word representations umumnya vector

(Distributed Word Representations) Word Embeddings

Alfan F. Wicaksono
Information Retrieval Lab.
Faculty of Computer Science
Universitas Indonesia

How to more robustly match a user's intent?

Query: "**motel** di depok"

Sistem diharapkan tidak hanya mengembalikan dokumen tentang "**motel** di depok"; tetapi juga "**hotel** di depok", "**penginapan** di depok", dsb.

Expansion Using Query Logs

Context-Free Query Expansion

- Misal, dari analisis query logs didapatkan bahwa **“wet ground”** = **“wet earth”**.
- Oleh karena itu, jika kedepannya ada query term **“ground”**, perlu di-expand dengan term **“earth”**.
- Problem: **“ground coffee”** -> **“earth coffee”** ??

Thesaurus-based Approach

setiap kata ada hierarki

Contoh: Path-based similarity

Two terms are similar if they are near each other in the thesaurus hierarchy.

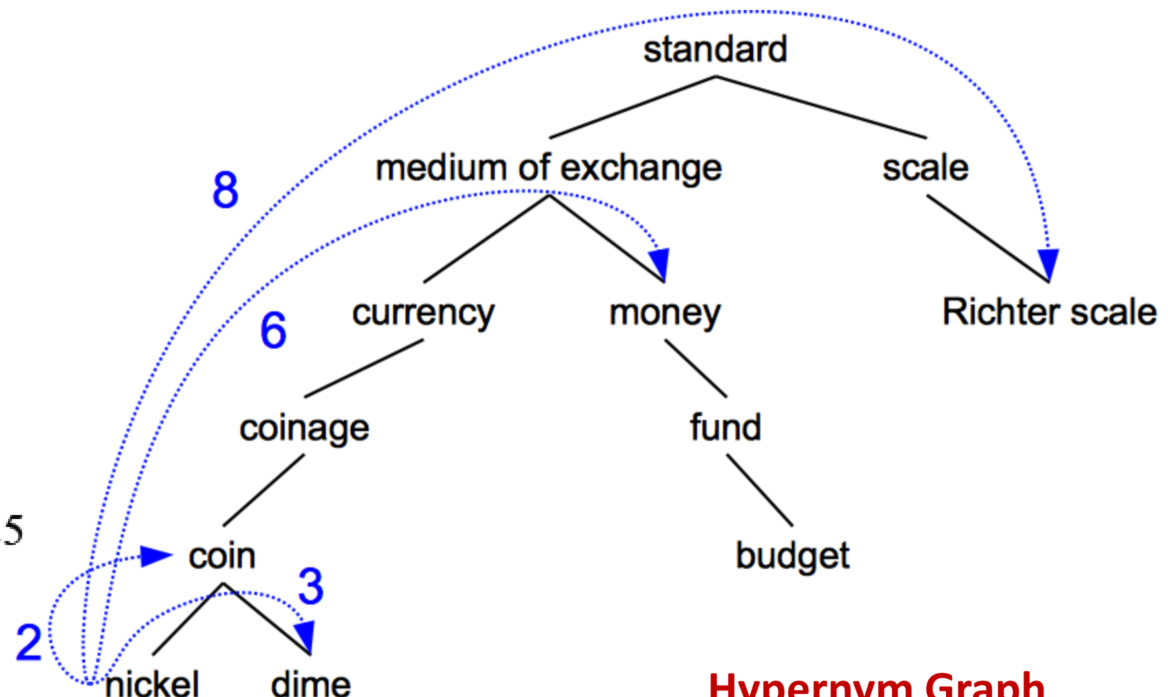
$$\text{simpath}(c_1,c_2) = \frac{1}{\text{pathlen}(c_1,c_2)}$$

$$\text{wordsim}(w_1,w_2) = \max_{c_1 \in \text{senses}(w_1), c_2 \in \text{senses}(w_2)} \text{sim}(c_1,c_2)$$

simpath(*nickel,coin*) = $1/2 = .5$

simpath(*fund,budget*) = $1/2 = .5$

simpath(*nickel,currency*) = $1/4 = .25$



Hypernym Graph

Dan Jurafsky

Thesaurus-based Approach

Problems...

- We don’t have a thesaurus for every language
- For Indonesian, our WordNet is not complete
  - Many words are missing
  - Connections between senses are missing
  - ...

Word -> Vector; Term Relations?

Pada konsep VSM standar, setiap term menjadi **basis/axis** di vector space.

Query: "hotel"

kalau pakai tf idf aja gak mungkin cocok

Document: "motel motel motel"

$$q = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, \dots, 0]$$

$$d = [0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, \dots, 0]$$

$$sim(q, d) = q.d^T = 0$$

Orthogonal!

Word -> Vector; Term Relations?

Pada konsep VSM standar, setiap term menjadi **basis/axis** di vector space.

$$sim(q,d) = q.d^T$$

q = “hotel di depok”

$$[\underset{\text{hotel}}{1}, \underset{\text{motel}}{0}, \underset{\text{di}}{1}, \underset{\text{depok}}{1}] \times \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} = 2.0$$

Make Sense?

d = “motel di depok”

Orthogonal!

Word -> Vector; Term Relations?

Query Translation Model (Berger & Lafferty, 99)

Tambahkan **trainable parameter W** untuk menangkap **similarity** atau **translasi** antar kata.

W = V * V

ini adalah W (trainable parameter)

$$
\begin{array}{c|cccc}
 & \textbf{hotel} & \textbf{motel} & \textbf{di} & \textbf{depok} \\
\textbf{hotel} \\
\textbf{motel} \\
\textbf{di} \\
\textbf{depok}
\end{array}
\begin{bmatrix}
1.0 & 0.8 & 0.0 & 0.0 \\
0.8 & 1.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 1.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 1.0
\end{bmatrix}
$$

Word -> Vector; Term Relations?

Query Translation Model (Berger & Lafferty, 99)

Tambahkan parameter **W** untuk menangkap similarity antar kata.

$$sim(q, d) = q.W.d^T$$

q = "hotel di depok"

Better!

$$[\underset{\text{hotel}}{1}, \underset{\text{motel}}{0}, \underset{\text{di}}{1}, \underset{\text{depok}}{1}] \times \begin{bmatrix} 1.0 & 0.8 & 0.0 & 0.0 \\ 0.8 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} = 2.8$$

d = "motel di depok"

Tetap saja isu Sparsity muncul. Ukuran W besar!

Distributional-based Approach

- Can we learn a **dense low-dimensional** representation of a word such that dot products $u.v^T$ express word similarity?

- Masih mungkin juga menyisipkan "translation" matrix antar vocab (misal, cross-language): $u.W.v^T$

Distributional-based Approach

- Based on the idea that contextual information alone constitutes a viable representation of linguistic items.
- As opposed to formal lingustics and the Chomsky tradition.

lihat context based on surrounding terms

- Zellig Haris (1954): "…**if A and B have almost identical environments, we say that they are synonyms**…"

- J. R. Firth (1957): "**You shall know a word by the company it keeps**"

Distributional-based Approach

gogos

Ada yang tahu apa itu **gogos**?

Distributional-based A

Makan **gogos** dengan sambal sungguh nikmat.
Beras ketan diperlukan untuk membuat **gogos**.
Teman-teman menikmati **gogos** hangat di kantin.
Menikmati makanan **gogos**, lemper dari makasar.

- From context words humans can guess ***gogos*** means
  - A traditional food
- Intuition for algorithm:
  - **Two words are similar if they have similar word contexts**

Latent Semantic Analysis

Alin Revisited: Rank

Let C be an M × N matrix. The **rank** of a matrix is the number of **linearly independent rows** (or columns) in it; thus, **rank(C) ≤ min{M, N}**.

$$C = \begin{bmatrix} 2 & 4 & 8 \\ 1 & 2 & 4 \end{bmatrix}, \qquad rank(C) = 1$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad rank(C) = 3$$

full rank

Alin Revisited: Rank

Let C be an M × N matrix. The **rank** of a matrix is the number of **linearly independent rows** (or columns) in it; thus, **rank(C) ≤ min{M, N}**.

Secara umum, ubah dahulu ke bentuk **Row-Echelon Form**; lalu hitung ada **berapa baris yang non-zero**.

$$C = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 4 \\ 3 & 0 & 5 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -2 \\ 0 & -6 & -4 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -2 \\ 0 & 0 & 0 \end{bmatrix}$$

$R_2 \rightarrow R_2 - 2R_1$
$R_3 \rightarrow R_3 - 3R_1$

$R_3 \rightarrow R_3 - 2R_2$

Rank(C) = 2

Alin Revisited: Eigenvector

For a square M × M matrix C and a non-zero vector **x**, the values of λ satisfying

eigen value

$$Cx = \lambda x$$

eigen vector

are called the **eigenvalues** of C.

x disebut **eigenvetor**.

Banyaknya non-zero eigenvalues dari C adalah paling banyak Rank(C).

Contoh:

$$C = \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix}$$

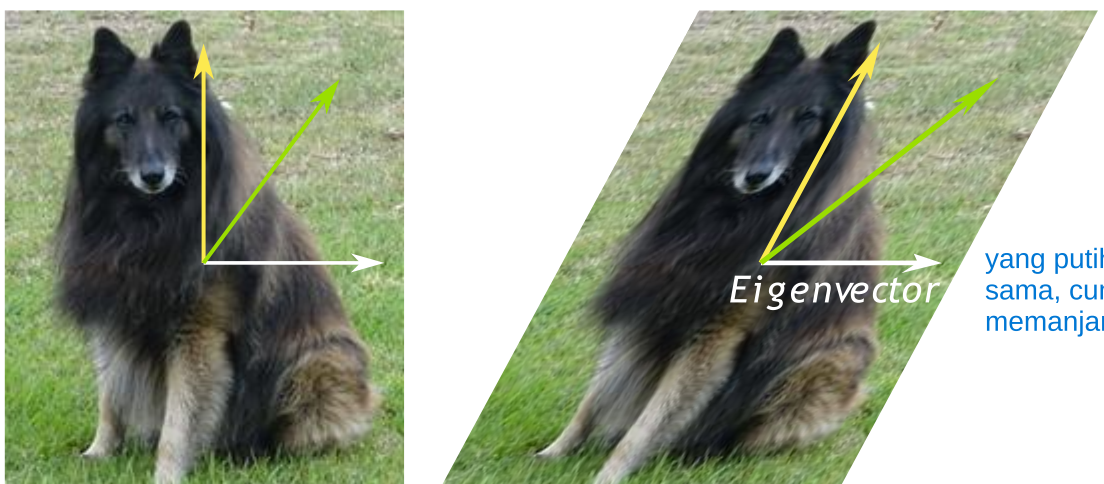
$$x_1 = \begin{bmatrix} 1 \\ 4 \end{bmatrix} \quad \lambda_1 = 6$$

$$x_2 = \begin{bmatrix} -3 \\ 1 \end{bmatrix} \quad \lambda_2 = -7$$

Alin Revisited: Eigenvector

Singkat cerita, eigenvector tidak akan berubah arah ketika ditransformasi (hanya memanjang atau memendek).

“ditransformasi” = “dikali matriks”



yang putih eigen vector arahnya sama, cuman bisa memanjang/memendek.

https://www.mathsisfun.com/algebra/eigenvalue.html

Alin Revisited: Eigenvector

Sekedar istilah …

Right-eigenvector of C $Cx = \lambda x$

$$C = \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix} \quad x = \begin{bmatrix} 1 \\ 4 \end{bmatrix} \quad \lambda = 6$$

Left-eigenvector of C $y^T C = \lambda y^T$

$$C = \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix} \quad y^T = [? \quad ?] \quad \lambda =?$$

Latihan: cari sebuah left-eigenvector dari C!

