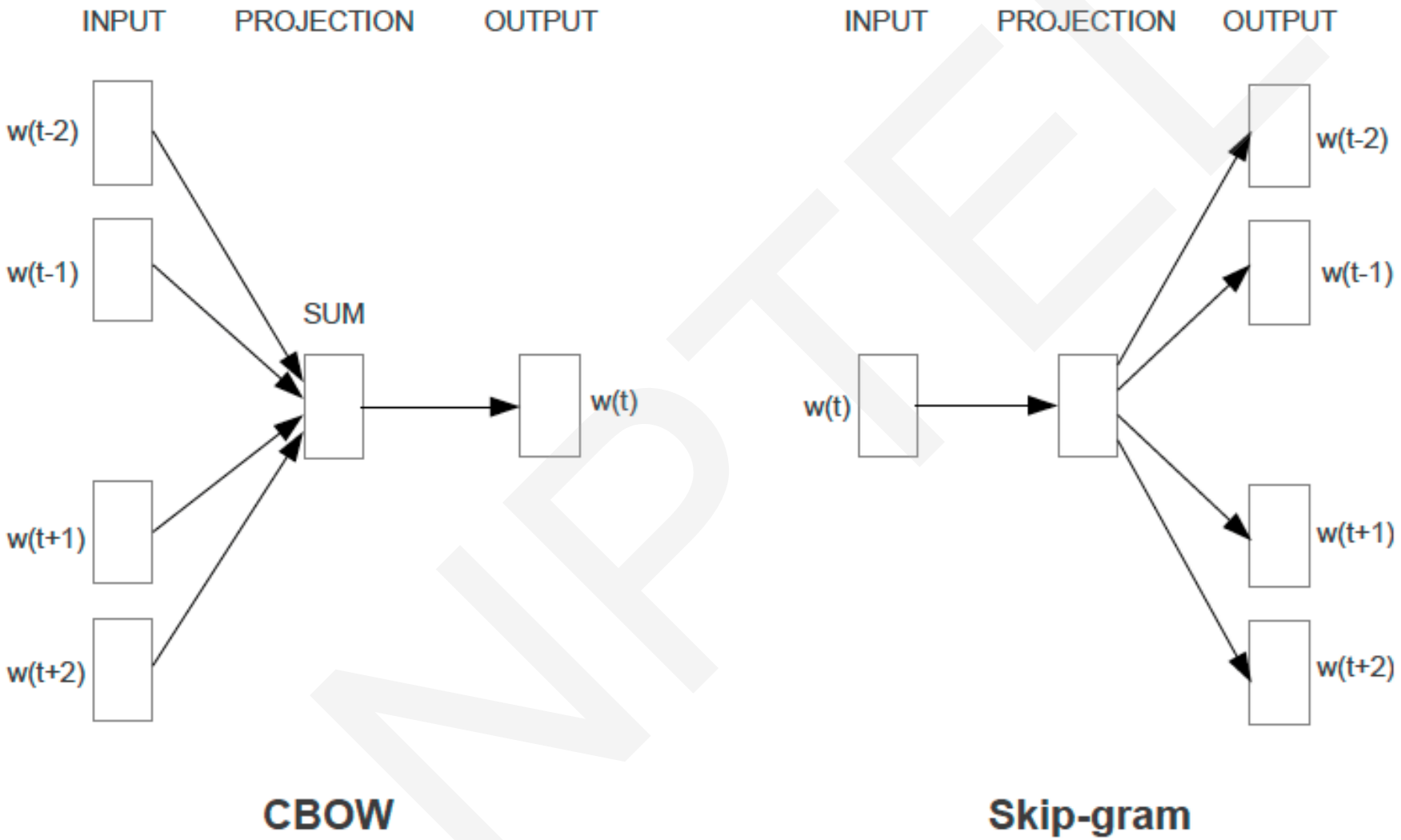
Learning Word Vectors

Learning Word Vectors: Overview

Basic Idea

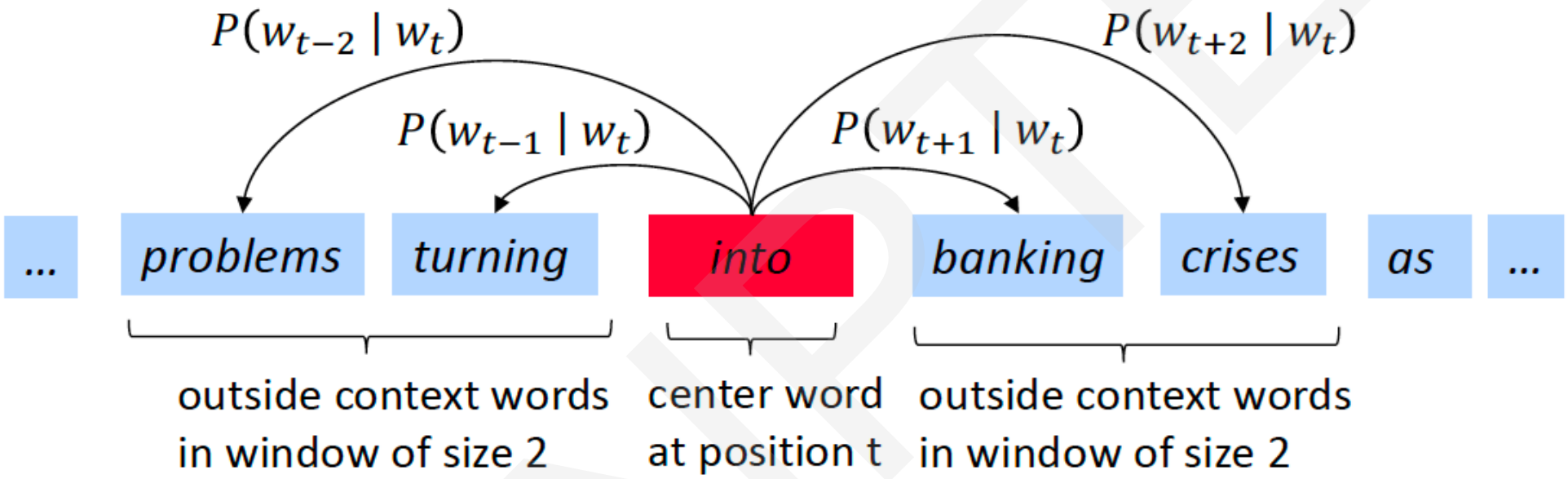
- We have a large corpus of text
- Every word in a fixed vocabulary is represented by a *vector*
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability

Two Variations: CBOW and Skip-grams



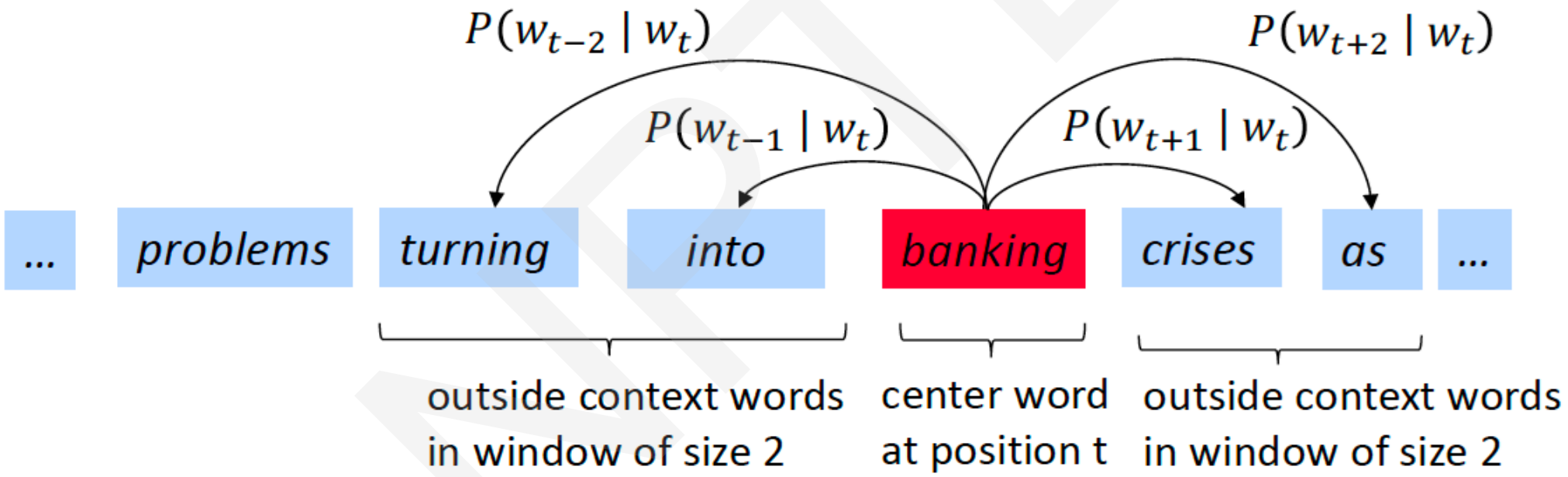
Word2Vec (Skip-gram) Overview

Example windows and process for computing $P(w_{t+j}|w_t)$



Word2Vec (Skip-gram) Overview

Example windows and process for computing $P(w_{t+j}|w_t)$



Word2Vec: objective function

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

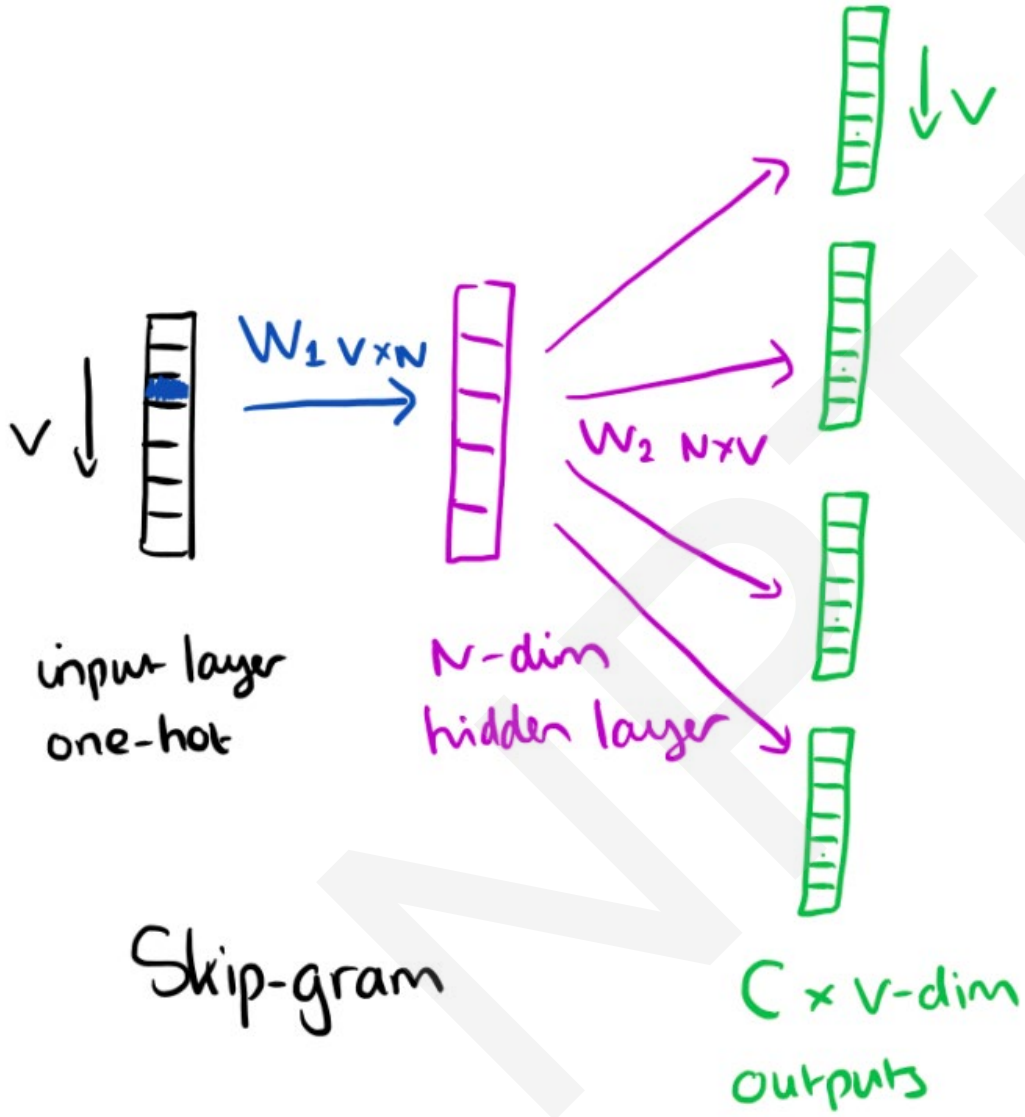
We will use two vectors per word w:

- v_w when w is a center word
- u_w when w is a context word

Then, for a center word c and a context word o

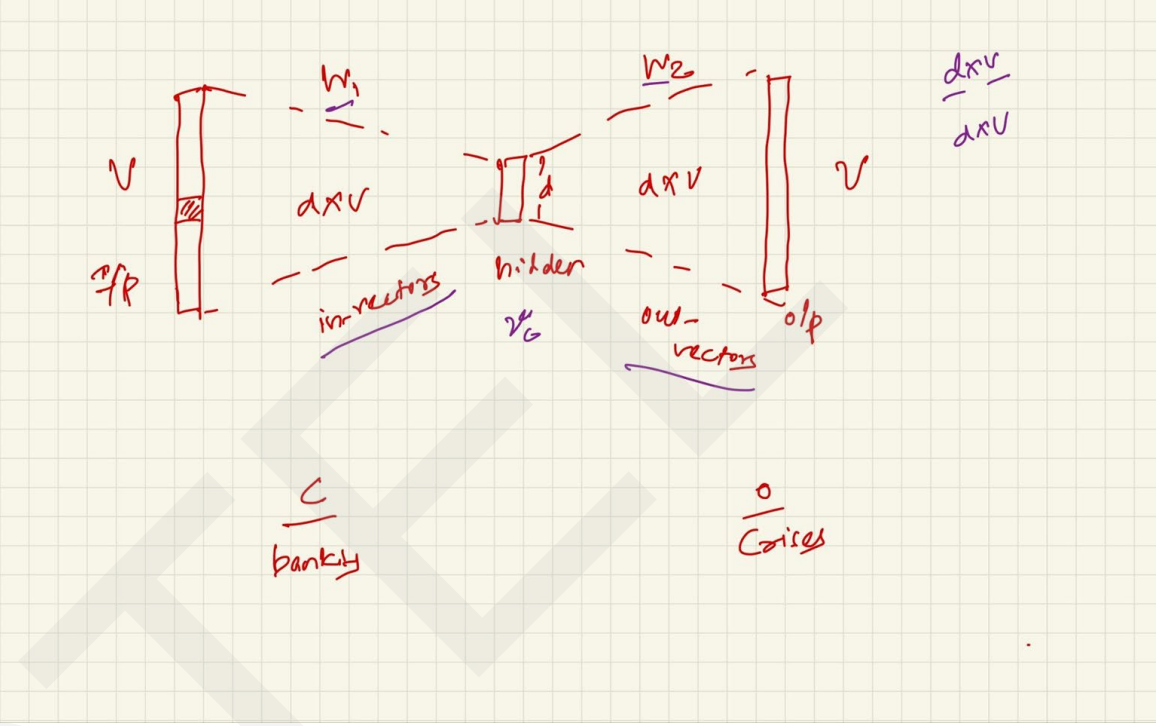
$$P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w \in V} exp(u_w^T v_c)}$$