Vector Space Model

Term-Document Matrix

Each cell is the count of word *t* in document *d* (bisa juga TF-IDF)

	D1	D2	D3	D4	D5
ekonomi	0	1	40	38	1
pusing	4	5	1	3	30
keuangan	1	2	30	25	2
sakit	4	6	0	4	25
inflasi	8	1	15	14	1

document embeeding

Vector of D2 = [1, 5, 2, 6, 1]

Vector Space Model

Term-Document Matrix

Each cell is the count of word *t* in document *d* (bisa juga TF-IDF)

	D1	**D2**	**D3**	**D4**	**D5**
ekonomi	0	1	40	38	1
pusing	4	5	1	3	30
keuangan	1	2	30	25	2
sakit	4	6	0	4	25
inflasi	8	1	15	14	1

Two documents are similar if they have similar vector!

D3 = [40, 1, 30, 0, 15]

D4 = [38, 3, 25, 4, 14]

Vector Space Model

Term-Document Matrix

Each cell is the count of word *t* in document *d* (bisa juga TF-IDF)

	D1	**D2**	**D3**	**D4**	**D5**
ekonomi	0	1	40	38	1
pusing	4	5	1	3	30
keuangan	1	2	30	25	2
sakit	4	6	0	4	25
inflasi	8	1	15	14	1

word
embeedin
g

Vector of word "sakit" = [4, 6, 0, 4, 25]

word embeeding: document yang jadi context
However, kalau document embeeding: word-wordnya yang jadi context

Vector Space Model

Term-Document Matrix

Each cell is the count of word *t* in document *d* (bisa juga TF-IDF)

	D1	D2	D3	D4	D5
ekonomi	0	1	40	38	1
pusing	4	5	1	3	30
keuangan	1	2	30	25	2
sakit	4	6	0	4	25
inflasi	8	1	15	14	1

Two words are similar if they have similar vector!

pusing = [4, 5, 1, 3, 30]

sakit = [4, 6, 0, 4, 25]

Vector Space Model

Jika **C** adalah term-document matrix,

contoh C = term ada 3, document ada 4 jadi 3x4. Nah hasilnya bakal 3x3 dimana setiap elemen itu menandakan kemiripan antar term (term_1 dengan term_2)

$C.C^T$ mengandung dot products (similarity) dari semua pasangan **term vectors**; dan

1. matrixsnya persegi
2. matriksnya simetrik

kebalikannya, ini menandakan kemiripan antar document

$C^T.C$ mengandung dot products (similarity) dari semua pasangan **document vectors**.

Dekomposisi Term-Document Matrix!

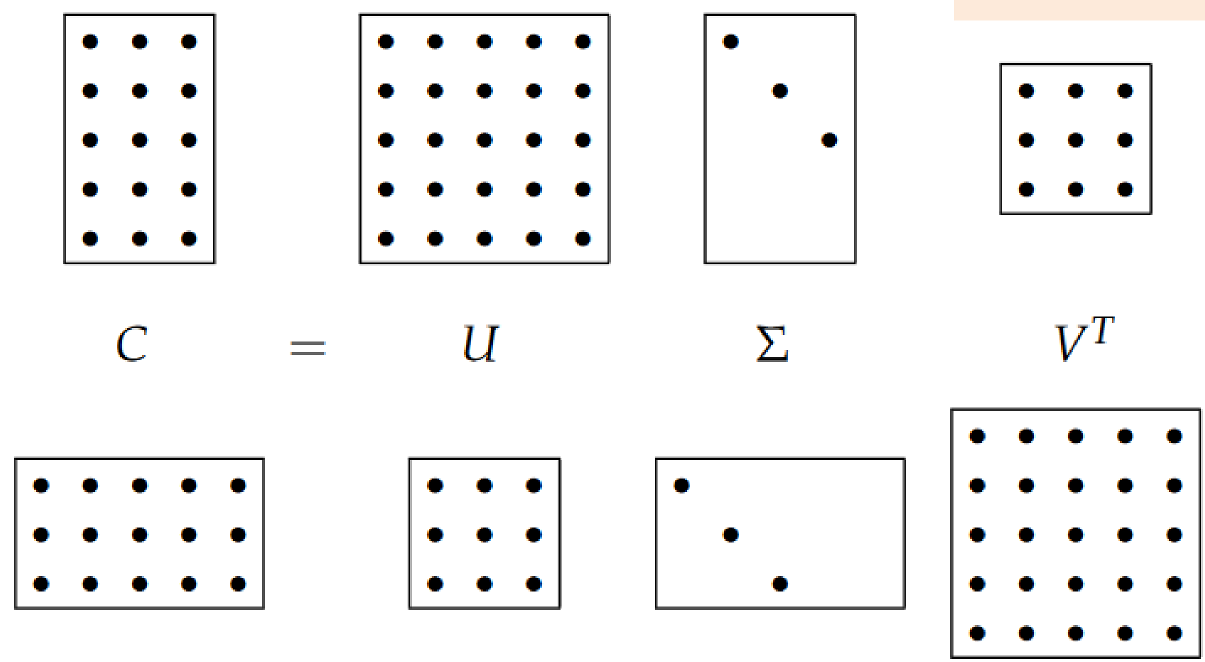
Misal, C adalah term-document matrix. C didekomposisi menjadi perkalian 3 matrix (**Singular Value Decomposition**).

$$C = U \times \Sigma \times V^T \qquad C \in R^{M \times N}$$

Cell selain titik = 0

Jika M > N

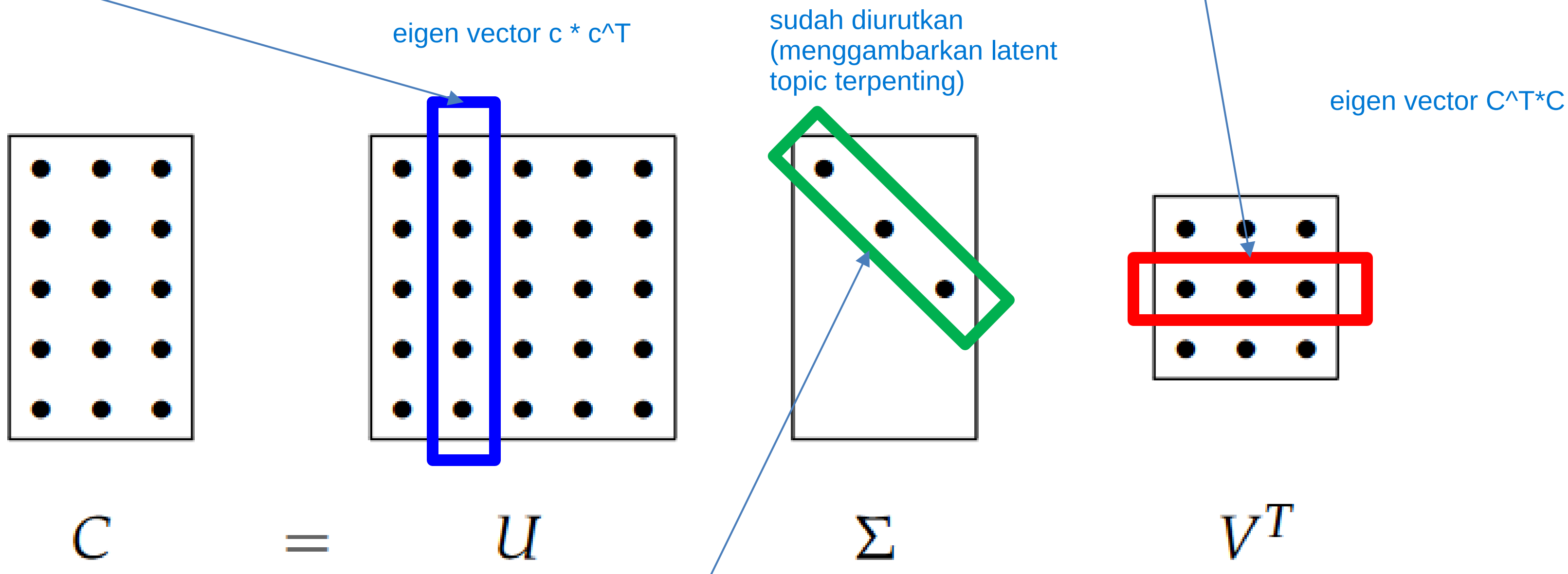
Jika M < N



Cara menghitung U, ∑, dan V^T

Kolom-kolom pada U adalah orthogonal eigenvector dari $C.C^T$

Baris-baris pada V^T adalah orthogonal eigenvector dari $C^T.C$



Singular Values (Akar kuadrat dari Eigenvalues) dari $C^T.C$ atau $C. C^T$ (sama saja).

Interpretasi Truncated SVD

Misal, C adalah term-document matrix. C didekomposisi menjadi perkalian 3 matrix.

Kolom yang di U, maksudnya topic 1, adalah eigen vector dari C * C^T

ortogonal

$$C = U \times \Sigma \times V^T$$

V^T:
- kolom: embeeding document terbaru
Misal document embeeding dokumen 1 terhadap topic 1

barisnya itu term
kolom itu document (menjadi context)

Term Document Matrix

	Doc-1	Doc-2	Doc-3	Doc-4
Term-1				
Term-2				
Term-3				
Term-4				

=

baris term
kolom: latent topic yang jadi context bagi term

Word Assignment to Topics

	Topic-1	Topic-2
Term-1		
Term-2		
Term-3		
Term-4		

×

seberapa penting latent topic

Topic Importance

	Topic-1	Topic-2
Topic-1		
Topic-2		

SIGMA (matriks diagonal)

×

Topic Distribution Across Documents

	Doc-1	Doc-2	Doc-3	Doc-4
Topic-1				
Topic-2				

ini matriks ortogonal juga

topic: campuran dokumen-dokumen

kolomnya itu C^T * C

U (Matriks ortogonal)

kita melakukan
diagonalisasi terhadap matriks covariant untuk menghilangkan regularisasi (situasi dimana fitur X korelasinya tinggi dengan fitur Y)

Gambar: https://www.datacamp.com/tutorial/discovering-hidden-topics-python

	D1	D2	D3	D4	D5
Cancer	6	0	10	1	7
Flower	2	8	1	9	0
Tumor	6	2	7	0	8
Rose	1	6	0	7	1

C

word embeeding kata cancer yang lama

U ini ortogonal kalau dihitung L2 Norm hasilnya 1 (artinya setiap term itu tegak lurus

diurutin sesuai kepentingan. Lihat deh besarnya signifikan. Artinya ada 2 latent topic yang signifikan

Σ

18.93	0	0	0
0	14.49	0	0
0	0	2.60	0
0	0	0	0.86

	1	2	3	4
Cancer	0.66	0.33	0.64	0.18
Flower	0.33	-0.71	0.18	-0.57
Tumor	0.61	0.25	-0.72	-0.19
Rose	0.24	-0.55	-0.18	0.77

U

U * S: word embeeding kata yang baru

S * V^T = document embeedng yang baru

	D1	D2	D3	D4	D5
1	0.45	0.28	0.59	0.28	0.51
2	0.10	-0.59	0.30	-0.69	0.26
3	-0.11	-0.41	0.59	0.38	-0.56
4	-0.51	-0.42	-0.12	0.44	0.58
5	-0.70	0.46	0.42	-0.30	0.04

V^T

di V^T setiap dokumen-dokumennya ini ortogonal

Kalau dihitung L2 Norm itu nilainya 1

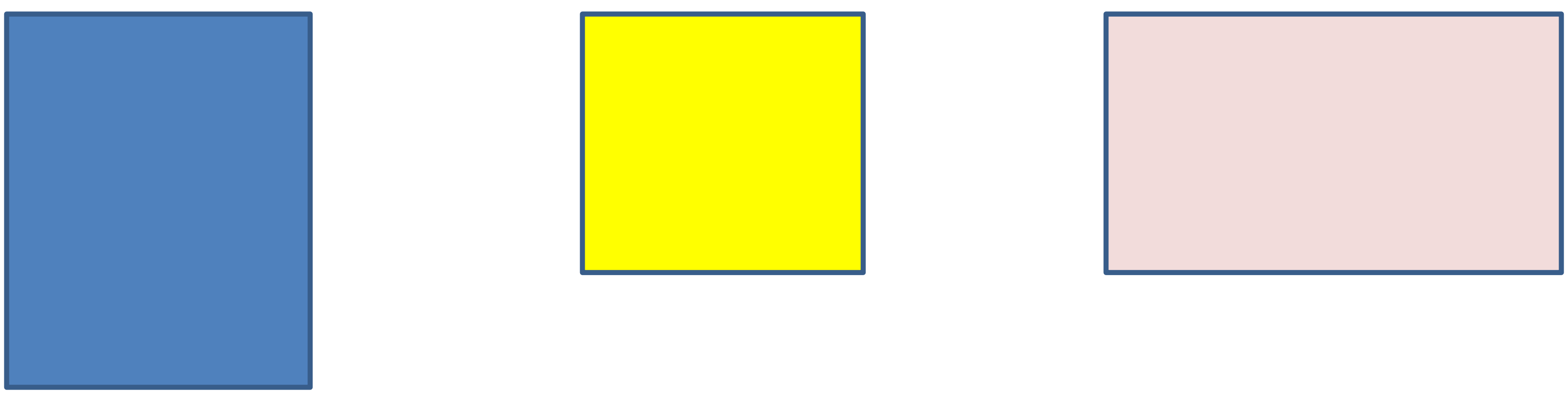
```
import numpy as np; u, s, vt = np.linalg.svd(C, full_matrices = True)
```

LSA: Low-rank approximation of C

Truncating **U, ∑, V^T** to **K dimensions** produces best possible **K** rank approximation of original matrix **C.**

Buang (jadikan nol) L - K singular values paling kecil di ∑!

$U, \mathrm{M} \times L$ $\qquad$ $\Sigma, L \times L$ $\qquad$ $V^T, L \times N$



C_k = rank-k approximation of C

Truncating **U, ∑, $\mathbf{V}^{\mathbf{T}}$** to **K dimensions** produces best possible **K** rank approximation of original matrix **C.**

Jadikan NOL L - K singular values paling kecil di ∑!

$U_k, \mathrm{M} \times K$ $\quad$ $\Sigma_k, K \times K$ $\quad$ $V_k^T, K \times N$

$$C_k = U_k \times \Sigma_k \times V_k^T$$

	D1	D2	D3	D4	D5
Cancer	6	0	10	1	7
Flower	2	8	1	9	0
Tumor	6	2	7	0	8
Rose	1	6	0	7	1

C

Σ

18.93	0	0	0
0	14.49	0	0
0	0	2.60	0
0	0	0	0.86

	1	2	3	4
Cancer	0.66	0.33	0.64	0.18
Flower	0.33	-0.71	0.18	-0.57
Tumor	0.61	0.25	-0.72	-0.19
Rose	0.24	-0.55	-0.18	0.77

U

	D1	D2	D3	D4	D5
1	0.45	0.28	0.59	0.28	0.51
2	0.10	-0.59	0.30	-0.69	0.26
3	-0.11	-0.41	0.59	0.38	-0.56
4	-0.51	-0.42	-0.12	0.44	0.58
5	-0.70	0.46	0.42	-0.30	0.04

V^T

```
import numpy as np; u, s, vt = np.linalg.svd(C, full_matrices = True)
```

Contoh: rank-2 approximation of C

C_2

	D1	D2	D3	D4	D5
Cancer	6.27	0.76	9.03	0.29	7.85
Flower	1.80	7.99	0.65	9.04	0.57
Tumor	5.70	1.15	8.09	0.79	7.03
Rose	1.29	6.08	0.38	6.89	0.33

Σ_2

18.93	0	[redacted]
0	14.49	

[redacted]

U_2

	1	2	
Cancer	0.66	0.33	[redacted]
Flower	0.33	-0.71	
Tumor	0.61	0.25	
Rose	0.24	-0.55	

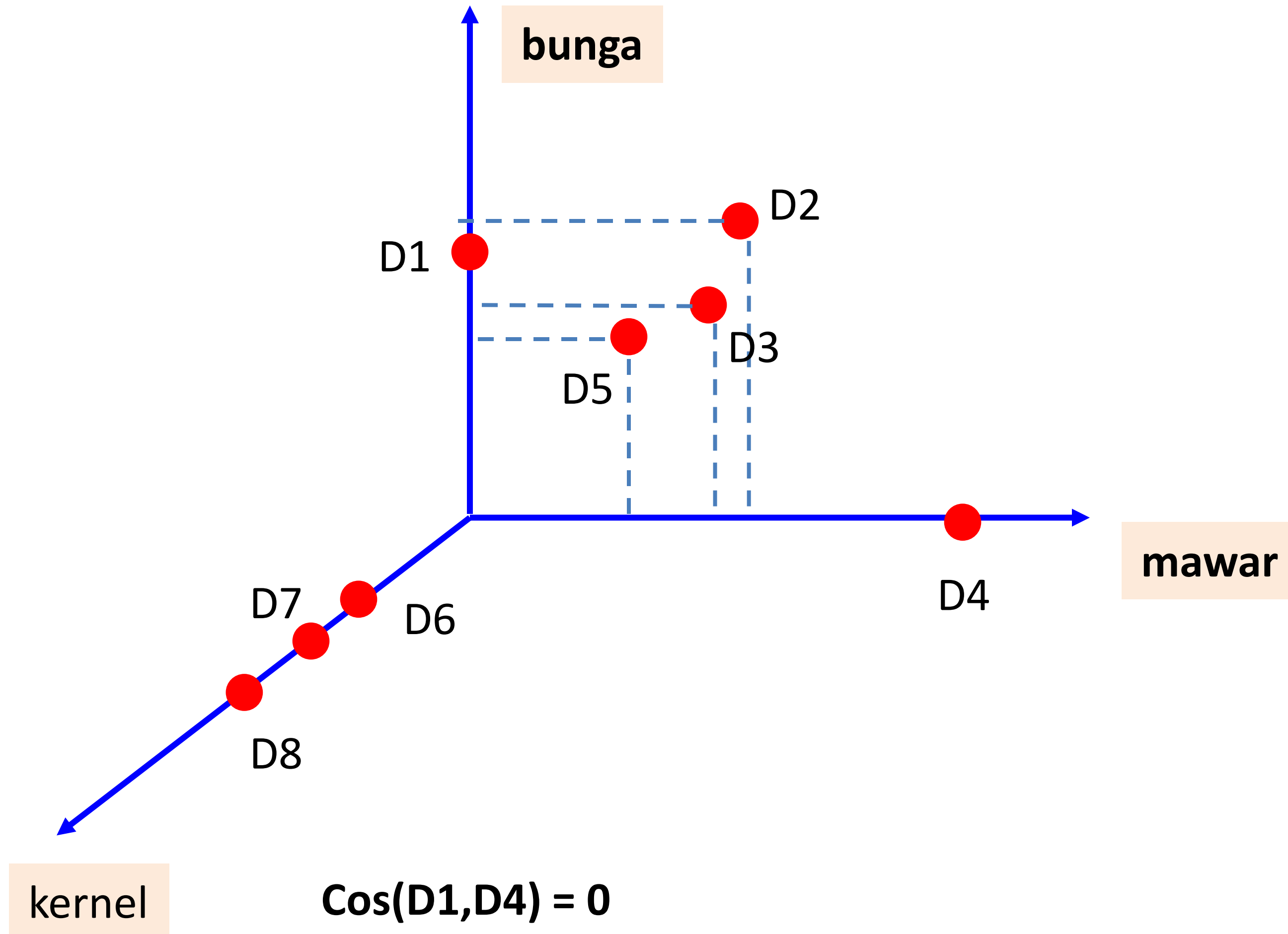
V_2^T

	D1	D2	D3	D4	D5
1	0.45	0.28	0.59	0.28	0.51
2	0.10	-0.59	0.30	-0.69	0.26

[redacted]

```
import numpy as np; u, s, vt = np.linalg.svd(C, full_matrices = True)
```

Jadi mengapa LSA/SVD berhasil menangkap “similarity” antar kata?



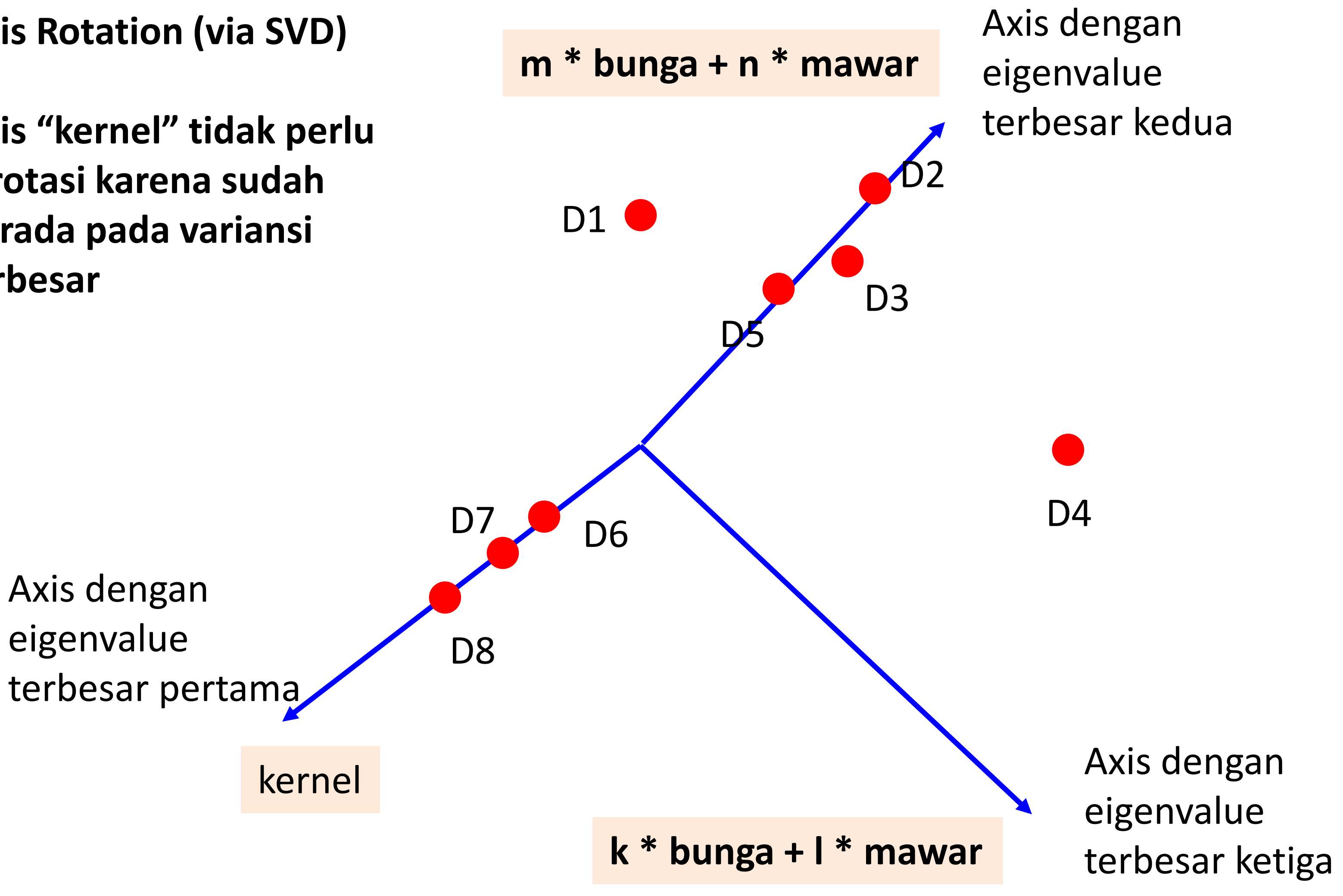
Cos(D1,D4) = 0

Padahal “bunga” dan “mawar” secara semantic “dekat”

Jadi mengapa LSA/SVD berhasil menangkap "similarity" antar kata?