Understanding P(o|c) further

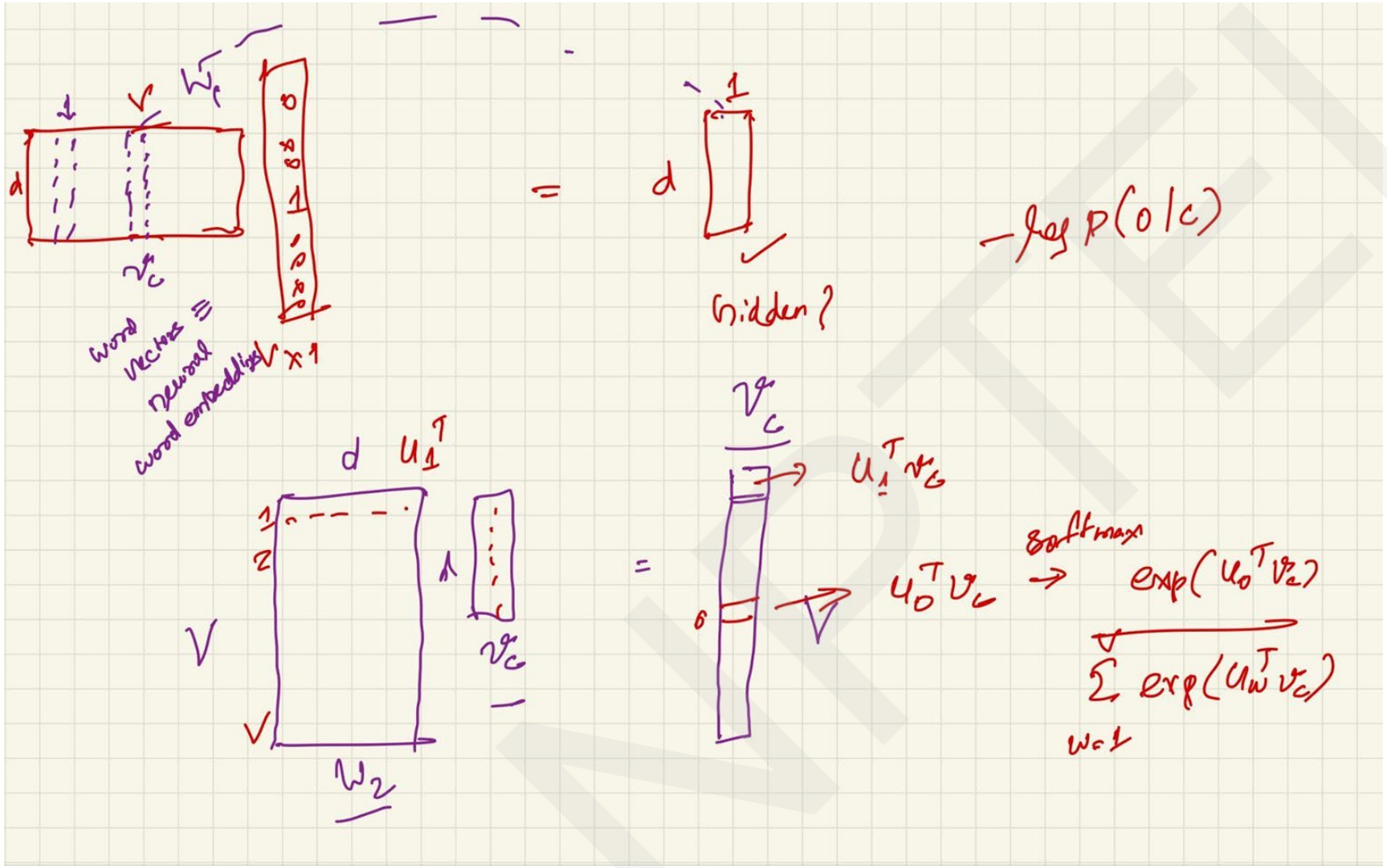


$$P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w \in V} exp(u_w^T v_c)}$$

More Explanation



More Explanation



Try this problem

Skip-gram

Suppose you are computing the word vectors using Skip-gram architecture. You have 5 words in your vocabulary, $\{passed, through, relu, activation, function\}$ in that order and suppose you have the window, '*through relu activation*' in your corpora. You use this window with 'relu' as the center word and one word before and after the center word as your context.

Compute the loss

Also, suppose that for each word, you have 2-dim in and out vectors, which have the same value at this point given by [1,-1],[1,1],[-2,1],[0,1],[1,0] for the 5 words, respectively. As per the Skip-gram architecture, the loss corresponding to the target word "activation" would be $-log(x)$. What is the value of x?

Try this problem

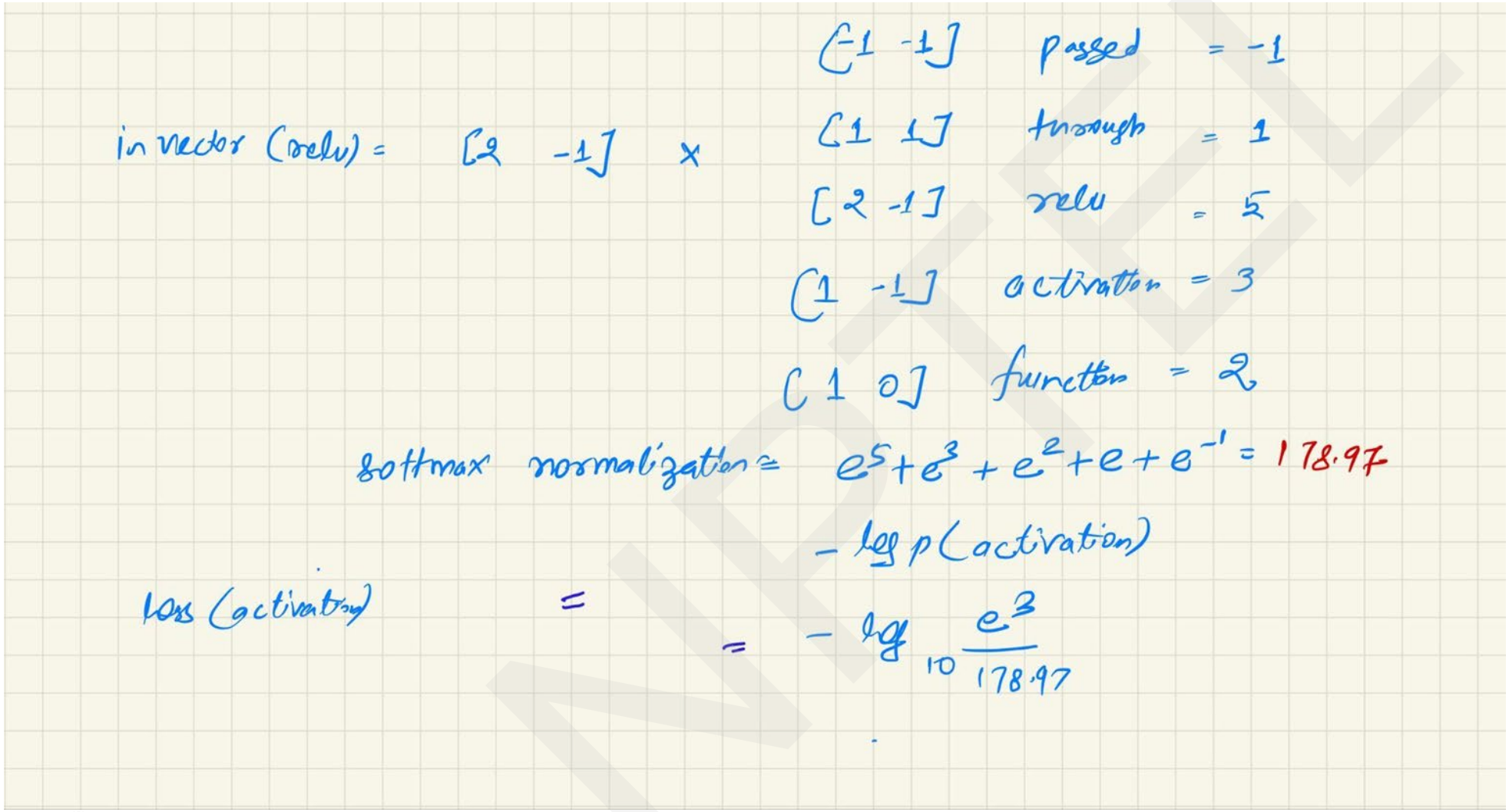
Skip-gram

Suppose you are computing the word vectors using Skip-gram architecture. You have 5 words in your vocabulary, $\{passed, through, relu, activation, function\}$ in that order and suppose you have the window, '*through relu activation*' in your corpora. You use this window with 'relu' as the center word and one word before and after the center word as your context.

Compute the loss

Also, suppose that for each word, you have 2-dim in and out vectors, which have the same value at this point given by [1,-1],[1,1],[-2,1],[0,1],[1,0] for the 5 words, respectively. As per the Skip-gram architecture, the loss corresponding to the target word "activation" would be $-log(x)$. What is the value of x?

Solution


in vector (relu) = [2 -1] x
[-1 -1] passed = -1
[1 1] through = 1
[2 -1] relu = 5
[1 -1] activation = 3
[1 0] function = 2
softmax normalization = $e^5+e^3+e^2+e+e^{-1} = 178.97$
loss (activation) = $-\log p(\text{activation})$
$= -\log_{10} \frac{e^3}{178.97}$

Gradient Descent for Parameter Updates

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

Training the model

Compute **all** vector gradients.

- θ represents all model parameters: in our case, V-many words, 2 $d-$ dimensional vectors for each word
- We optimize these parameters by walking down the gradient

Training the model

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

For *one example window* and *one example outside word*:

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

Try deriving the gradients for the center word v_c and the outside word u_o.

Source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture01-wordvecs1.pdf

Objective Function

Maximize $J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} p(w'_{t+j} | w_t; \theta)$

Or minimize ave. neg. log likelihood

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log p(w'_{t+j} | w_t)$$

[negate to minimize; log is monotone]

T: text length

window size

where

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

word IDs

We now take derivatives to work out minimum

Each word type (vocab entry) has two word representations: as center word and context word

Source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture01-wordvecs1.pdf

$$\frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w v_c)}$$

$$= \underbrace{\frac{\partial}{\partial v_c} \log \exp(u_o^T v_c)}_{①} - \underbrace{\frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp(u_w^T v_c)}_{②}$$

① $\frac{\partial}{\partial v_c} \log \exp(u_o^T v_c) = \frac{\partial}{\partial v_c} u_o^T v_c = u_o$

Vector! Not high school single variable calculus

inverses

You can do things one variable at a time, and this may be helpful when things get gnarly.

$$\forall j \quad \frac{\partial}{\partial (v_c)_j} u_o^T v_c = \frac{\partial}{\partial (v_c)_j} \sum_{i=1}^{d} (u_o)_i (v_c)_i$$

$$= (u_o)_j$$

Each term is zero except when $i = j$

Source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture01-wordvecs1.pdf

$$\textcircled{2}\ \frac{\partial}{\partial v_c} \underbrace{\log \underbrace{\sum_{w=1}^{V} \exp(u_w^T v_c)}_{z = g(v_c)}}_{f}$$

$$= \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \frac{\partial}{\partial v_c} \sum_{x=1}^{V} \exp(u_x^T v_c)$$

Important to change index

$$\frac{\partial}{\partial v_c} f(\overbrace{g(v_c)}^{z}) = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial v_c}$$

Use chain rule

$$= \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \left(\sum_{x=1}^{V} \frac{\partial}{\partial v_c} \underbrace{\exp}_{f} \underbrace{(u_x^T v_c)}_{z = g(v_c)} \right)$$

Move deriv inside sum

$$\left(\sum_{x=1}^{V} \exp(u_x^T v_c) \frac{\partial}{\partial v_c} u_x^T v_c \right)$$

Chain rule

$$\left(\sum_{x=1}^{V} \exp(u_x^T v_c)\, u_x \right)$$

Source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture01-wordvecs1.pdf

$$\frac{\partial}{\partial v_c} \log\left(p(o|c)\right) = u_o - \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot \left(\sum_{x=1}^{V} \exp(u_x^T v_c)\, u_x\right)$$

$$= u_o - \sum_{x=1}^{V} \frac{\exp(u_x^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}\, u_x$$

Distribute term across sum

$$= u_o - \sum_{x=1}^{V} p(x|c)\, u_x$$

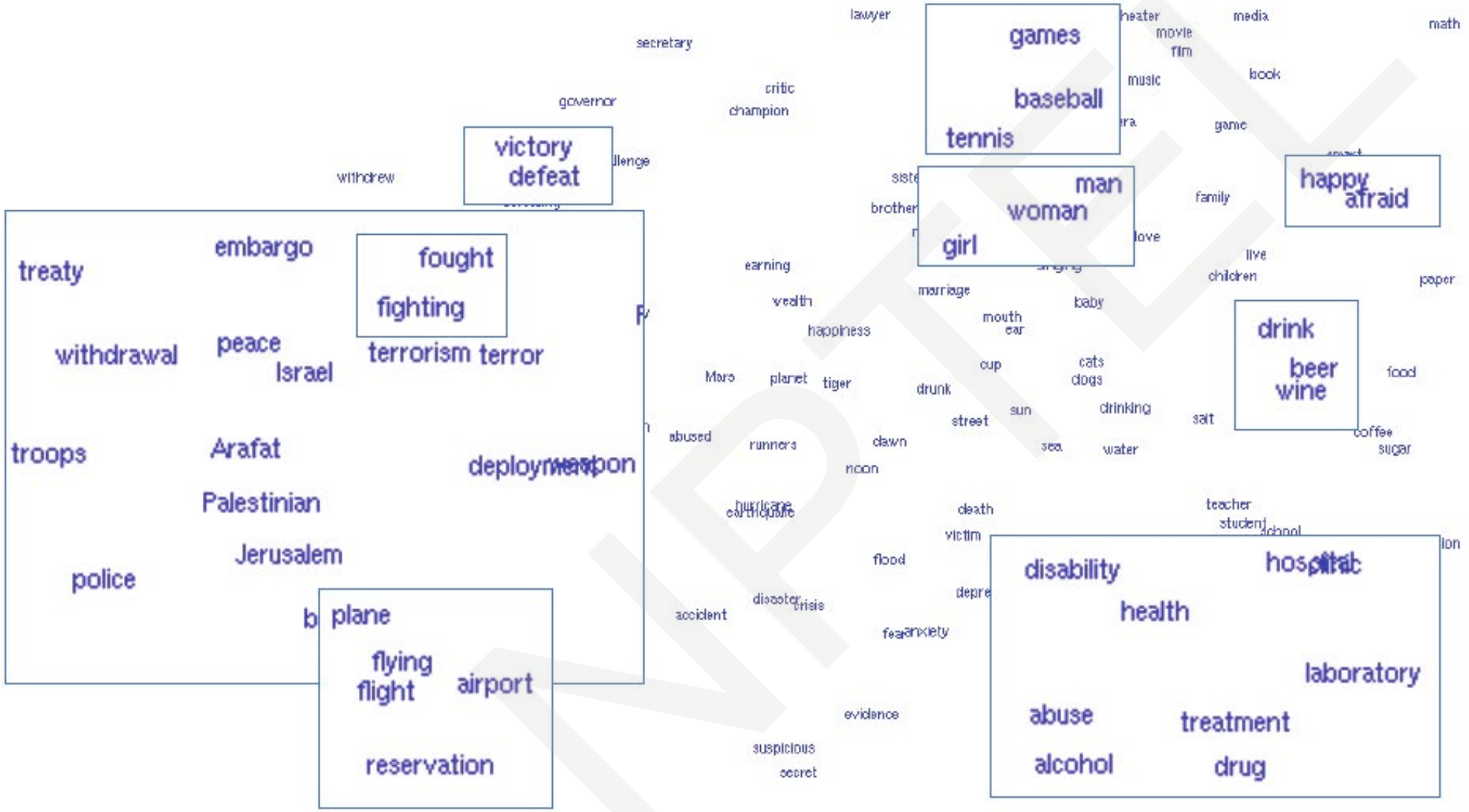
this an expectation: average over all context vectors weighted by their probability

$$= \text{observed} - \text{expected}$$

This is just the derivatives for the center vector parameters
Also need derivatives for output vector parameters
(they're similar)
Then we have derivative w.r.t. all parameters and can minimize

Source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture01-wordvecs1.pdf

Visualization



Issues with word2Vec

Computational Overhead

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

Solutions

- Negative Sampling
- Hierarchical Softmax

Both the solutions optimize a different objective function!