Axis Rotation (via SVD)

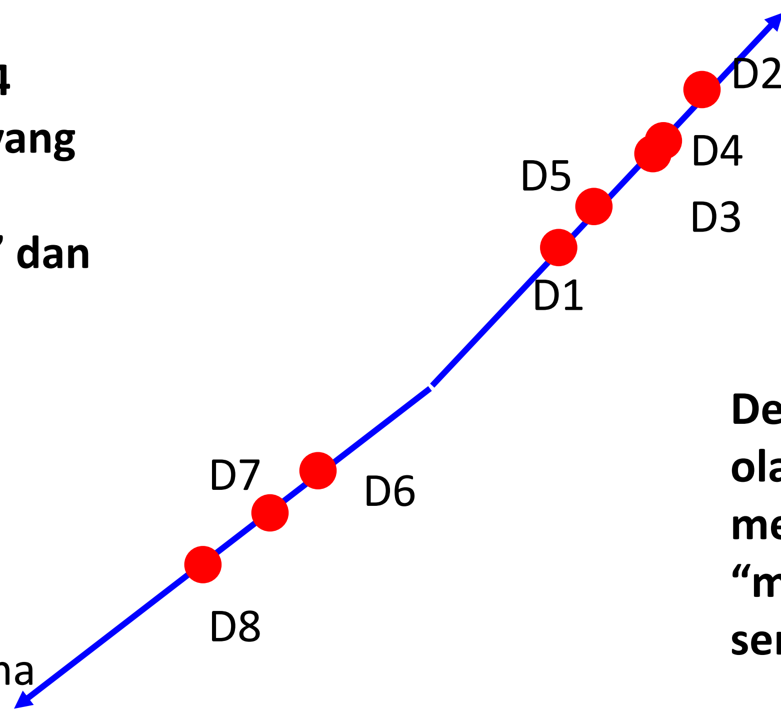
Axis "kernel" tidak perlu dirotasi karena sudah berada pada variansi terbesar



Jadi mengapa LSA/SVD berhasil menangkap “similarity” antar kata?

Buang Axis dengan eigenvalue paling kecil.

Akibatnya D1 & D4 berada pada axis yang sama, yaitu axis gabungan “bunga” dan “mawar”.



Dengan begitu, seolah-olah LSA/SVD berhasil menemukan “bunga” dan “mawar” dekat secara semantik.

Cos(D1, D4) > 0

Dimanakah *Document Vector*?

Jika ingin mendapatkan **rank-k vector representation** dari sebuah dokumen di **original vector space**:

=> Vektor kolom pada C_k

Dari Contoh sebelumnya:

	D1	**D2**	**D3**	**D4**	**D5**
Cancer	6.27	0.76	9.03	0.29	7.85
Flower	1.80	7.99	0.65	9.04	0.57
Tumor	5.70	1.15	8.09	0.79	7.03
Rose	1.29	6.08	0.38	6.89	0.33

C_2

Rank-2 Document Embedding dari D3 = [9.03, 0.65, 8.09, 0.38]

Dimanakah *Document Vector*?

Jika ingin mendapatkan **rank-k vector representation** dari sebuah dokumen di **vector space baru (rank-k subspace)**:

=> Vektor kolom pada $\Sigma_k \times V_k^T$

Dari Contoh sebelumnya:

2-Dimensional Document Embedding dari D3 adalah

Σ_2

18.93	0
0	14.49

$\times$ V_2^T

D3
0.59
0.30

$=$

D3
11.17
4.35

Dimanakah *Term Vector (Word Embedding)*?

Jika ingin mendapatkan **rank-k vector representation** dari sebuah kata di **vector space baru (rank-k subspace)**:

=> Vektor baris pada $U_k \times \Sigma_k$

Dari Contoh sebelumnya:

2-Dimensional Document Embedding dari "Cancer" adalah

$$\begin{array}{lcc} \textbf{Cancer} & 0.66 & 0.33 \end{array} \times \Sigma_2 \begin{bmatrix} 18.93 & 0 \\ 0 & 14.49 \end{bmatrix} = \begin{array}{lcc} \textbf{Cancer} & 12.49 & 4.78 \end{array}$$

Jika ada Query, bagaimana hitung sim(Q, D)?

Vektor **Query** yang masih berada di dimensi awal perlu dipetakan ke **LSA (Semantic) Space** yang berukuran **K** dengan cara:

$$q_k = U_k^T \times q$$

Proof?

$$sim(q, d) = sim(q_k, d_k)$$

Vektor kolom pada $\boldsymbol{\Sigma_k \times V_k^T}$ yang terasosiasi dengan **d**.

Uji Pemahaman Terhadap LSA

Refleksi

- Apa arti “distributional” pada “distributional word representations”?
- Apa itu “sparse word representations”? Apa contohnya?
- Apa itu “dense word representations”? Apa kelebihannya?
- Apa itu term-document matrix? Apa saja yang dapat digunakan untuk mengisi setiap cell pada matrix tersebut?

Refleksi

Misal, kita melakukan **Singular Value Decomposition** terhadap term-document matrix C (berukuran M x N):

$$C = U \times \Sigma \times V^T$$

- Informasi apa yang ada pada **U**?
- Informasi apa yang ada pada **Σ**? Apa makna singular values?
- Informasi apa yang ada pada $\mathbf{V^T}$?
- Bagaimana menghitung **U**, **Σ**, dan $\mathbf{V^T}$?
- Dimanakah *vector representation of words* berada?
- Bagaimana caranya mendapatkan *low-dimensional vector of words (dense)* berukuran **K (< min(M, N))**?
- Bagaimana caranya mendapatkan *low-dimensional vector of documents (dense)* berukuran **K (< min(M, N))**?

Refleksi

Perhatikan Term-Document matrix berikut (setiap cell berisi informasi TF):

	D1	**D2**	**D3**	**D4**	**D5**
Cancer	6	0	10	1	7
Flower	2	8	1	9	0
Tumor	6	2	7	0	8
Rose	1	6	0	7	1

Sebenarnya apa sih yang dilakukan LSA?

When forced to squeeze the terms/documents down to a k-dimensional space, the SVD should bring to- gether terms with similar co-occurrences.

Refleksi

Perhatikan Term-Document matrix berikut (setiap cell berisi informasi TF):

Kira-kira ada berapa latent topics yang penting di sini?

	D1	D3	D5	D2	D4
Cancer	6	10	7	0	1
Tumor	6	7	8	2	0
Flower	2	1	0	8	9
Rose	1	0	1	6	7

LSA akan memindahkan term (baris) dan dokumen (kolom) **sehingga vektor-vektor yang mirip akan berdekatan (ter-cluster)**.

Setelah ter-cluster, LSA kemudian bisa menemukan, kira-kira ada berapa **“latent topics penting”** yang ada pada koleksi dokumen tersebut.

	D1	D2	D3	D4	D5
Cancer	6	0	10	1	7
Flower	2	8	1	9	0
Tumor	6	2	7	0	8
Rose	1	6	0	7	1

Itulah mengapa hanya ada 2 singular values yang sangat besar dibandingkan 2 yang terkecil.

18.93	0	0	0
0	14.49	0	0
0	0	2.60	0
0	0	0	0.86

	1	2	3	4
Cancer	0.66	0.33	0.64	0.18
Flower	0.33	-0.71	0.18	-0.57
Tumor	0.61	0.25	-0.72	-0.19
Rose	0.24	-0.55	-0.18	0.77

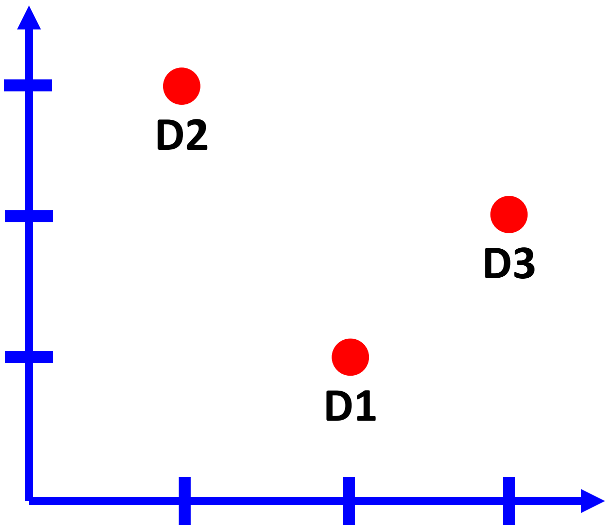
	D1	D2	D3	D4	D5
1	0.45	0.28	0.59	0.28	0.51
2	0.10	-0.59	0.30	-0.69	0.26
3	-0.11	-0.41	0.59	0.38	-0.56
4	-0.51	-0.42	-0.12	0.44	0.58
5	-0.70	0.46	0.42	-0.30	0.04

```
import numpy as np; u, s, vt = np.linalg.svd(C, full_matrices = True)
```

Simple Explanation of LSA

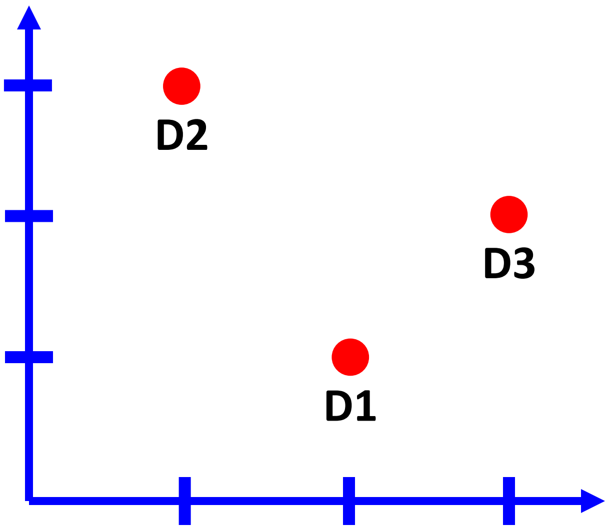
Misal, kita mempunyai term-document matrix berikut (bobot TF):

$$C = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \end{bmatrix}$$



Misal, kita mempunyai term-document matrix berikut (bobot TF):

$$C = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \end{bmatrix}$$



Misal, kita lakukan SVD:

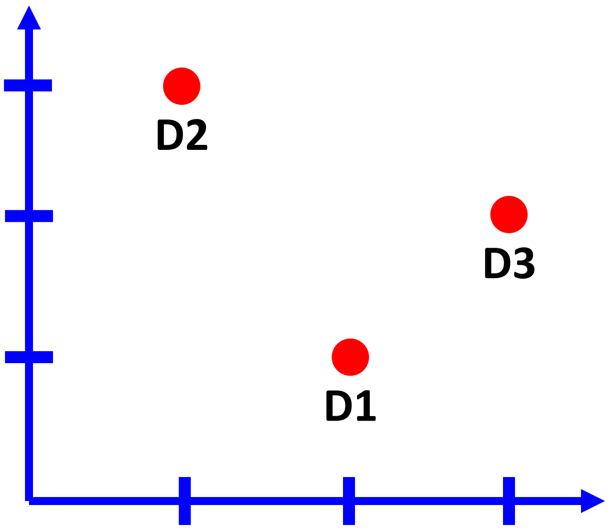
$$U = \begin{bmatrix} 0.71 & -0.71 \\ 0.71 & 0.71 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 1.7 & 0 \end{bmatrix}$$

$$V^T = \begin{bmatrix} 0.42 & 0.57 & 0.71 \\ -0.41 & 0.82 & -0.41 \\ -0.81 & -0.11 & 0.8 \end{bmatrix}$$

Misal, kita mempunyai term-document matrix berikut (bobot TF):

$$C = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \end{bmatrix}$$



Rank-1 approximation dari C:

$$U = \begin{bmatrix} 0.71 & -0.71 \\ 0.71 & 0.71 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 5 & 0 & 0 \\ 0 & \mathbf{0} & 0 \end{bmatrix}$$

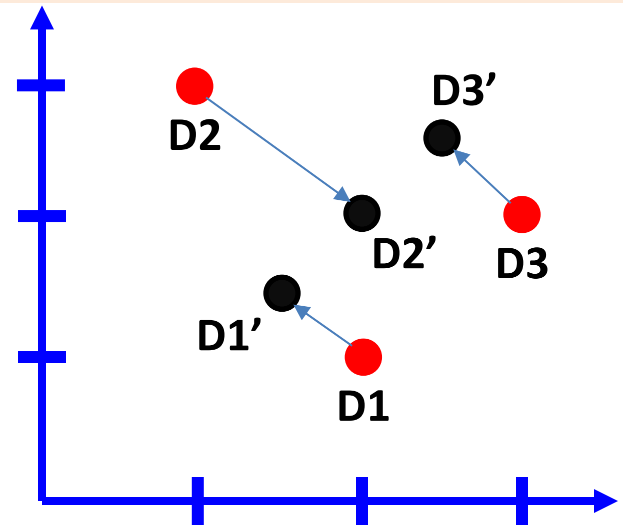
$$V^T = \begin{bmatrix} 0.42 & 0.57 & 0.71 \\ -0.41 & 0.82 & -0.41 \\ -0.81 & -0.11 & 0.8 \end{bmatrix}$$

$$C_1 = U \times \Sigma_1 \times V^T = \begin{bmatrix} 1.5 & 2 & 2.5 \\ 1.5 & 2 & 2.5 \end{bmatrix}$$

Misal, kita mempunyai term-document matrix berikut (bobot TF):

We move the points to **the smallest squared distance.**

$$C = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \end{bmatrix}$$



Rank-1 approximation dari C (Rank-1 Document Vector):

$$U = \begin{bmatrix} 0.71 & -0.71 \\ 0.71 & 0.71 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 5 & 0 & 0 \\ 0 & \mathbf{0} & 0 \end{bmatrix}$$

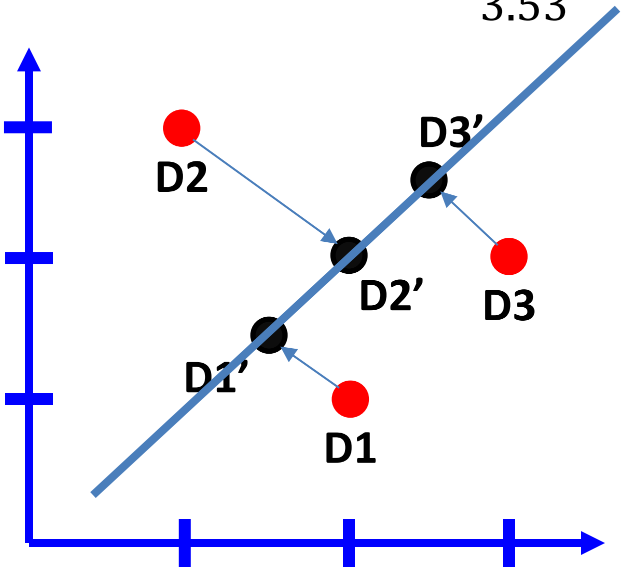
$$V^T = \begin{bmatrix} 0.42 & 0.57 & 0.71 \\ -0.41 & 0.82 & -0.41 \\ -0.81 & -0.11 & 0.8 \end{bmatrix}$$

$$C_1 = U \times \Sigma_1 \times V^T = \begin{bmatrix} 1.5 & 2 & 2.5 \\ 1.5 & 2 & 2.5 \end{bmatrix}$$

Misal, kita mempunyai term-document matrix berikut (bobot TF):

$$C = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \end{bmatrix}$$

$$y = \frac{3.53}{3.53} x = x$$



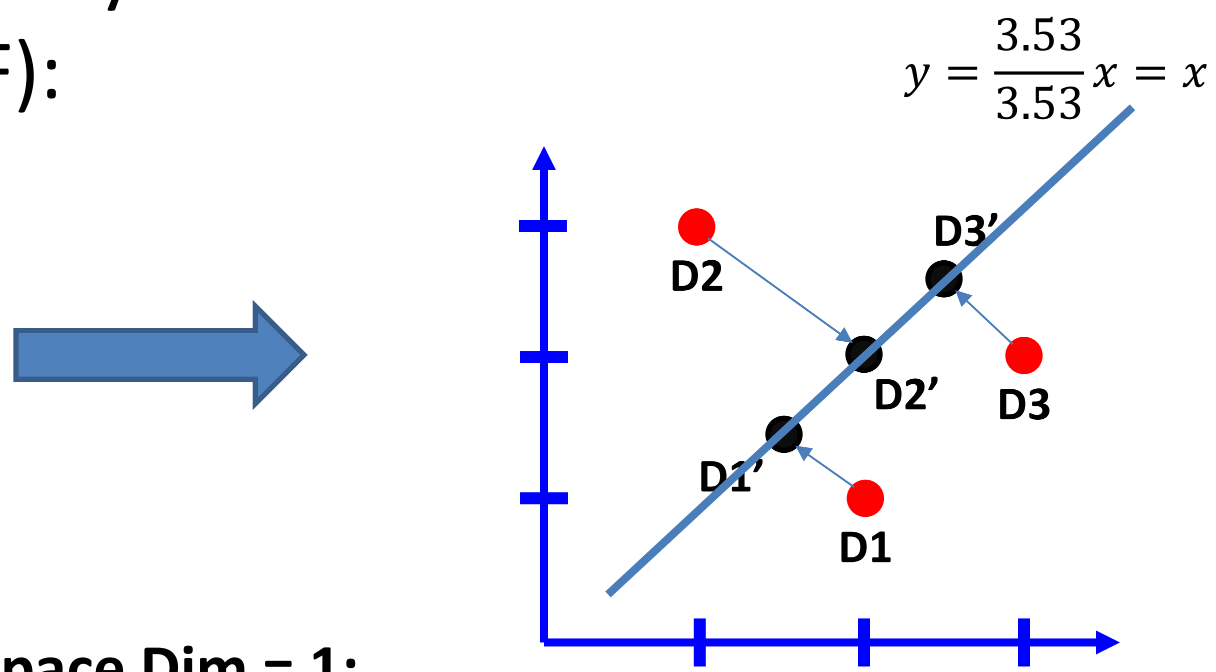
Vektor Kata:

$$U \times \Sigma_1 = \begin{bmatrix} 3.53 & 0 & 0 \\ 3.53 & 0 & 0 \end{bmatrix}$$

$$C_1 = U \times \Sigma_1 \times V^T = \begin{bmatrix} 1.5 & 2 & 2.5 \\ 1.5 & 2 & 2.5 \end{bmatrix}$$

Misal, kita mempunyai term-document matrix berikut (bobot TF):

$$C = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \end{bmatrix}$$



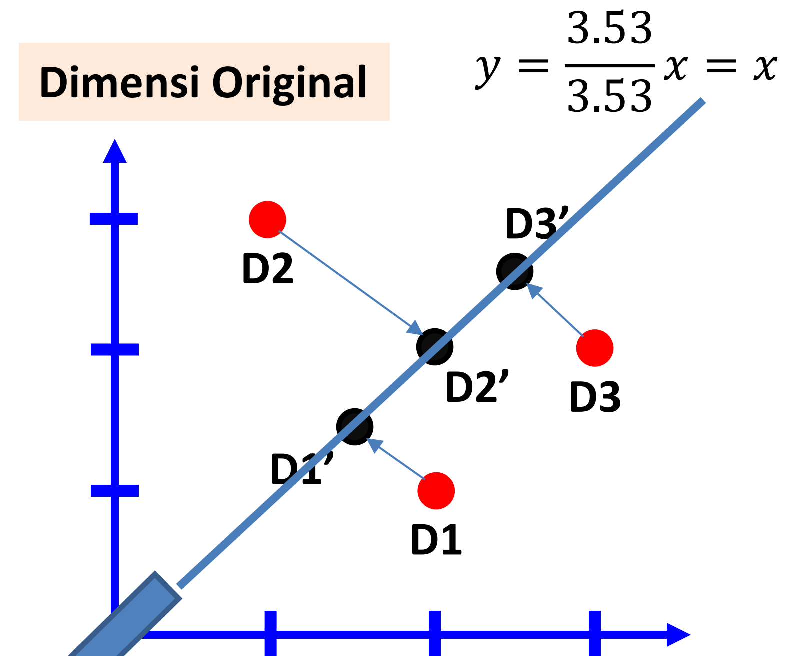
Vektor Dokumen di Space Dim = 1:

$$\Sigma_1 \times V^T = \begin{bmatrix} 2.1 & 2.8 & 3.5 \\ 0 & 0 & 0 \end{bmatrix}$$

$$C_1 = U \times \Sigma_1 \times V^T = \begin{bmatrix} 1.5 & 2 & 2.5 \\ 1.5 & 2 & 2.5 \end{bmatrix}$$

Misal, kita mempunyai term-document matrix berikut (bobot TF):

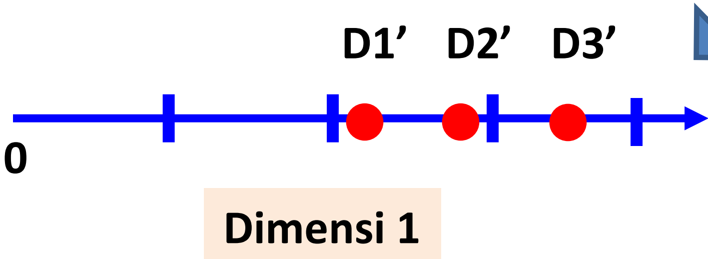
$$C = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 3 & 2 \end{bmatrix}$$