REFERENCES

Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. [Chapter 6]


THANK YOU

NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 13 : Learning Word Representation - II

PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- Skip-gram: Negative Sampling, Hierarchical Softmax
- Glove Word Vectors

Issues with word2Vec

Computational Overhead

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

Solutions

- Negative Sampling
- Hierarchical Softmax

Both the solutions optimize a different objective function!

Negative Sampling: Formulation

- Consider a pair (w, c) of word and context. *Did this pair come from the training data?*
- Let $P(D = 1|w, c)$ denote the probability that (w, c) came from the corpus data.
- $P(D = 0|w, c)$ will be the probability that it did not come.
- $P(D = 1|w, c)$ is denoted with the sigmoid function

$$P(D = 1|w, c, \theta) = \sigma(v_c^T v_w) = \frac{1}{1 + e^{(-v_c^T v_w)}}$$

Negative Sampling: Objective function

- maximize the probability of a word and context being in the corpus data if it indeed is, and
- maximize the probability of a word and context not being in the corpus data if it indeed is not

$$\theta = \underset{\theta}{\text{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} P(D=0|w,c,\theta)$$

$$= \underset{\theta}{\text{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} (1-P(D=1|w,c,\theta))$$

$$= \underset{\theta}{\text{argmax}} \sum_{(w,c)\in D} \log P(D=1|w,c,\theta) + \sum_{(w,c)\in \tilde{D}} \log(1-P(D=1|w,c,\theta))$$

$$= \underset{\theta}{\text{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(1-\frac{1}{1+\exp(-u_w^T v_c)})$$

$$= \underset{\theta}{\text{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(\frac{1}{1+\exp(u_w^T v_c)})$$

Negative Sampling: Objective function

Minimizing the negative log likelihood

$$J = -\sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} - \sum_{(w,c)\in \tilde{D}} \log\left(\frac{1}{1+\exp(u_w^T v_c)}\right)$$

$\tilde{D}$ is a "false" or "nagative" corpus. We can generate $\tilde{D}$ on the fly by randomly sampling from the word bank

$$-\log \sigma(u_c^T \cdot \hat{v}) - \sum_{k=1}^{K} \log \sigma(-\tilde{u}_k^T \cdot \hat{v})$$

Here $\{\tilde{u_k} | k = 1, \ldots, K\}$ are sampled from $P_n(w)$, where $P_n(w)$ is unigram model raised to the power $3/4$. Some intuition:

$$\text{is: } 0.9^{3/4} = 0.92$$

$$\text{Constitution: } 0.09^{3/4} = 0.16$$

$$\text{bombastic: } 0.01^{3/4} = 0.032$$

Skip-Gram Training data

…lemon, a [tablespoon of apricot jam, a] pinch…

c1 c2 [target] c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Skip-gram classifier – Intuition

```
... lemon,   a [tablespoon of apricot jam,       a] pinch ...
                c1          c2     w     c3       c4
```

$$\sigma(z) = \frac{1}{1+e^{-z}} \ldots \text{sigmoid function}$$

$$p(+|w,c) = 1$$

$p(+|\text{apricot,tablespoon}) = 1$

$p(+|\text{apricot,of}) = 1$

$p(+|\text{apricot,jam}) = 1$

$p(+|\text{apricot,a}) = 1$

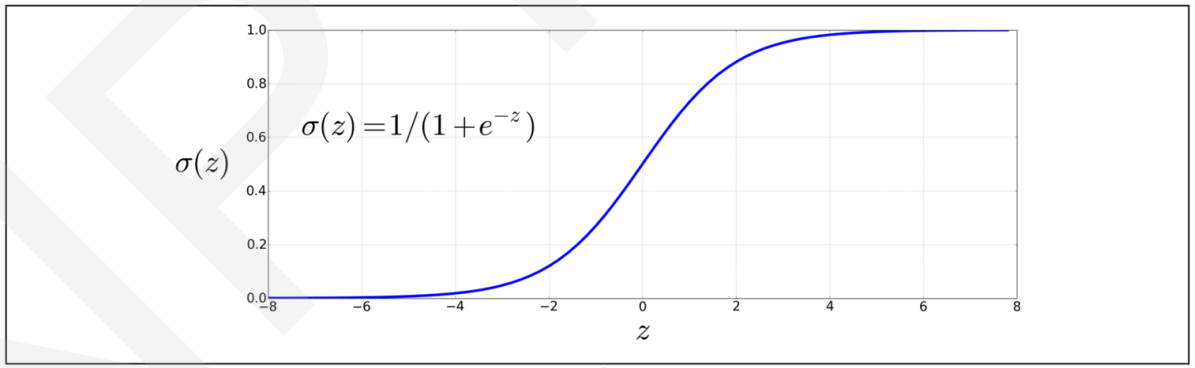


Figure 5.1 The sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range $(0,1)$. It is nearly linear around 0 but outlier values get squashed toward 0 or 1.

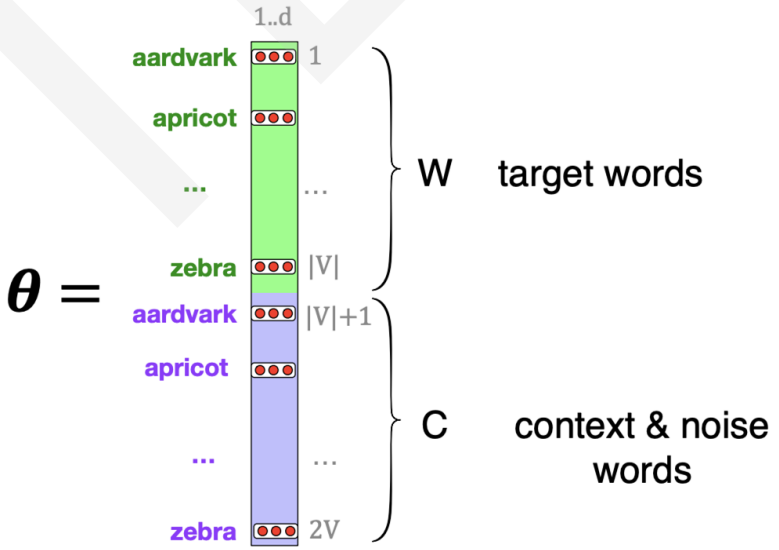
embedding similarity high ⇒ probability high too

Skip-gram model

$$\mathbb{P}(+|w, c_{1:L}) = \prod_{i=1}^{L} \mathbb{P}(+|w, c_i) = \prod_{i=1}^{L} \frac{e^{c_i \cdot w}}{1 + e^{c_i \cdot w}} \Rightarrow \log \mathbb{P}(+|w, c_{1:L}) = \sum_{i=1}^{L} \log \frac{e^{c_i \cdot w}}{1 + e^{c_i \cdot w}}$$

Given a target word w & its context window of L words c_{1:L} , assigns a probability based on how similar this context window is to the target word

The probability is based on applying the sigmoid function to the dot product of the embeddings of the target word with each context word



Skip-gram learning algorithm

Given a large corpus, **positive** and **negative examples**, & **randomly initialized embeddings**

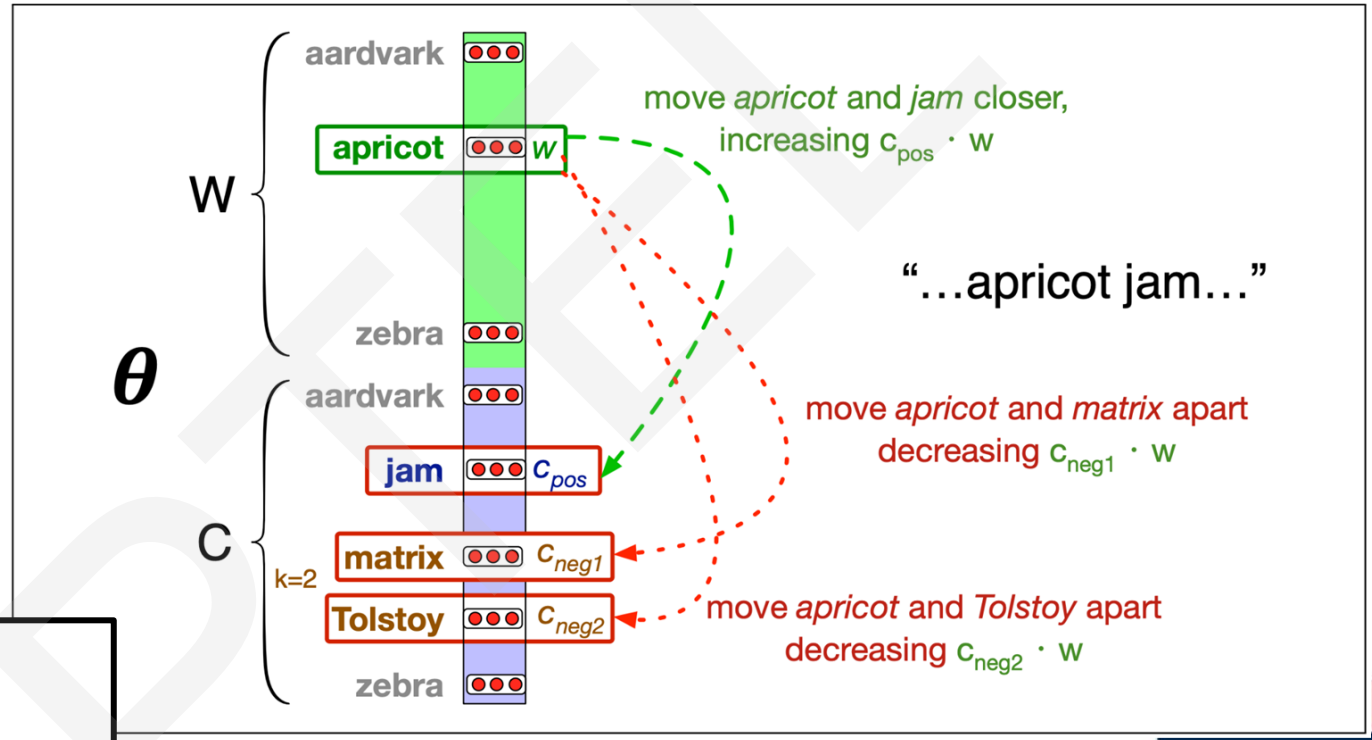
$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log\sigma(c_{pos}\cdot w) + \sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

k is a hyperparameter
$\in \{2,...,5\}$ for larger corpora

Maximize the emb. similarity of the real target-context word pairs (w,c_pos) drawn from the corpus

Maximize the emb. distance between target noncontext word pairs (w,c_neg) drawn from the negative examples

Skip-gram learning algorithm – Stochastic gradient descent



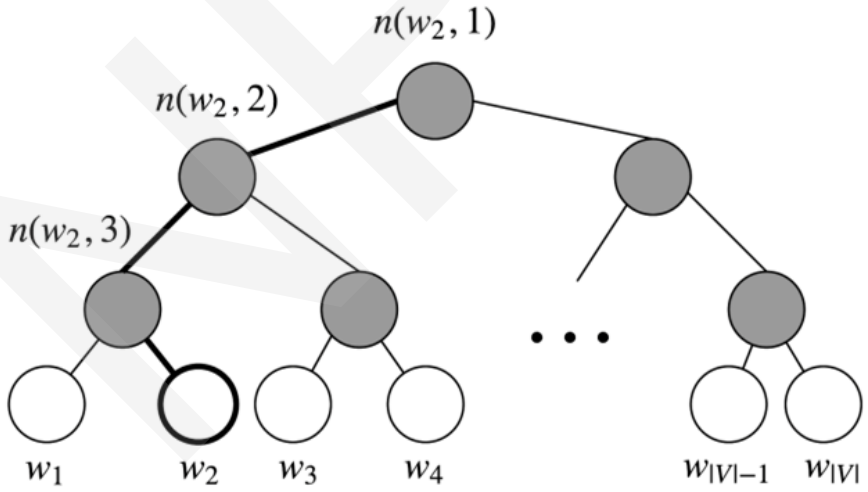
$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{w}$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(\mathbf{c}_{neg} \cdot \mathbf{w})]\mathbf{w}$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{c}_{pos} + \sum_{i=1}^{k}[\sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w})]\mathbf{c}_{neg_i}$$

Hierarchical Softmax: Formulation

- It uses a binary tree to represent all words in the vocabulary (as leaf nodes).
- There is no output representation, instead each non-leaf node is associated with a learnable vector.
- Let $L(w)$ be the number of non-leaf nodes in the path from root to the leaf w ($= 3$ for w_2)
- Let $n(w, i)$ be the i-th node on this path with vector $v_{n(w,i)}$



Hierarchical Softmax: Formulation

- For each inner node n, we arbitrarily choose one of its children and call it $ch(n)$ (e.g., always the left node)
- Probability of a word w given w_i is defined as the probability of a random walk from the root to the word's leaf.