Vektor Dokumen di Space Dim = 1:

$$\Sigma_1 \times V^T = \begin{bmatrix} 2.1 & 2.8 & 3.5 \\ 0 & 0 & 0 \end{bmatrix}$$

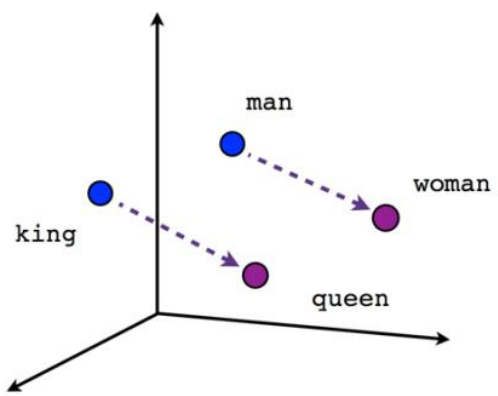
$$C_1 = U \times \Sigma_1 \times V^T = \begin{bmatrix} 1.5 & 2 & 2.5 \\ 1.5 & 2 & 2.5 \end{bmatrix}$$



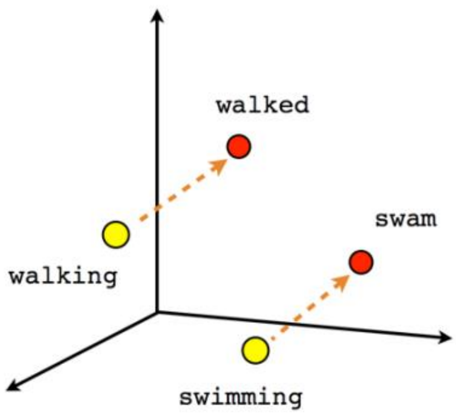
Neural Embeddings

Why Word Embeddings?

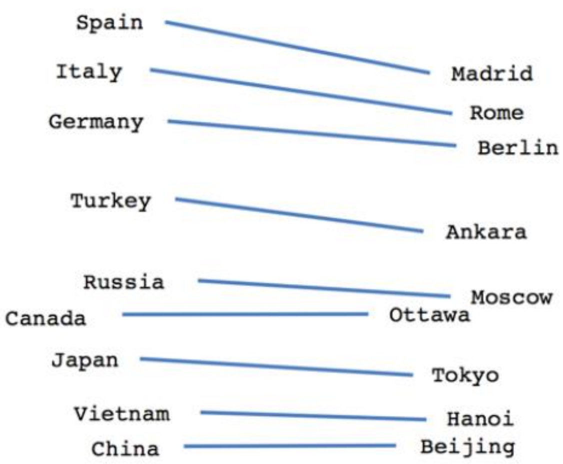
- Can capture the rich relational structure of the lexicon



Male-Female



Verb tense



Country-Capital

https://www.tensorflow.org/tutorials/word2vec

Word Embeddings

- Any technique that maps a word (or phrase) from **it's original high-dimensional sparse input** space to **a lower-dimensional dense vector space**.

- Vectors whose relative similarities correlate with semantic similarity

- Such vectors are used both as an end in itself (for computing similarities between terms), and as a representational basis for downstream NLP tasks, such as POS tagging, NER, text classification, etc.

Continuous Representation of Words

The Differences:

- In **information retrieval,** LSA and topic models use ***documents as contexts***.
  - Capture semantic relatedness (**“boat” and “water”**)

- **Distributional semantic models** use ***words as contexts*** (more natural in linguistic perspective)
  - Capture semantic similarity **(“boat” and “ship”)**

Word Embedding

DSMs or Word Embeddings

- Count-based model
  - first collecting context vectors and then reweighting these vectors based on various criteria
- Predictive-based model (neural network)
  - vector weights are directly set to optimally predict the contexts in which the corresponding words tend to appear
  - Similar words occur in similar contexts, the system naturally learns to assign similar vectors to similar words.

Distributional Semantic Models

Other classification based on (Baroni et al., ACL 2014)

- Count-based models
  - Simple VSMs
  - Singular Value Decomposition (Golub & VanLoan, 1996)
  - Non-negative Matrix Factorization (Lee & Seung, 2000)
- Predictive-based models (**neural network**)
  - Self Organizing Map
  - Bengio et al's **Word Embedding (2003)**
  - Mikolov et al's **Word2Vec (2013)**

Baroni et la., "Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors". ACL 2014

Word Analogy Task

- **Father** is to **Mother** as **King** is to ______ ?
- **Good** is to **Best** as **Smart** is to ______ ?
- **Indonesia** is to **Jakarta** as **Malaysia** is to _____ ?

- It turns out that the previous Word-Context based vector model is good for such analogy task.

$$V_{king} - V_{father} + V_{mother} = V_{queen}$$

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. NIPS, 1–9

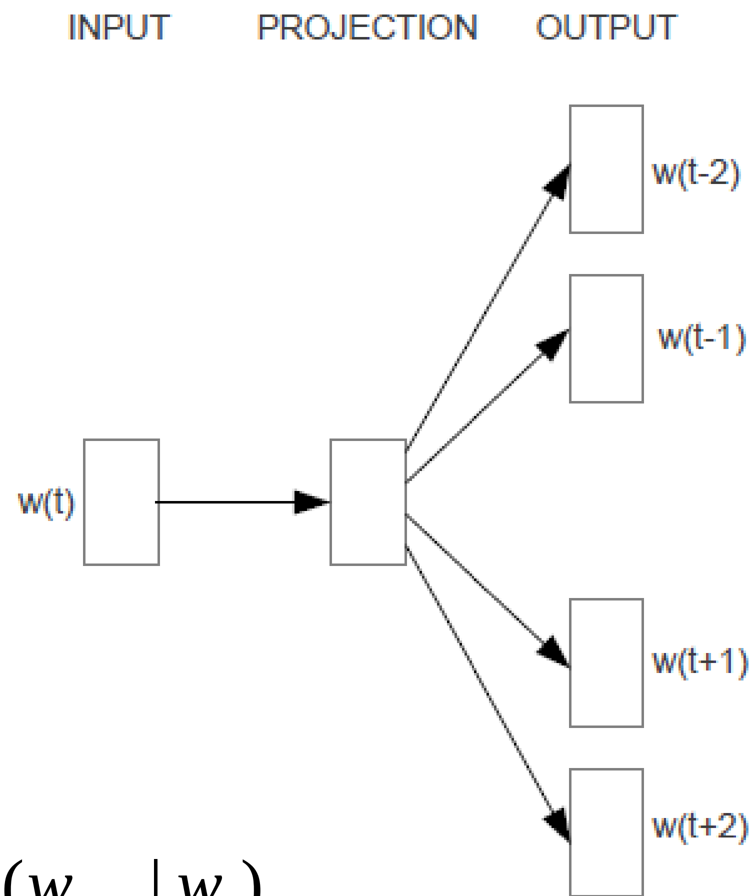
Word2Vec (Mikolov et al., 2013)

Word2Vec

- One of the most popular Word Embedding models nowadays!
- There are two types of models:
  - Skip-Gram Model
  - Continuous Bag of Words Model (**CBOW**)

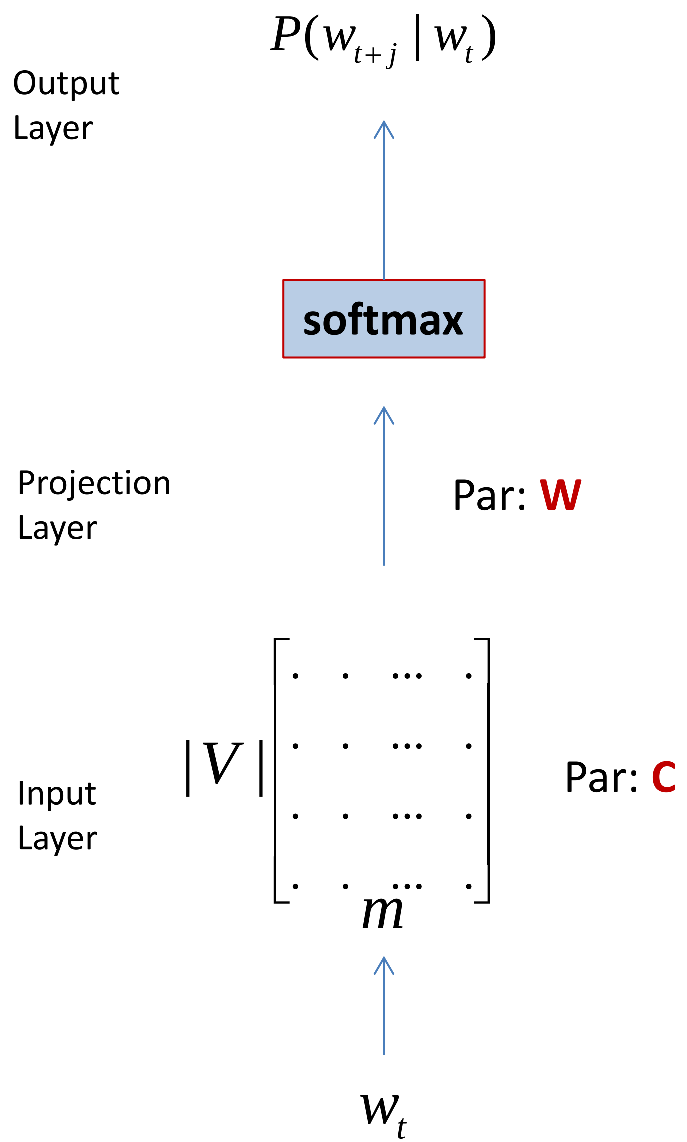
Skip-Gram

- **N-Gram language model** only looks at previous words as a context for predictions.
- This model tries to maximize classification of a word based on another word in the same sentence.
- Use each current word as an input to a log-linear classifier with continuous projection layer, and predict words within a certain range before and after the current word.