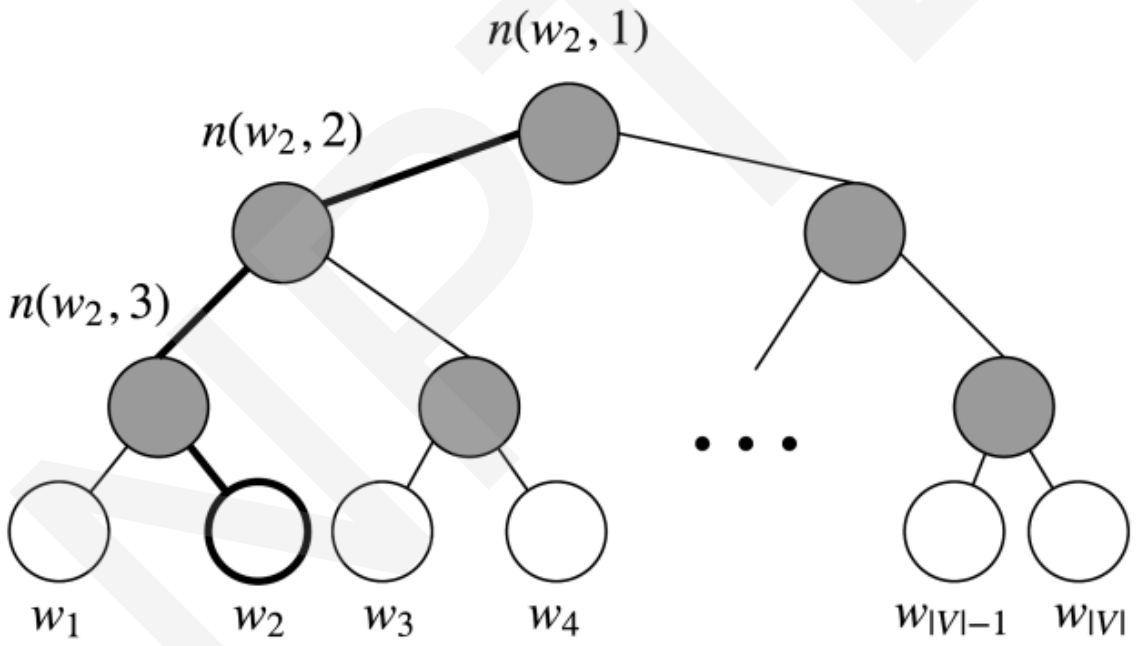
$$P(w|w_i) = \prod_{j=1}^{L(w)} \sigma([n(w,j+1) = ch(n(w,j))] \cdot v_{n(w,j)}^T v_{w_i})$$

$$[x] = \begin{cases} 1 \text{ if } x \text{ is true} \\ -1 \text{ otherwise} \end{cases}$$

$$\begin{aligned} P(w_2|w_i) &= p(n(w_2,1),\text{left}) \cdot p(n(w_2,2),\text{left}) \cdot p(n(w_2,3),\text{right}) \\ &= \sigma(v_{n(w_2,1)}^T v_{w_i}) \cdot \sigma(v_{n(w_2,2)}^T v_{w_i}) \cdot \sigma(-v_{n(w_2,3)}^T v_{w_i}) \end{aligned}$$

Hierarchical Softmax: Formulation

The objective is still to minimize $-logP(w|w_i)$. However, *instead of updating output vectors per word, we update the vectors of the nodes in the binary tree (on the path).*



Hierarchical Softmax: Try this Problem

Suppose you are using hierarchical softmax to compute the probability of the context word given the center word. Suppose, at a given time during training, the center word embeddings be [0.5 0.3 -0.4 0.2], and the path to the context word goes via three nodes starting from root (left, right, right) having embeddings as [0.3 0.1 -0.1 0.05], [-0.4 0.5 0.3 -0.6] and [0.5 0.3 -0.8 -0.9], respectively. What will be the probability of the context word given the center word?

w_i : center word w_o : context word

$$P(w_o|w_i) = \prod_{j=1}^{L(w)} \sigma\left([n(w_o, j+1) = ch(n(w_o, j))] \cdot v'^{T}_{n(w_o,j)} v_{w_i}\right)$$

x

$L(w) = 3$

$x = 1, -1, -1$ (1 for left)

$v'^{T}_{n(w_o,j)} v_{w_i}$ = [0.5 0.3 -0.4 0.2] [0.3 0.1 -0.1 0.05] = 0.23

[-0.4 0.5 0.3 -0.6] = -0.29

[0.5 0.3 -0.8 -0.9] = 0.48

$\Rightarrow \sigma(0.23) * \sigma(-(-0.29)) * \sigma(-(0.48))$

$= \sigma(0.23) * \sigma(0.29) * \sigma(-0.48)$

$= 0.557 * 0.572 * 0.382$

$= 0.1218$

Two sets of embeddings

Skip-Gram with negative sampling (SGNS) learns two sets of embeddings

Target embeddings matrix W

Context embedding matrix C

It's common to just add them together, representing word *i* as the vector $w_i + c_i$

Summary: How to learn word2vec (skip-gram) embeddings

Start with V random d-dimensional vectors as initial embeddings

Train a classifier based on embedding similarity

- Take a corpus and take pairs of words that co-occur as positive examples
- Take pairs of words that don't co-occur as negative examples
- Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
- Throw away the classifier code and keep the embeddings.

Why not capture co-occurrence directly?

- Window based co-occurrence matrix (window length 5-10)
- Symmetric (irrelevant whether left or right context)
- Example corpus (take window length = 1)
  - 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

Co-occurrence vectors

- Simple count co-occurrence vectors
  - Vectors increase in size with vocabulary
  - Very high dimensional: require a lot of storage (though sparse)
  - Subsequent classification models have sparsity issues → Models are less robust

- Low-dimensional vectors
  - Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
  - Usually 25–1000 dimensions, similar to word2vec
  - How to reduce the dimensionality?

Glove Formulation

Table 1: Co-occurrence probabilities for target words *ice* and *steam* with selected context words from a 6 billion token corpus. Only in the ratio does noise from non-discriminative words like *water* and *fashion* cancel out, so that large values (much greater than 1) correlate well with properties specific to ice, and small values (much less than 1) correlate well with properties specific of steam.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?

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

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

GloVe Formulation

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i)$$

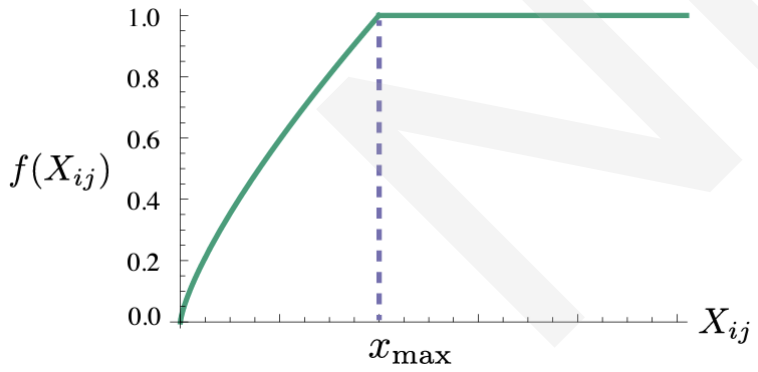
log(X_i) can be absorbed in bias

Symmetric formulation

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

but it weights all co-occurrences equally

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



f(x) should be relatively small for large values of x, so that frequent co-occurrences are not over-weighted.

Connection with Skip-gram

Let Q_{ij} be the probability that word *j* appears in the context of word *i*.

$$Q_{ij} = \frac{\exp(w_i^T \tilde{w}_j)}{\sum_{k=1}^{V} \exp(w_i^T \tilde{w}_k)}$$

The global objective function can be written as

$$J = -\sum_{\substack{i \in \text{corpus} \\ j \in \text{context}(i)}} \log Q_{ij} = -\sum_{i=1}^{V} \sum_{j=1}^{V} X_{ij} \log Q_{ij}$$

Recalling our notation for $X_i = \sum_k X_{ik}$ and $P_{ij} = X_{ij}/X_i$, we can rewrite J as,

$$J = -\sum_{i=1}^{V} X_i \sum_{j=1}^{V} P_{ij} \log Q_{ij} = \sum_{i=1}^{V} X_i H(P_i, Q_i)$$

weighted cross-entropy resembles the least square formulation

REFERENCES

Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. [Chapter 6]


THANK YOU

FREE ONLINE EDUCATION
swayam
शिक्षित भारत, उन्नत भारत

NPTEL

NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 14 : Word Vectors: Other Extensions

PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- Phrase embeddings
- Subword-based Representation: FastText
- Sentence and Document Representation: Doc2Vec
- Hierarchical, network and knowledge graph representation

GloVe vectors

Download pre-trained word vectors

The links below contain word vectors obtained from the respective corpora. If you want word vectors trained on massive web datasets, you need only download one of these text files! Pre-trained word vectors are made available under the Public Domain Dedication and License.

- Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip [mirror]
- Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip [mirror]
- Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 300d vectors, 822 MB download): glove.6B.zip [mirror]
- Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 200d vectors, 1.42 GB download): glove.twitter.27B.zip [mirror]

https://github.com/stanfordnlp/GloVe

How to Learn Phrase Embeddings?

- Some words appear frequently together, e.g., “New York Times”. It is easier if we treat these as phrases instead of individual words.
- These words will be replaced by unique tokens, while a bigram “this is” remains unchanged.

How to identify such candidate phrases?

Use unigram and bigram counts to compute a score, and choose phrases above a threshold

$$\text{score}(w_i, w_j) = \frac{\text{count}(w_i w_j) - \delta}{\text{count}(w_i) \times \text{count}(w_j)}.$$

Other kinds of static embeddings

fastText [Bojanowski et al, 2017]

- Limitation of word2vec: a distinct vector representation for each word, but we learned about subwords and their benefits
- Cannot handle out-of-vocabulary (OOV) tokens
- An extension which takes into account subword information
- https://fasttext.cc/ is also a library for learning of word embeddings and text classification created by Facebook's AI Research lab ⇒ **Word representations · fastText**

So, how does fastText work?

For a word at position *t* and a chosen context position *c*, using the binary logistic loss, we obtain the following negative log-likelihood:

$$\log\left(1+e^{-s(w_t,\, w_c)}\right)+\sum_{n\in\mathcal{N}_{t,c}}\log\left(1+e^{s(w_t,\, n)}\right)$$

Negative samples

However, this formulation ignores the internal structure of the word

fastText denotes each word as a bag of character n-grams

Image source: https://amitness.com/posts/fasttext-embeddings

So, how does fastText work?

For the word, all character n-grams between 3 to 6 characters are used

Word	Length(n)	Character n-grams
eating	3	<ea, eat, ati, tin, ing, ng>
eating	4	<eat, eati, atin, ting, ing>
eating	5	<eati, eatin, ating, ting>
eating	6	<eatin, eating, ating>

We also include the word w itself in the set of its n-grams: <eating>

Image source: https://amitness.com/posts/fasttext-embeddings

So, how does fastText work?

$$\log\left(1+e^{-s(w_t,\, w_c)}\right)+\sum_{n\in\mathcal{N}_{t,c}}\log\left(1+e^{s(w_t,\, n)}\right)$$

How to calculate s(w,c)?

Suppose that you are given a dictionary of n-grams of size G. Given a word w, let us denote by $\mathcal{G}_w \subset \{1,\ldots,G\}$ the set of n-grams appearing in w. We associate a vector representation $\mathbf{z}_g$ to each n-gram g. We represent a word by the sum of the vector representations of its n-grams. We thus obtain the scoring function:

$$s(w,c)=\sum_{g\in\mathcal{G}_w}\mathbf{z}_g^\top\mathbf{v}_c.$$

https://aclanthology.org/Q17-1010.pdf

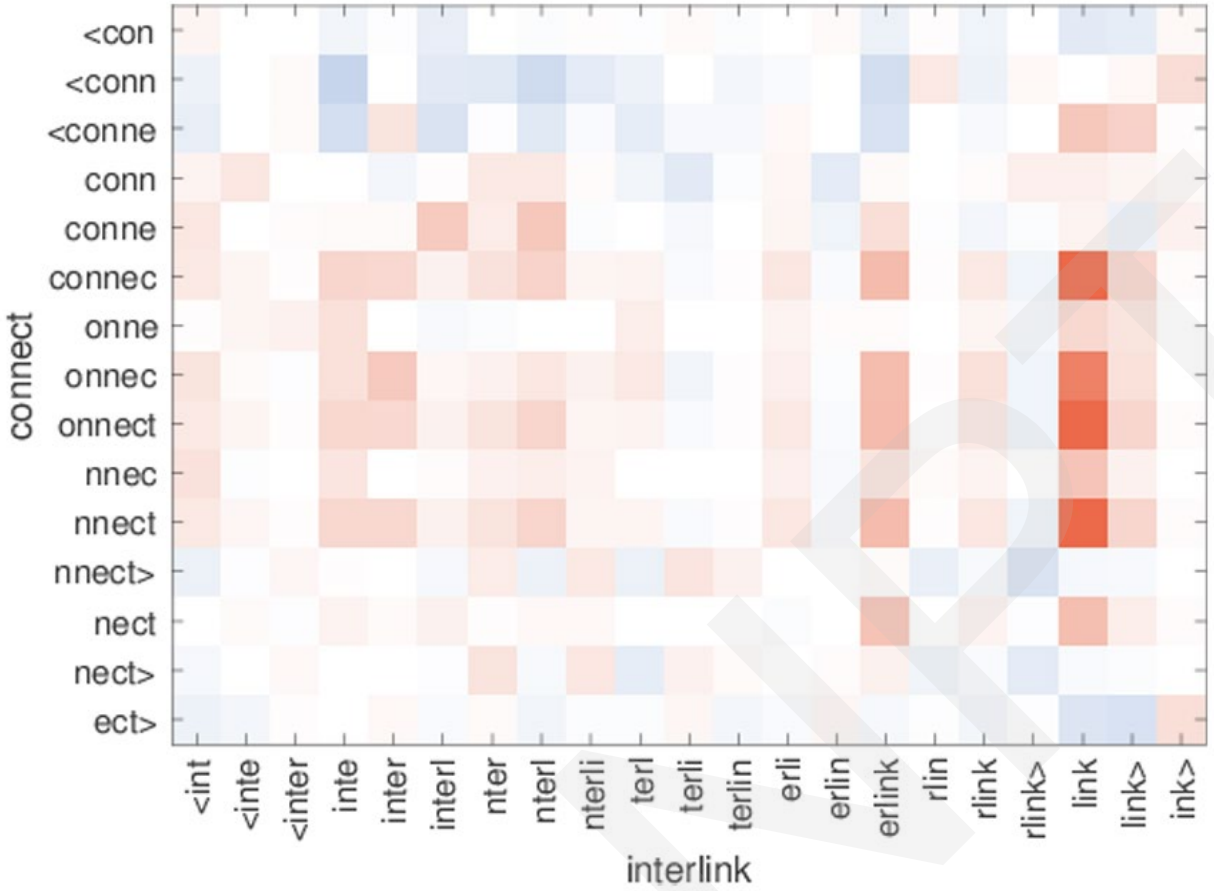
So, how does fastText work?

But the unique n-grams can be huge!

Apply hashing to bound the memory requirements.
Instead of learning an embedding for each unique n-gram, we learn total B embeddings where B denotes the bucket size. The paper used a bucket of a size of 2 million.

Image source: https://amitness.com/posts/fasttext-embeddings

How does fastText handle OOV words?



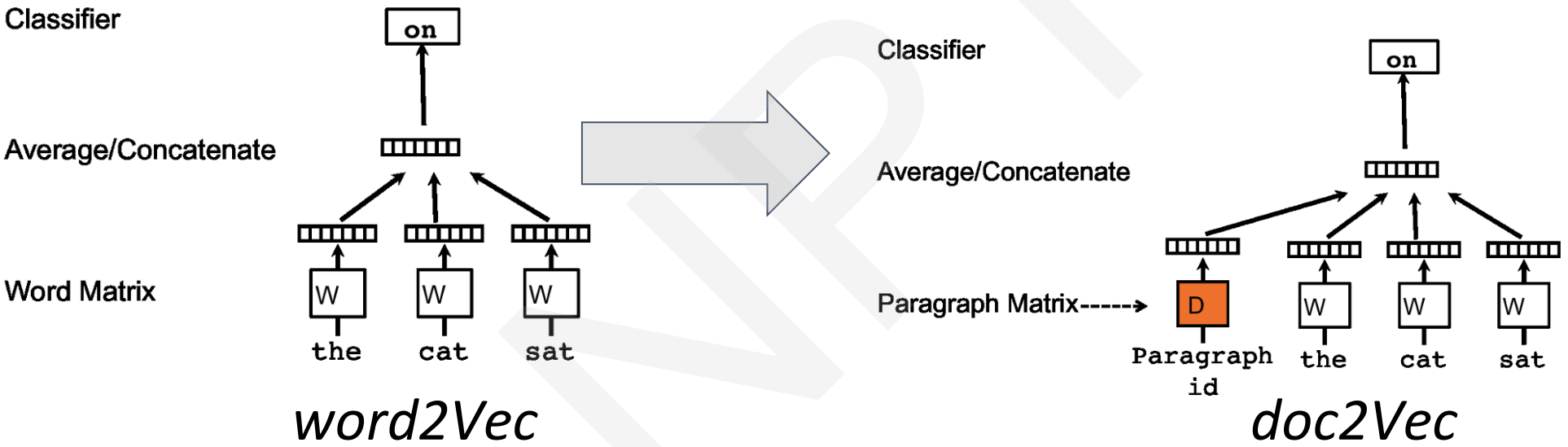
Simply average the vector representation of its n-grams

Illustration of the similarity between character n-grams in out-of-vocabulary words. The OOV word is shown on the x axis. Red indicates positive cosine, while blue negative.

Image source: https://aclanthology.org/Q17-1010.pdf

Beyond words: Sentence and documents

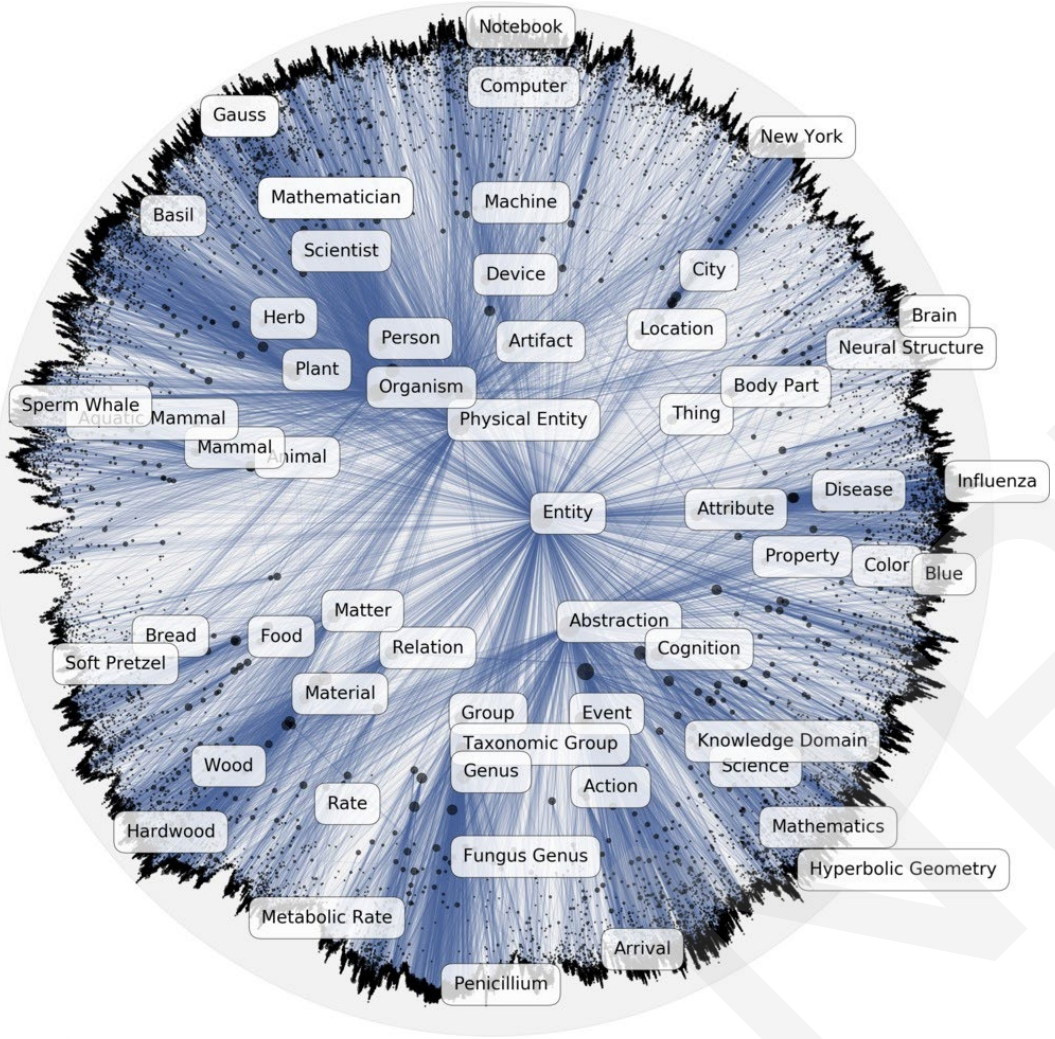
- After word2Vec, the default way for representing larger units such as sentences was to use mean embeddings
- Some works also attempted use of weighted average (tf-idf weights)
- But can we also learn the representations of larger units directly?



word2Vec *doc2Vec*

Le and Mikolov, "*Distributed Representations of Sentences and Documents*", 2014

Hierarchical Representations



Poincaré Embeddings for Learning Hierarchical Representations

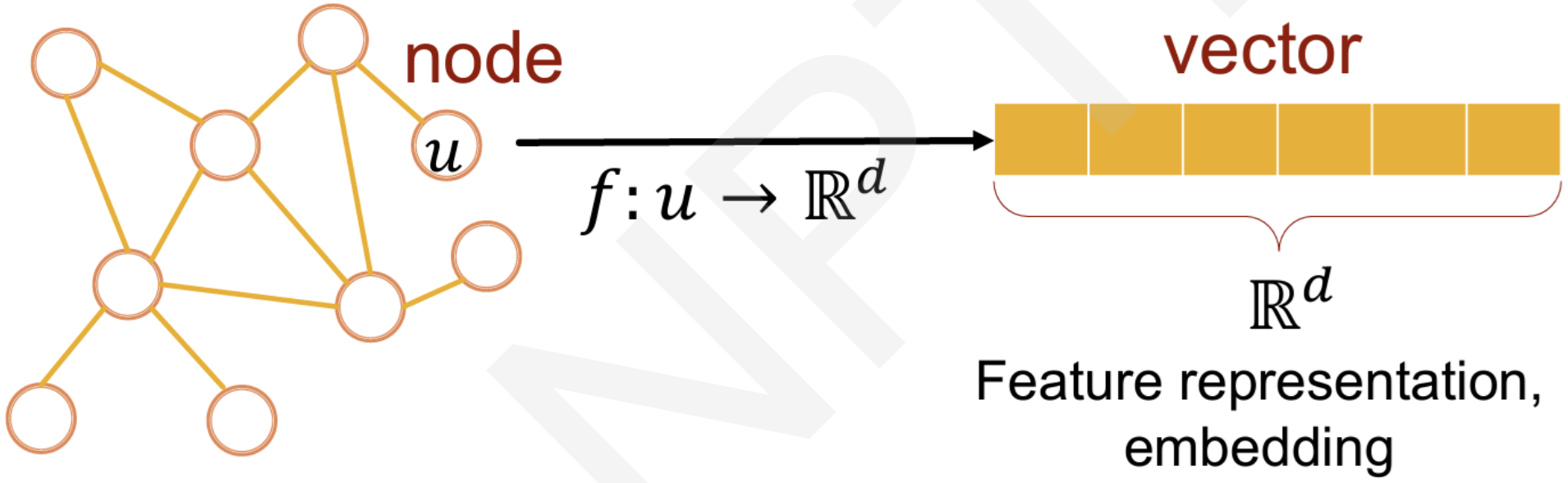
Maximilian Nickel
Facebook AI Research
maxn@fb.com

Douwe Kiela
Facebook AI Research
dkiela@fb.com

https://github.com/facebookresearch/poincare-embeddings

Node embeddings

Goal: Efficient task-independent feature learning for machine learning with graphs!



https://web.stanford.edu/class/cs224w/

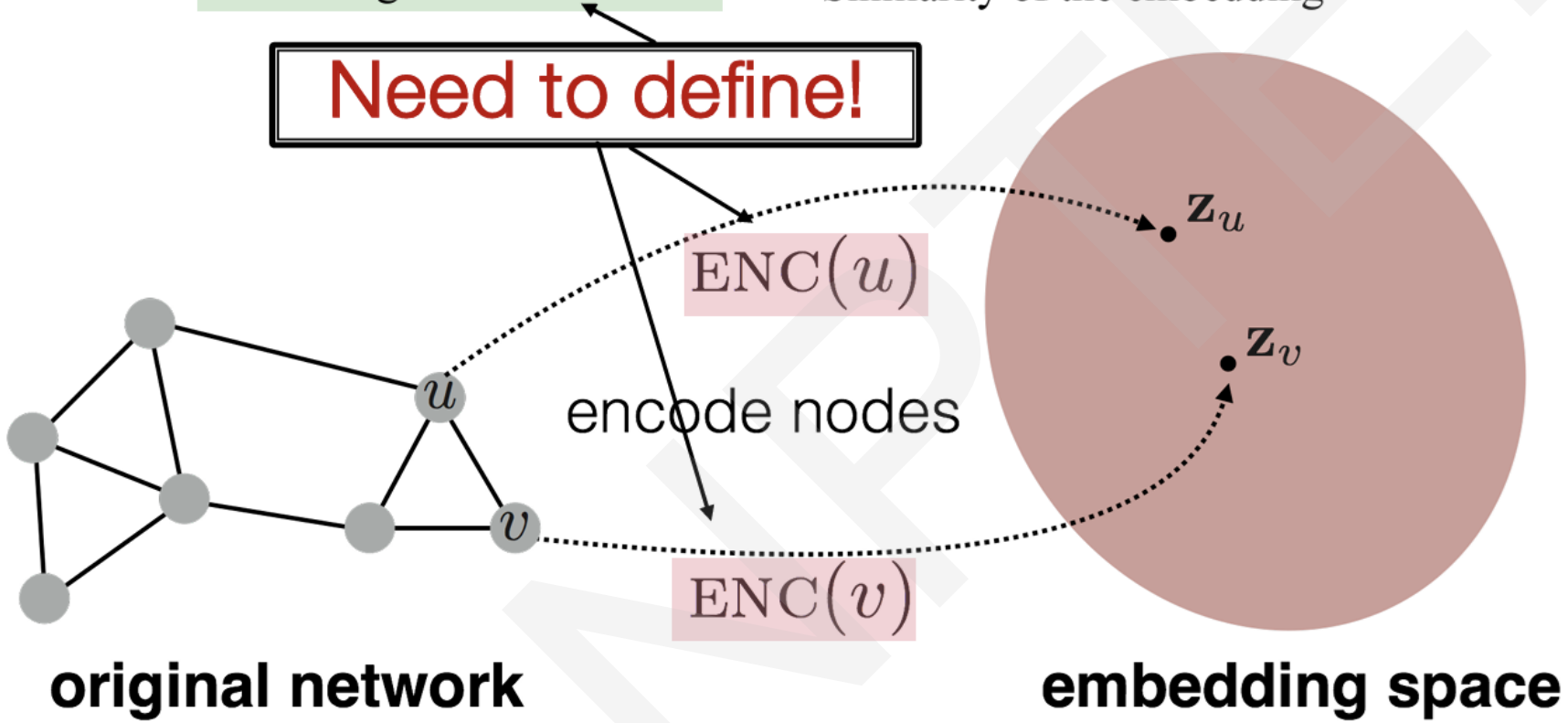
Graph Representation Learning

Goal: $\text{similarity}(u, v) \approx \mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$

in the original network

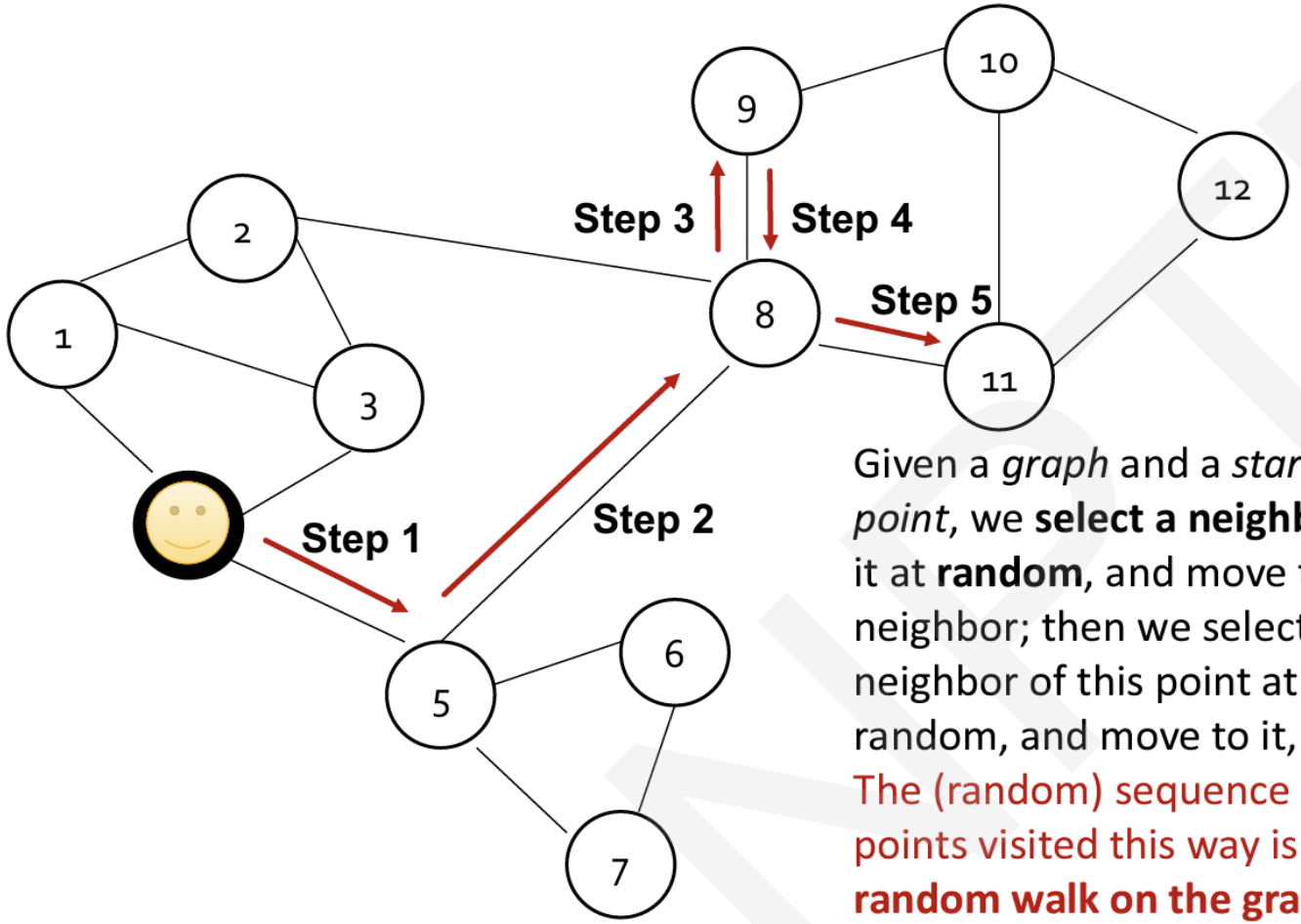
Similarity of the embedding

Need to define!