We seek a model for $P(w_{t+j} \mid w_t)$

Skip-Gram



Feed-Forward Process

$$P(w_{t+j} \mid w_t) = \frac{\exp(y_{w_{t+j}})}{\sum_{i \in V} \exp(y_i)}$$

$$\boldsymbol{y_{w_t} = W.x}$$

$$x = C(w_t)$$

Total Parameters:

$$\theta = \{W, C\}$$

$$C \in R^{|V| \times m}$$

$$W \in R^{m \times |V|}$$

Skip-Gram

Sudut Pandang Lain

Sebuah kata **t** terasosiasi dengan dua buah vector:

- Vector ketika **t** berperan sebagai **center word** (baris pada matriks $\boldsymbol{C}$)
- Vector ketika **t** berperan sebagai **context word** (kolom pada matriks $\boldsymbol{W}$)

Skip-Gram

Sudut Pandang Lain

Misal Vocab = [cancer, tumor, flower, rose]

Matriks C
Center word
embeddings of tumor

Low-dimensional Dense Embeddings of Tumor

$$\begin{bmatrix}0\\1\\0\\0\end{bmatrix}^T \times \begin{bmatrix}0.1 & 0.2 & 0.2\\ 0.1 & 0.1 & 0.3\\ 0.8 & 0.4 & 0.1\\ 0.9 & 0.3 & 0.1\end{bmatrix} = \boxed{\begin{bmatrix}0.1\\0.1\\0.3\end{bmatrix}^T}$$

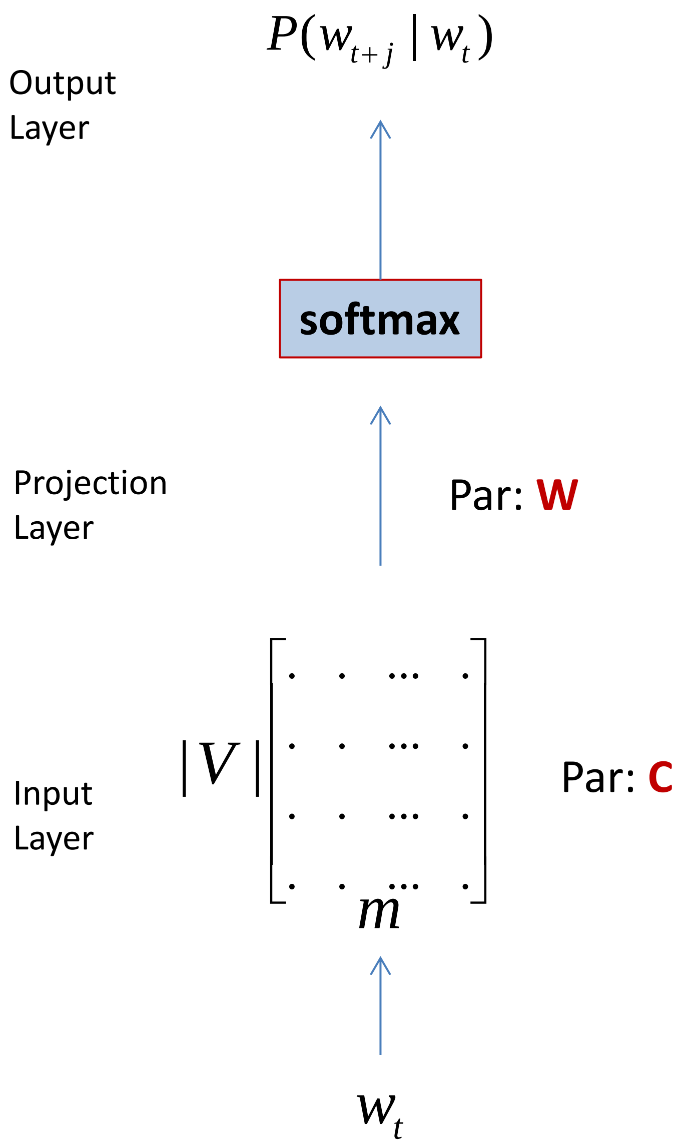
$$\boldsymbol{P(rose|tumor) = 0.25}$$

One-hot representation of tumor

Matriks W
Context word
embeddings of **rose**

$$softmax\left\{\begin{bmatrix}0.1\\0.1\\0.3\end{bmatrix}^T \times \begin{bmatrix}0.3 & 0.2 & 0.3 & 0.1\\ 0.2 & 0.2 & 0.4 & 0.7\\ 0.2 & 0.3 & 0.3 & 0.1\end{bmatrix}\right\} = \begin{bmatrix}0.24\\0.25\\0.26\\ \boxed{0.25}\end{bmatrix}$$

Skip-Gram



Dengan kata lain, Skip-Gram juga bisa dinyatakan dengan:

$$\boldsymbol{P(w_{t+j}|w_t) = \frac{exp(s(w_{t+j}, w_t))}{\sum_{i \in V} exp(s(w_i, w_t))}}$$

$$\boldsymbol{s(w_i, w_t) = center(w_t)^T . context(w_i)}$$

Baris di matriks C

Kolom di matriks W

Where are the Word Embeddings?

- The previous model actually aims at building the language model.
  - So, **where is the Word Embedding model that we need?**

- **The answer is:** If you just need the Word Embedding model, you just need the matrix **C**

Where are the Word Embeddings?

After all parameters (including **C**) are optimized, then we can use **C** to map a word into its vector !

A Word **w**

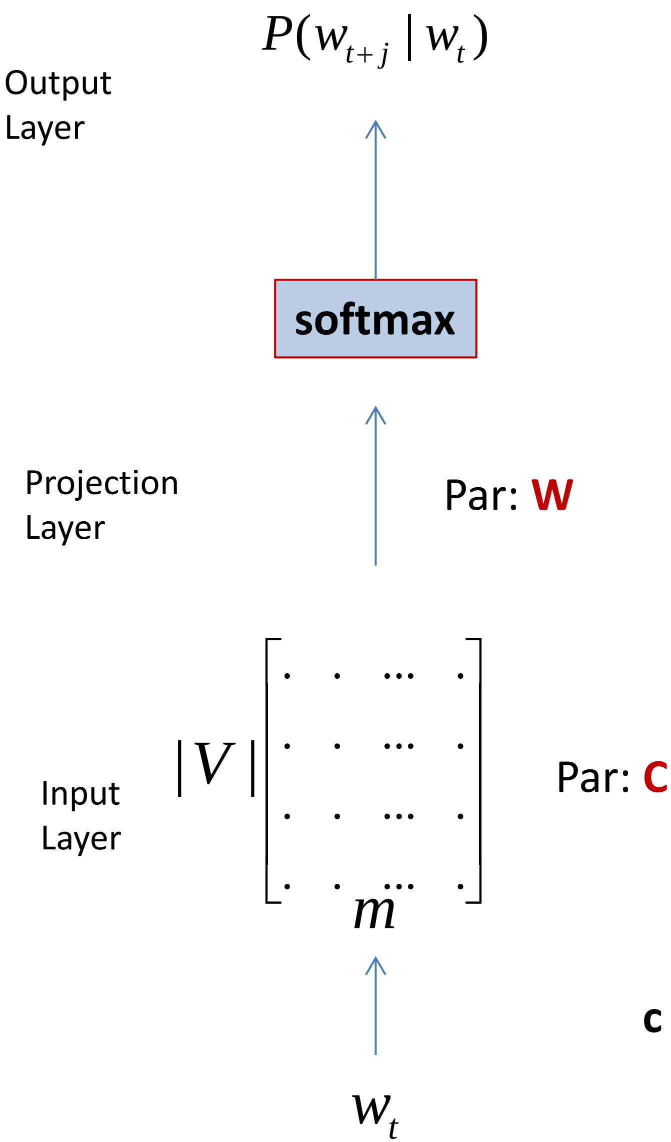
$$w \in V$$

$$\Rightarrow \quad |V| \underbrace{\begin{bmatrix} \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \end{bmatrix}}_{m} \quad \Rightarrow \quad C(w) = \begin{bmatrix} 0.2 \\ 0.31 \\ \vdots \\ 0.76 \end{bmatrix}$$

$$C \in R^{|V| \times m}$$

$$C(w) \in R^{m}$$

Skip-Gram



Training

Training is achieved by looking **θ** that maximizes the following Cost Function:

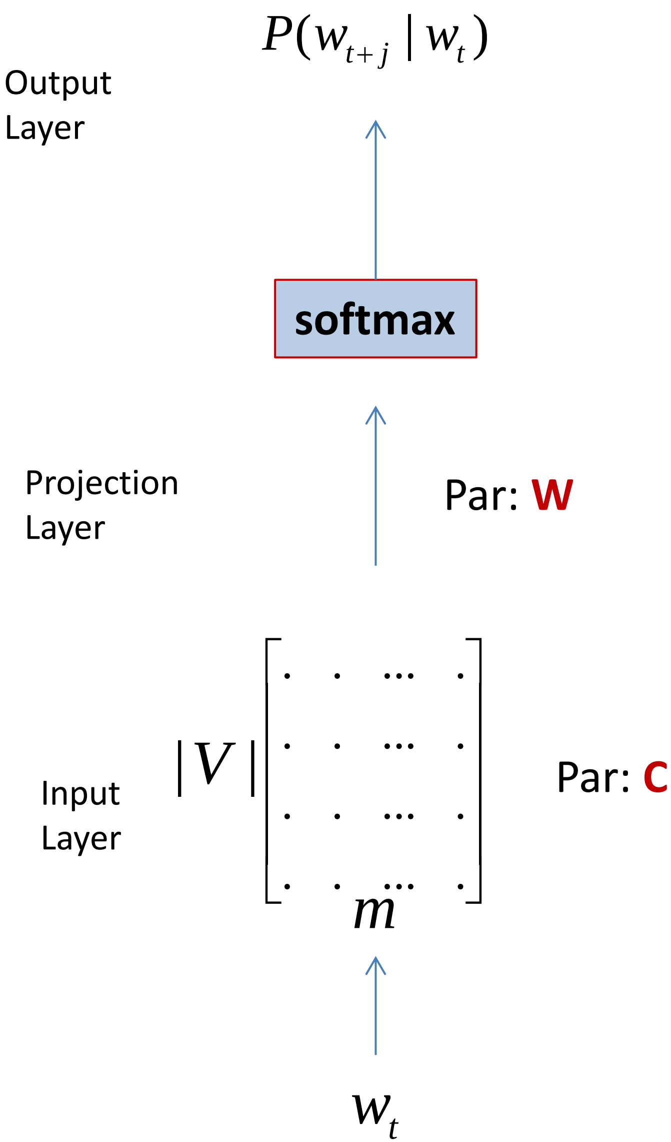
Given training data $w_1, w_2, w_3, \dots, w_{T-1}, w_T$

$$J(w_1 \dots w_T; \theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \le j \le c, j \ne 0} \log P(w_{t+j} | w_t) + \boxed{R(\theta)}$$

Regularization Terms

c is the maximum distance of the words, or **WINDOW size**

Skip-Gram



Training

Training is achieved by looking **θ** that maximizes the following Cost Function:

Given training data $w_1, w_2, w_3, \ldots, w_{T-1}, w_T$

$$J(w_1 \ldots w_T; \theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \le j \le c,\, j \ne 0} \log P(w_{t+j} \mid w_t) + R(\theta)$$

Proses optimasi (memaksimalkan) fungsi J = menggunakan Categorical Cross Entropy sebagai loss function pada ujung softmax layer.

Reference: https://www.tensorflow.org/tutorials/word2vec

Skip-Gram

How to develop dataset?