https://web.stanford.edu/class/cs224w/

Graph Representation Learning



Given a *graph* and a *starting point*, we **select a neighbor** of it at **random**, and move to this neighbor; then we select a neighbor of this point at random, and move to it, etc. The (random) sequence of points visited this way is a **random walk on the graph**.

$$\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v \approx$$

probability that u and v co-occur on a random walk over the graph

https://web.stanford.edu/class/cs224w/

Graph Representation Learning: DeepWalk

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^T \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^T \mathbf{z}_n)}\right)$$

sum over all nodes u

sum over nodes v seen on random walks starting from u

predicted probability of u and v co-occuring on random walk

Optimizing random walk embeddings =

Finding embeddings $\mathbf{z}_u$ that minimize $\mathcal{L}$

https://web.stanford.edu/class/cs224w/

Knowledge Graph Embeddings

- **Publicly available KGs:**
  - FreeBase, Wikidata, Dbpedia, YAGO, NELL, etc.
- **Common characteristics:**
  - **Massive**: Millions of nodes and edges
  - **Incomplete**: Many true edges are missing

Given a massive KG, enumerating all the possible facts is intractable! ⇨ **Can we predict plausible BUT missing links?**

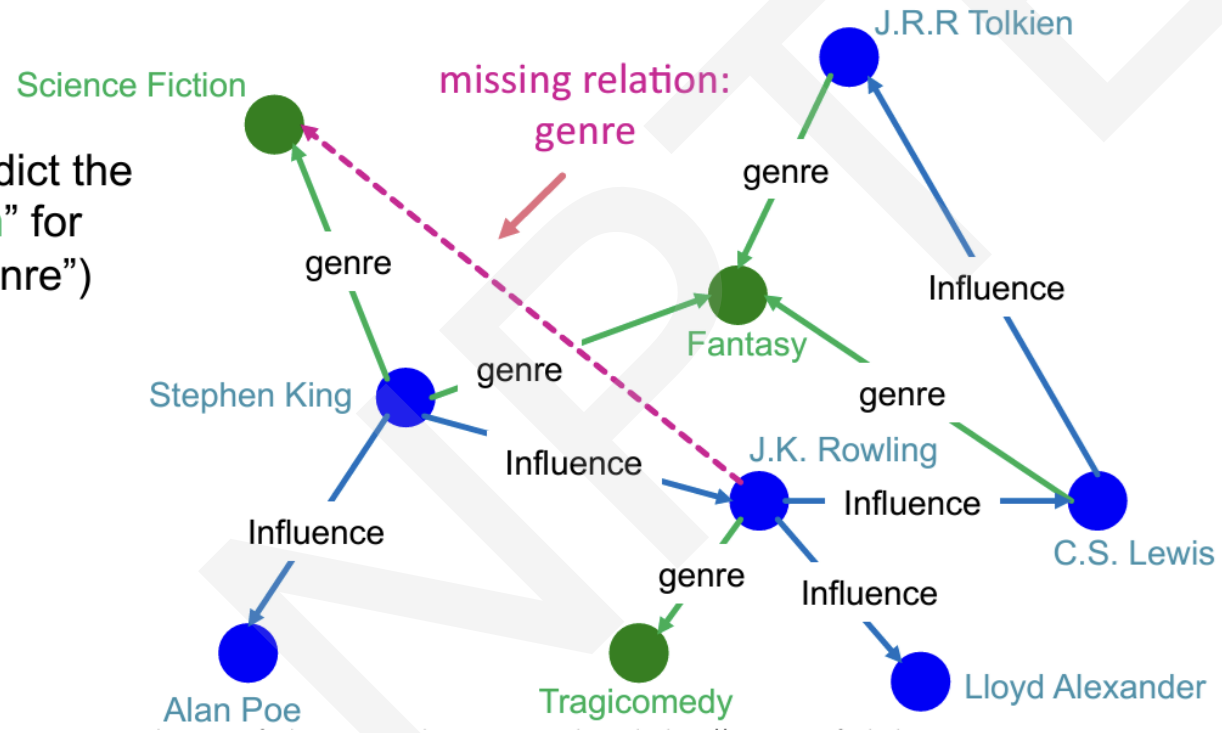
https://web.stanford.edu/class/cs224w/

Knowledge Graph Embeddings

Given an enormous KG, can we complete the KG?

- For a given (**head**, **relation**), we predict missing **tails**.
  - (Note this is slightly different from link prediction task)

Example task: predict the tail "Science Fiction" for ("J.K. Rowling", "genre")



https://web.stanford.edu/class/cs224w/

Knowledge Graph Embeddings: TransE

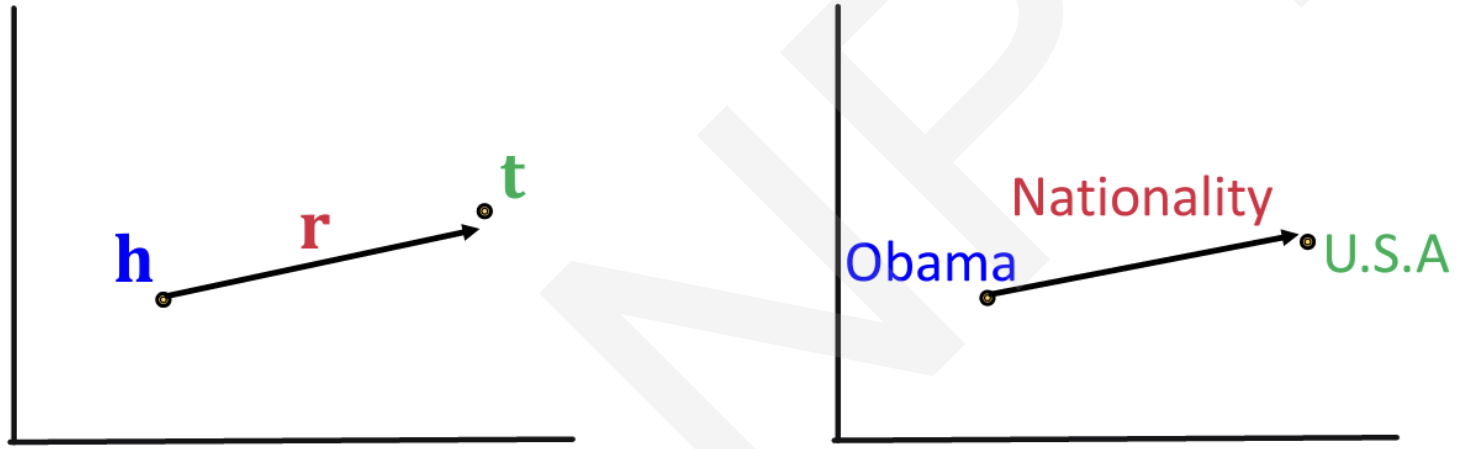
- **Intuition: Translation**

  For a triple (h, r, t), let $\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{R}^k$ be embedding vectors.

  embedding vectors will appear in boldface

- **TransE:** $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$ if the given link exists else $\mathbf{h} + \mathbf{r} \neq \mathbf{t}$

Entity scoring function: $f_r(h, t) = -||\mathbf{h} + \mathbf{r} - \mathbf{t}||$



https://web.stanford.edu/class/cs224w/

TransE: How to Learn?

Algorithm 1 Learning TransE

input Training set $S = \{(h, r, t)\}$, entities and rel. sets E and R, margin γ, embeddings dim. k.

1: **initialize** $r \leftarrow \text{uniform}(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each $r \in R$
2: $r \leftarrow r / \|r\|$ for each $r \in R$
3: $\mathbf{e} \leftarrow \text{uniform}(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each entity $e \in E$
4: **loop**
5: $\mathbf{e} \leftarrow \mathbf{e} / \|\mathbf{e}\|$ for each entity $e \in E$
6: $S_{batch} \leftarrow$ sample(S, b) // sample a minibatch of size b
7: $T_{batch} \leftarrow \emptyset$ // initialize the set of pairs of triplets
8: **for** $(h, r, t) \in S_{batch}$ **do**
9: $(h', r, t') \leftarrow$ sample$(S'_{(h,r,t)})$ // sample a corrupted triplet
10: $T_{batch} \leftarrow T_{batch} \cup \{((h, r, t), (h', r, t'))\}$
11: **end for**
12: Update embeddings w.r.t.

$$\sum_{((h,r,t),(h',r,t')) \in T_{batch}} \nabla \left[\gamma + d(\boldsymbol{h} + \boldsymbol{r}, \boldsymbol{t}) - d(\boldsymbol{h'} + \boldsymbol{r}, \boldsymbol{t'})\right]_+$$

(positive sample) (negative sample)

13: **end loop**

Initialize relations r and entities e uniformly, then normalize. γ is the margin.

Sample triplet (h', r, t) that does not appear in the KG.

d represents distance (negative of score)

Contrastive loss: Favors lower distance (or higher score) for valid triplets, high distance (or lower score) for corrupted ones

https://web.stanford.edu/class/cs224w/

REFERENCES

Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. [Chapter 6]


THANK YOU

FREE ONLINE EDUCATION
swayam
शिक्षित भारत, उन्नत भारत

NPTEL

NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 15 : Cross-Lingual Representations

PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- Cross-lingual word representations
- Various approaches: Monolingual, Cross-lingual

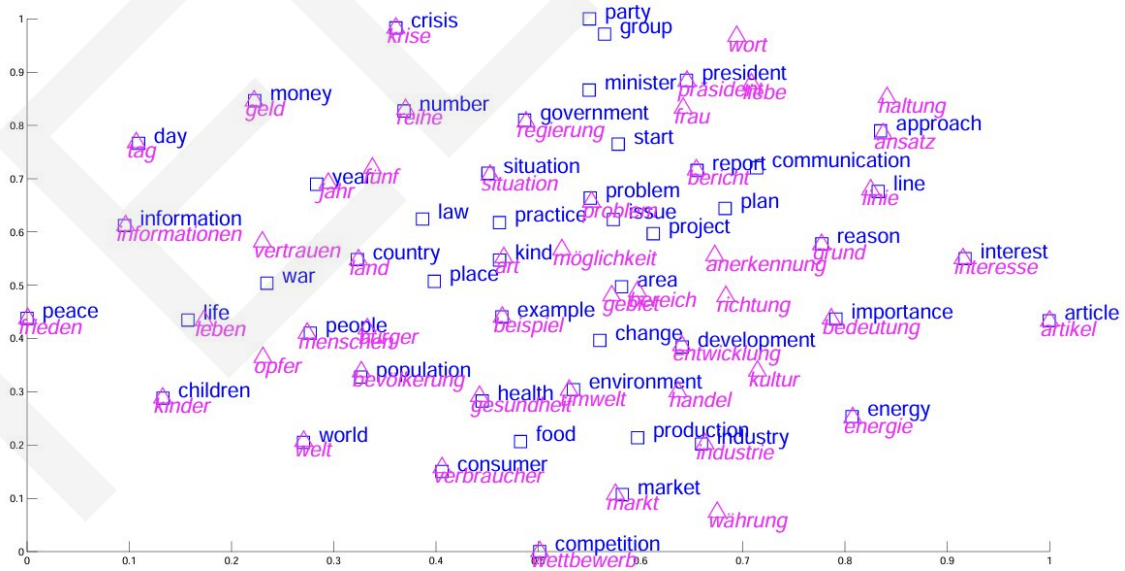
Need for cross-lingual representations

Many tasks are inherently multi-lingual / cross-lingual (e.g., translation)

Data availability across languages is very rarely the same

Can learning be transferred from High-resource to low-resource?

Can we save training efforts if we already have good representations for some languages but yet to learn for others?



https://nlp.stanford.edu/~lmthang/data/papers/naacl15_bivec.pdf

Types of cross-lingual embedding models

1. **Monolingual mapping**: Train monolingual word embeddings on large monolingual corpora. Then learn a linear mapping
2. **Pseudo-cross-lingual**: Create a pseudo-cross-lingual corpus by mixing contexts of different languages. Train an off-the-shelf word embedding model on the created corpus.
3. **Cross-lingual training**: Train their embeddings on a parallel corpus and optimize a cross-lingual constraint between embeddings of different languages
4. **Joint optimization**: Train their models on parallel (and optionally monolingual data). They jointly optimise a combination of monolingual and cross-lingual losses.

https://nlp.stanford.edu/~lmthang/data/papers/naacl15_bivec.pdf

Dataset Requirements

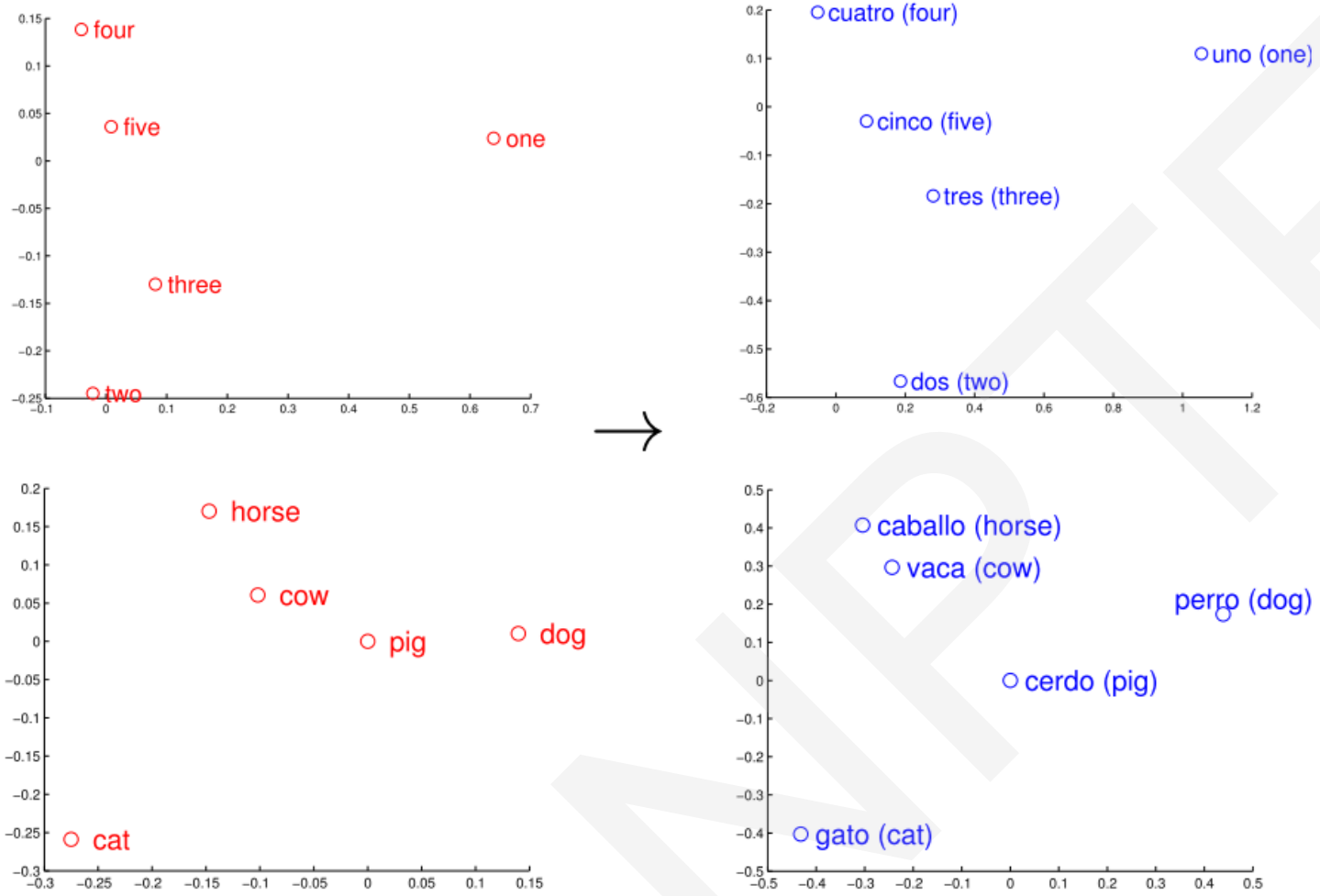
From most expensive to least expensive

1. **Word-aligned data**: A parallel corpus with word alignments
2. **Sentence-aligned data**: A parallel corpus without word alignments.
3. **Document-aligned data**: A corpus containing documents in different languages. The documents can be topic-aligned (e.g. Wikipedia) or label/class-aligned (e.g. sentiment analysis and multi-class classification datasets).
4. **Lexicon**: A bilingual or cross-lingual dictionary with pairs of translations between words in different languages.
5. **No parallel data**: No parallel data whatsoever.

https://nlp.stanford.edu/~lmthang/data/papers/naacl15_bivec.pdf

Monolingual Mapping

Word representations across languages



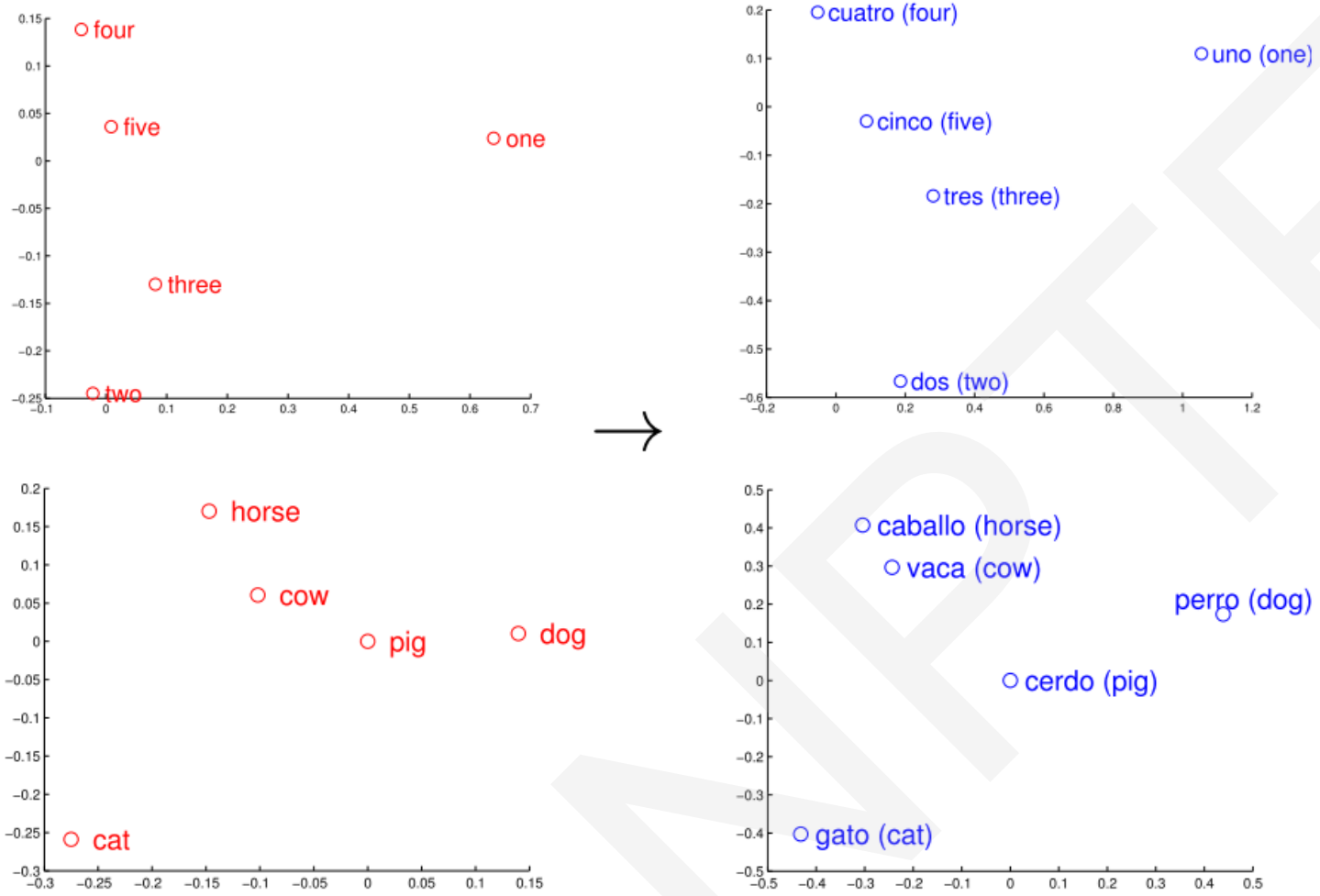
Mikolov et al. have popularised the notion that vector spaces can encode meaningful relations between words.

In addition, they notice that the geometric relations that hold between words are similar across languages

So, how can we transform one language's vector space into the space of another?

https://www.ruder.io/cross-lingual-embeddings/

Word representations across languages



So, how can we transform one language's vector space into the space of another?

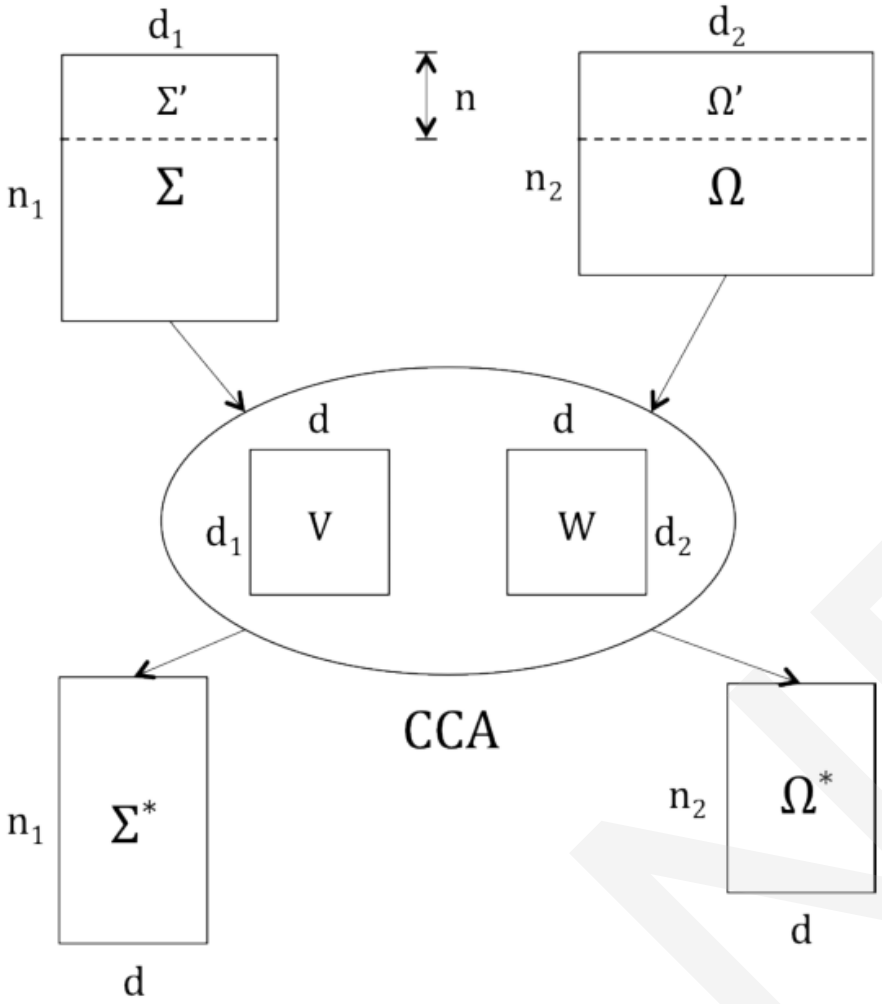
Translate the (5,000) most frequent words from the source language and use these (5,000) translations pairs as bilingual dictionary

W can be learned using stochastic gradient descent

$$\min_W \sum_{i=1}^{n} |Wx_i - z_i|^2$$

https://www.ruder.io/cross-lingual-embeddings/

Projection via CCA



Faruqui and Dyer proposed to use canonical correlation analysis (CCA) to project words from two languages into a shared embedding space.

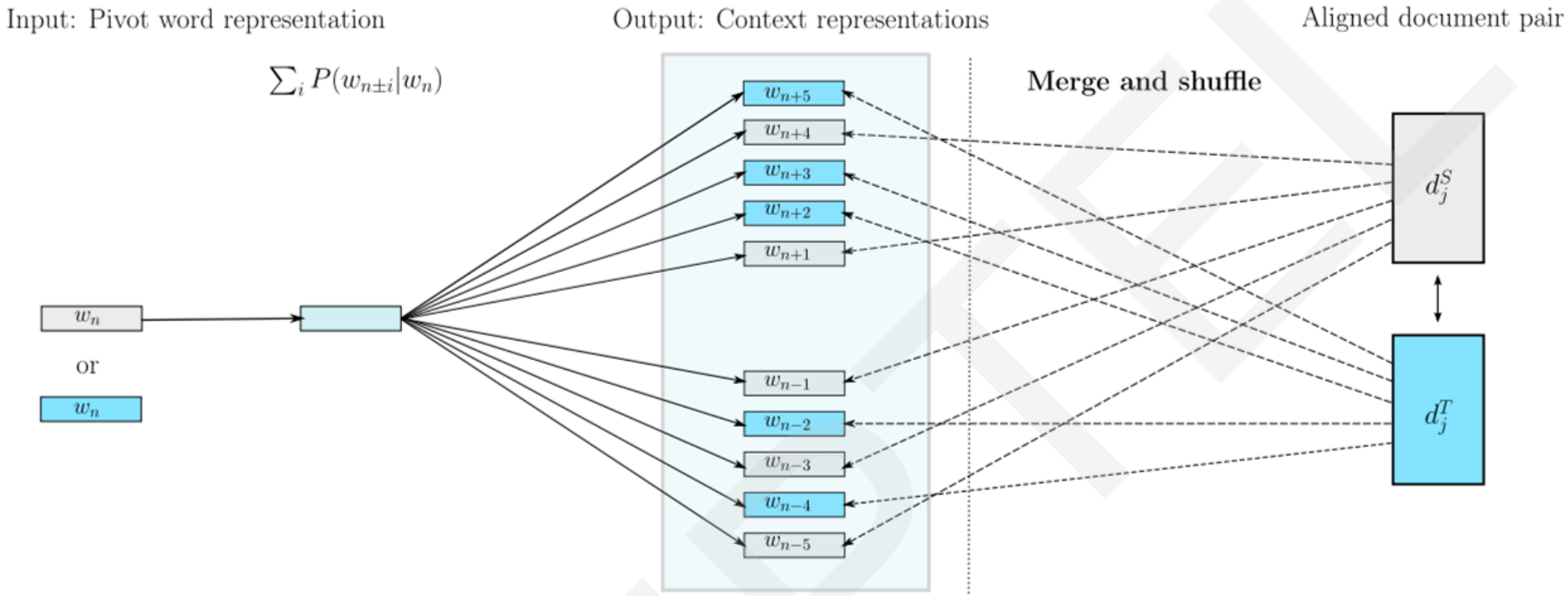
CCA learns a transformation matrix for every language

Note that Σ^* and Ω^* can be seen as the same shared embedding space

https://www.ruder.io/cross-lingual-embeddings/

Pseudo Cross-Lingual

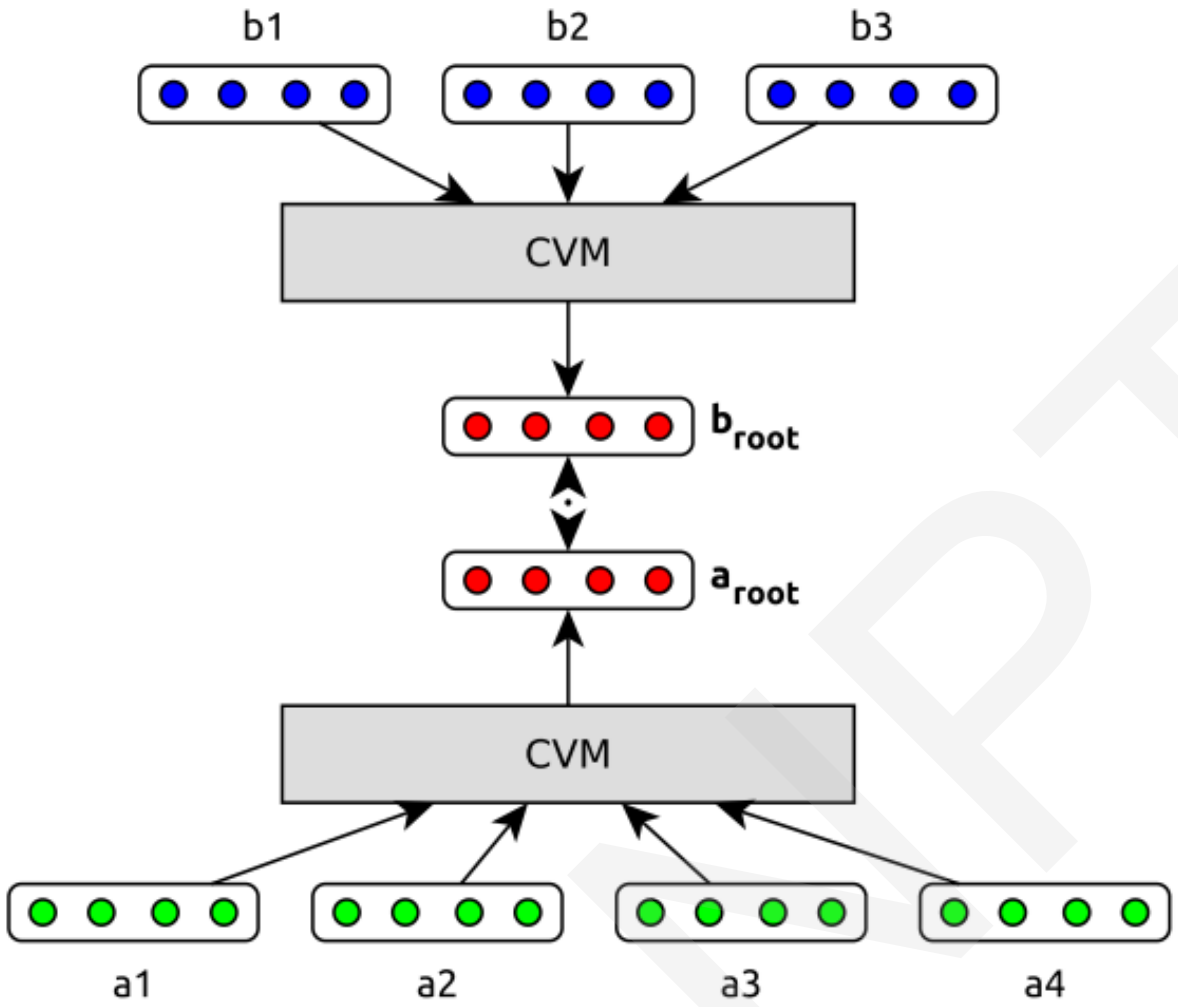
Merge and Shuffle



Vulić and Moens concatenate the (aligned) documents and then shuffle them by randomly permuting the words.
Shuffling the documents would lead to bilingual contexts for each word that will enable the creation of a robust embedding space.

Cross-Lingual Training

Bilingual compositional sentence model

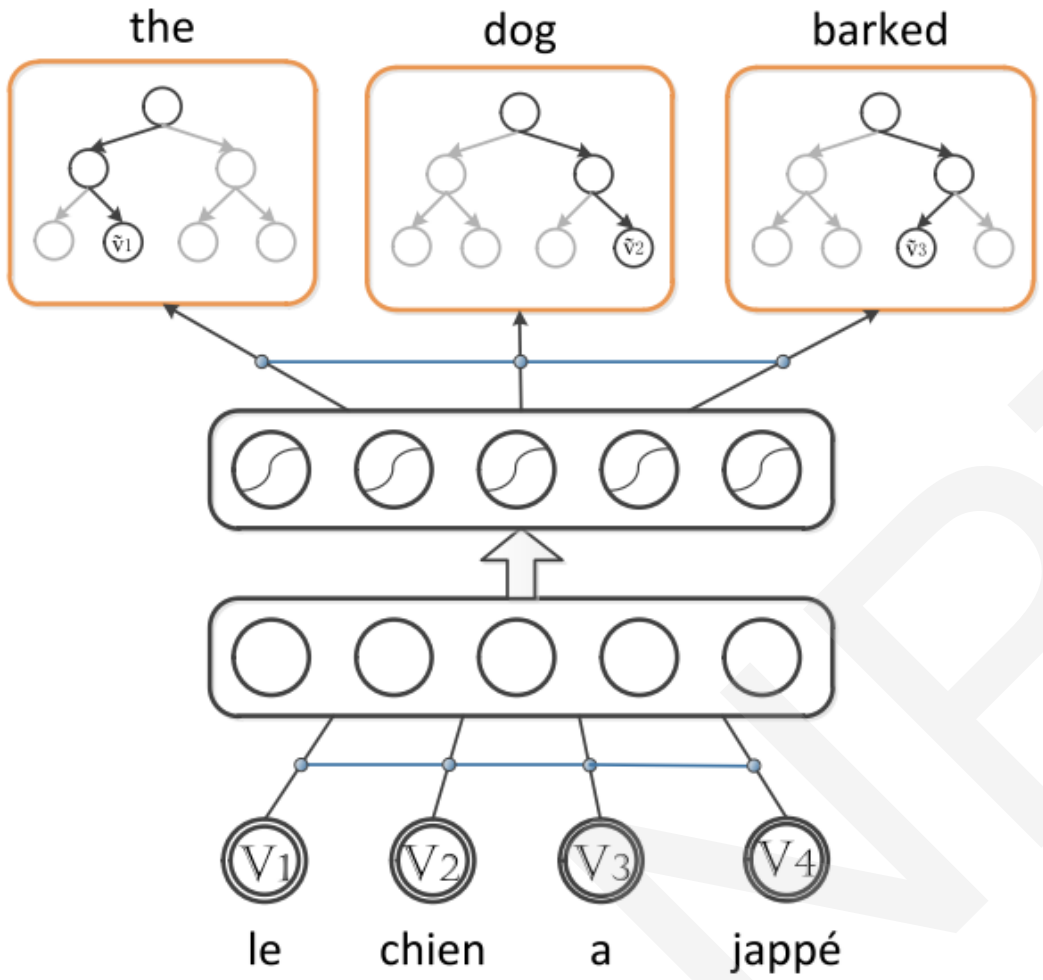


Hermann and Blunsom train two models to produce sentence representations of aligned sentences in two languages and use the distance between the two sentence representations as objective.

They compose the sentence representations simply as the sum of the embeddings of the words in the corresponding sentence.

https://www.ruder.io/cross-lingual-embeddings/

Bilingual bag-of-words autoencoder



- Lauly et al. aim to reconstruct the target sentence from the original source sentence.
- They start with a monolingual autoencoder that encodes an input sentence as a sum of its word embeddings and tries to reconstruct the original source sentence.
- For efficient reconstruction, they opt for a tree-based decoder that is similar to a hierarchical softmax.
- They then augment this autoencoder with a second decoder that reconstructs the aligned target sentence from the representation of the source sentence

For an aligned sentence pair, they then train the model with four reconstruction losses: they reconstruct from the sentence to itself and to its equivalent in the other language.

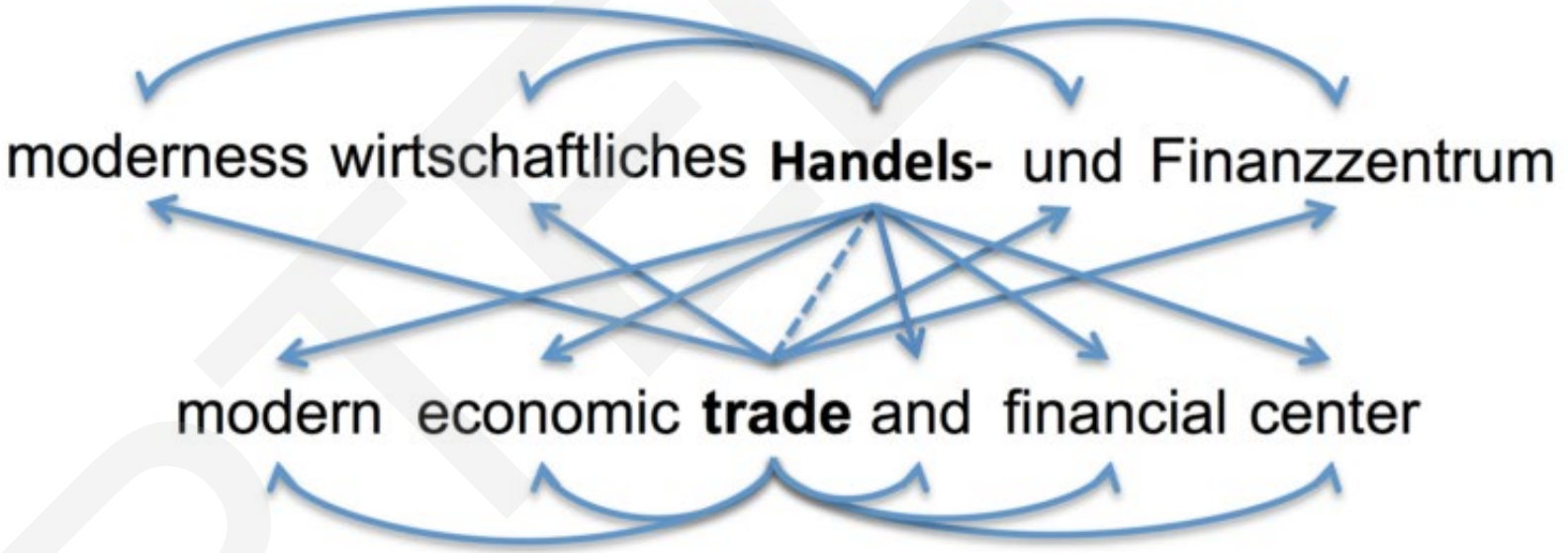
https://www.ruder.io/cross-lingual-embeddings/

Joint Optimization

Bilingual skip-gram

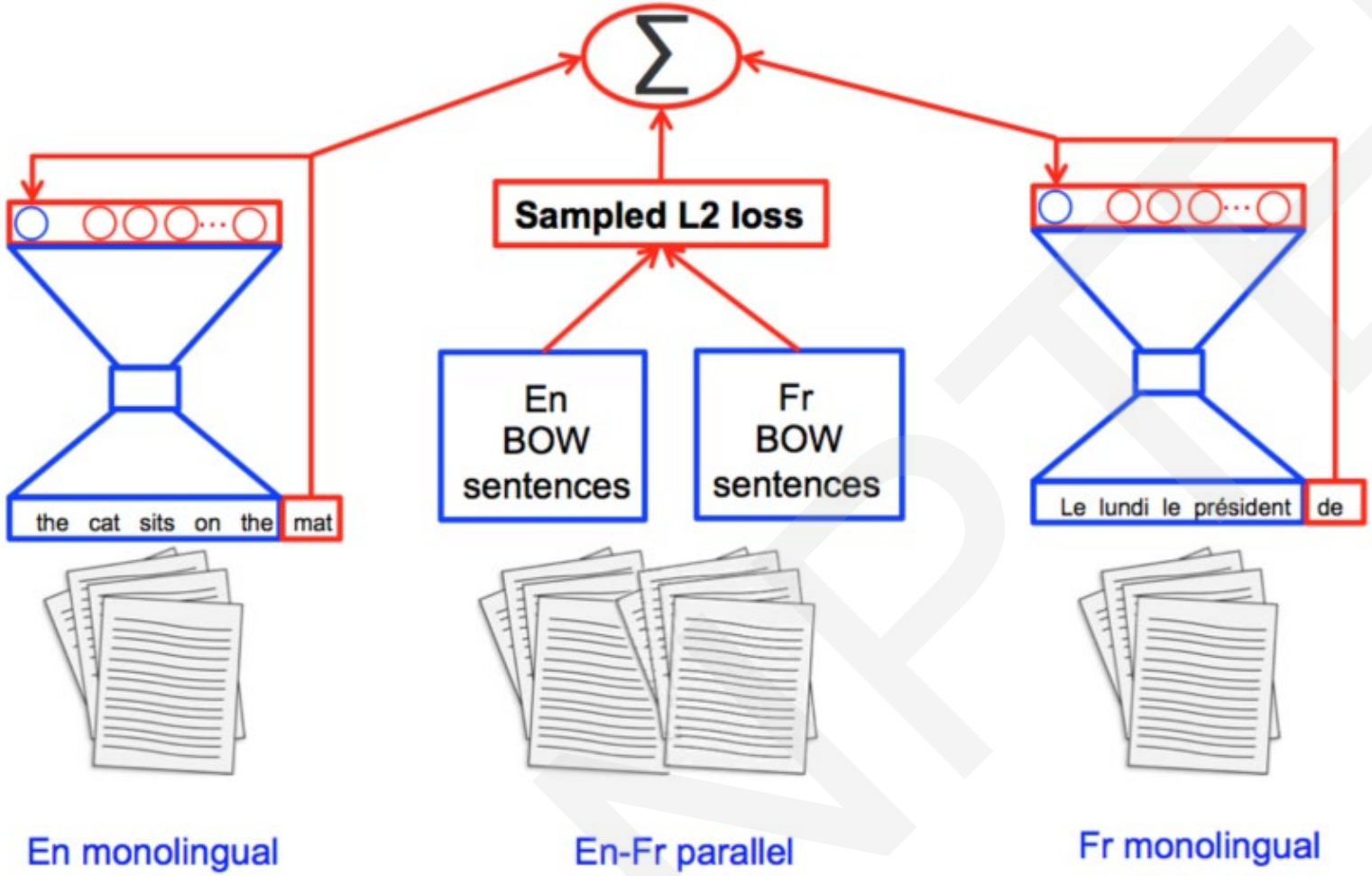
Luong et al. extend skip-gram to the cross-lingual setting and use the skip-gram objectives as monolingual and cross-lingual objectives.

Rather than just predicting the surrounding words in the source language, they use the words in the source language to additionally predict their aligned words in the target language



https://www.ruder.io/cross-lingual-embeddings/

Bilingual bag-of-words without alignment



Gouws et al. propose a Bilingual Bag-of-Words without Word Alignments (BilBOWA) that leverages additional monolingual data.

Instead of minimising the distance between words that were aligned to each other, they minimise the distance between the means of the word representations in the aligned sentences

https://www.ruder.io/cross-lingual-embeddings/

Indic FastText (IndicFT)

More

IndicFT

fastText is a subword-aware word embedding model. It is particularly well-suited for Indian languages due to their highly agglutinative morphology. We train fastText models on our IndicNLP Corpora and evaluate them on a set of tasks to measure its performance.

Our fastText models are available for 11 Indian languages: Assamese, Bengali, English, Gujarati, Hindi, Kannada, Malayalam, Marathi, Oriya, Punjabi, Tamil, Telugu.

Usage

To use our fastText models, first download them. Next, install the fastText library:

```
pip3 install fasttext
```

and then load the models like this:

```
import fasttext
model = fasttext.load_model(path_to_binary_file)
```

Evaluation

Word Similarity

Language	fastText wiki	fastText wiki+CC	Indic fastText
pa	0.467	0.384	**0.445**
hi	0.575	0.551	**0.598**
gu	0.507	0.521	**0.600**
mr	0.497	**0.544**	0.509
te	0.559	0.543	**0.578**
ta	**0.439**	0.438	0.422
Average	0.507	0.497	**0.525**

Evaluation

News Genre Classification

Language	fastText wiki	fastText wiki+CC	Indic fastText
pa	**97.12**	95.53	96.47
bn	96.57	97.57	**97.71**
or	94.80	96.20	**98.43**
gu	95.12	94.63	**99.02**
mr	96.44	97.07	**99.37**
kn	95.93	96.53	**97.43**
te	98.67	98.08	**99.17**
ml	89.02	89.18	**92.83**
ta	95.99	95.90	**97.26**
Average	95.52	95.63	**97.52**

REFERENCES

https://www.ruder.io/cross -lingual-embeddings/


THANK YOU