For example, let's consider the following dataset:

the quick brown fox jumped over the lazy dog

Using **c = 1 (or window = 1)**, we then have dataset:

```
([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox), ...
```

Therefore, our **(input, output)** dataset becomes:

```
(quick, the), (quick, brown), (brown, quick), (brown, fox), ...
```

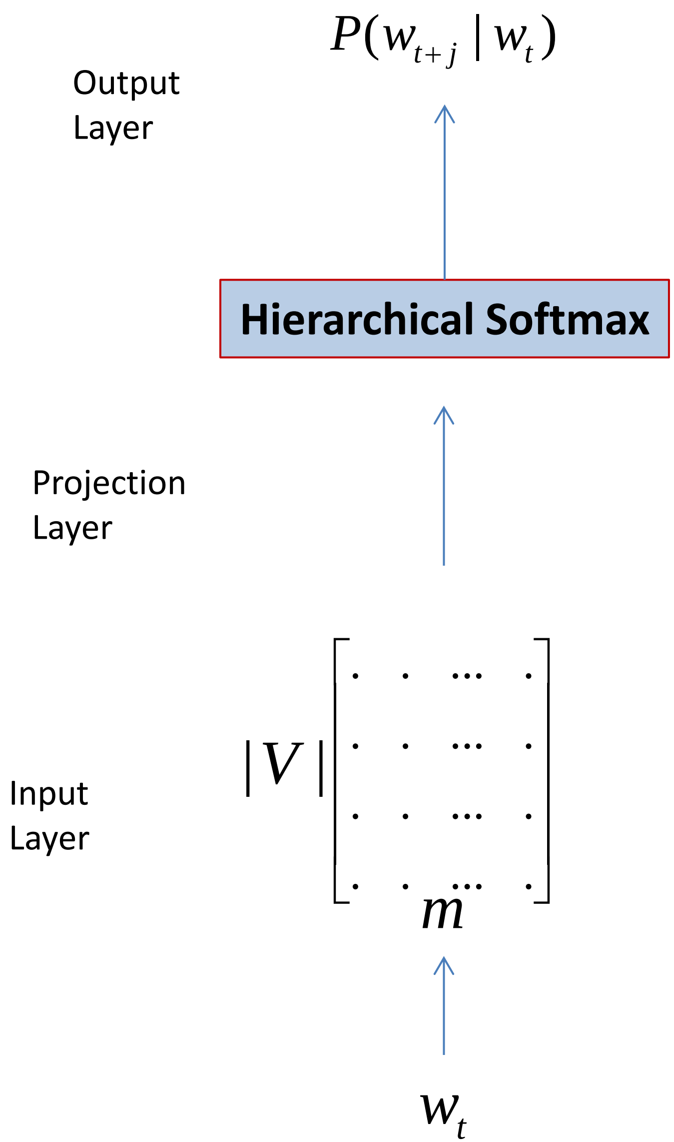
For example, $P(w_{t+1} = brown \mid w_t = quick)$

Use this dataset to learn $P(w_{t+j} \mid w_t)$

So that, the cost function is optimized!

$$J(w_1 \ldots w_T; \theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \le j \le c, j \ne 0} \log P(w_{t+j} \mid w_t)$$

Skip-Gram



Training

Actually, if we use **vanilla softmax**, then the computational complexity **per instance** (**Q**) is still costly.

$$Q = D \times (m + m \times |V|)$$

D is the maximum distance of the words

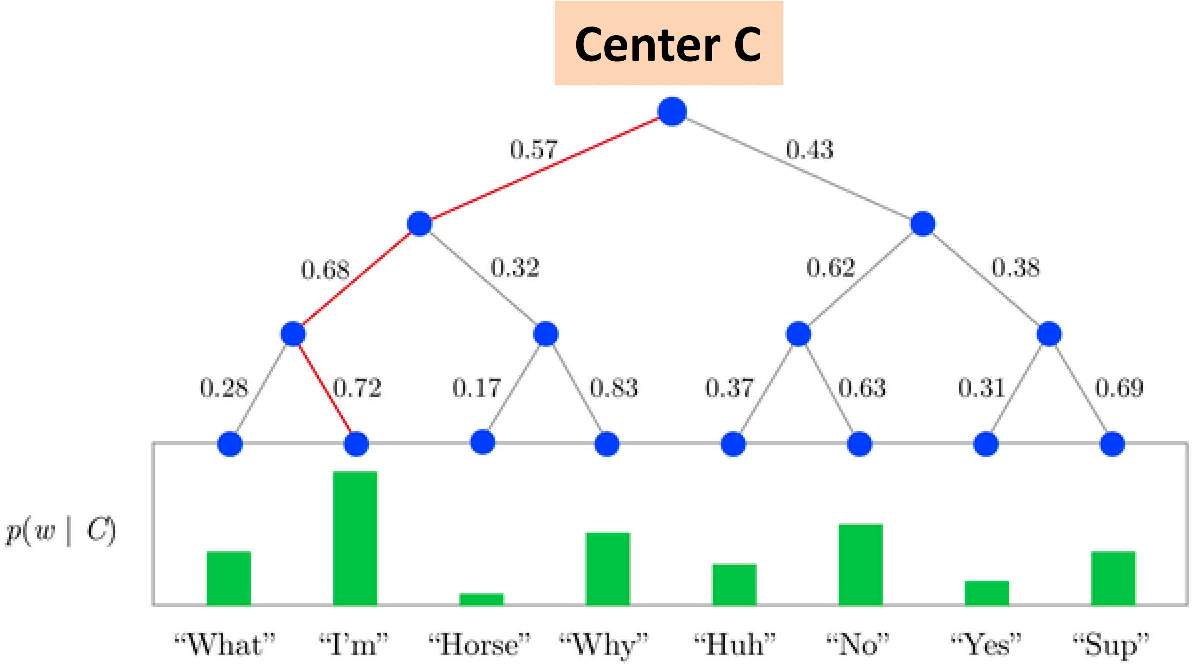
To solve this problem, they use **Hierarchical Softmax layer**. This layer uses a **binary tree representation** of the output layer with |V| units.

$$Q = D \times (m + m \times \log_2 |V|)$$

(Morin & Bengio, 2005)

Hierarchical Softmax

- A **multi-layer binary tree**
- The probability of a word is calculated through the **product of probabilities on each edge on the path to that node**.
- It is **O(log n)**, compared to **O(n)** for vanilla softmax.



Hierarchical Softmax

- Misal, diberikan sebuah kata input **“kernel”** sebagai center, kita ingin memprediksi sebuah kata **konteks** **w**.
- Melakukan traversal Huffman Tree menggunakan kode binary yang diassign ke kata **w**.
- Perhitungan probability hanya melibatkan sekumpulan kecil node pada jalur root ke kata **w**.

Hierarchical Softmax

Huffman Tree

Meminimalkan expected search length

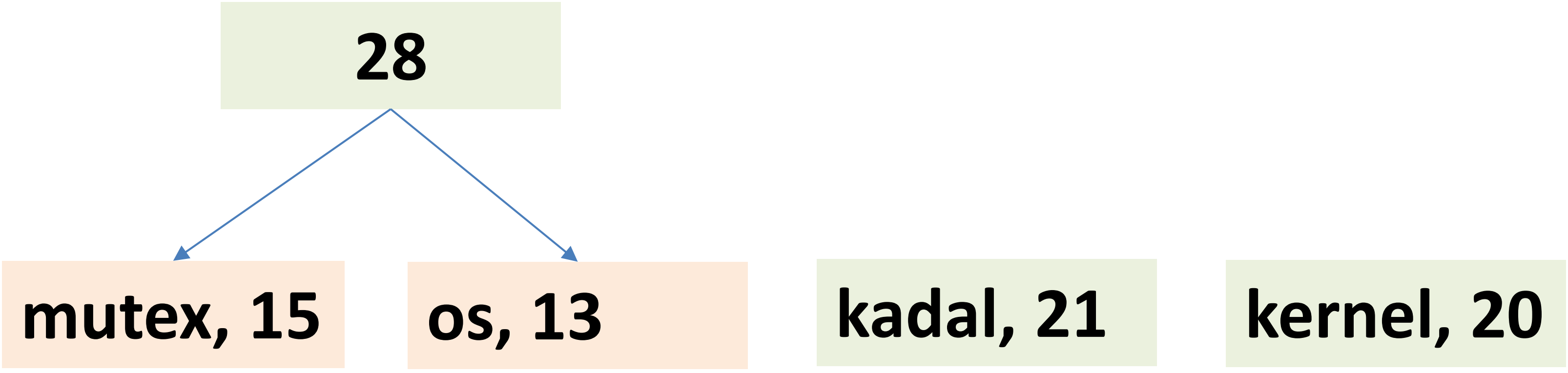
Gabung dua buah kata dengan frekuensi paling kecil; dan proses ini dilakukan terus.

kadal, 21 **mutex, 15** **os, 13** **kernel, 20**

Hierarchical Softmax

Huffman Tree

Gabung dua buah kata dengan frekuensi paling kecil; dan proses ini dilakukan terus.



Hierarchical Softmax

Huffman Tree

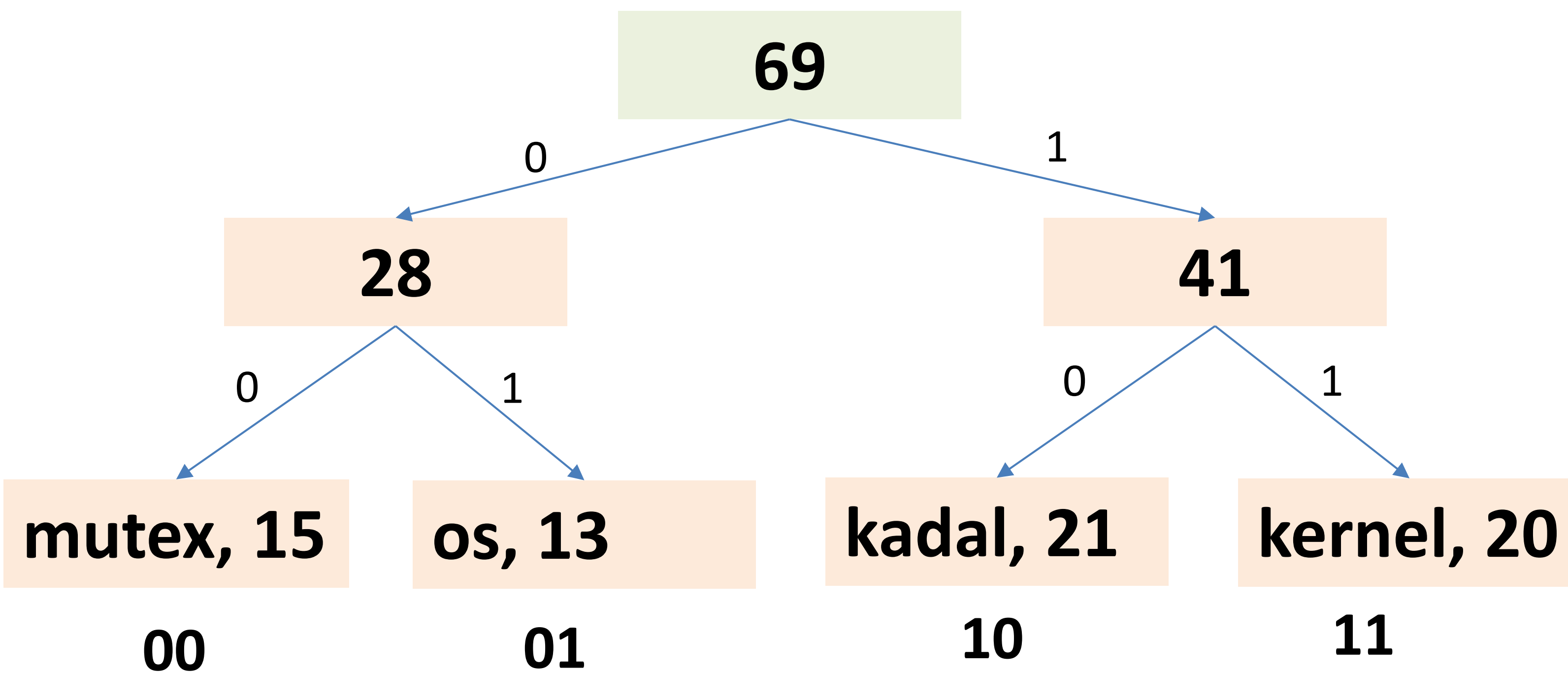
Gabung dua buah kata dengan frekuensi paling kecil; dan proses ini dilakukan terus.

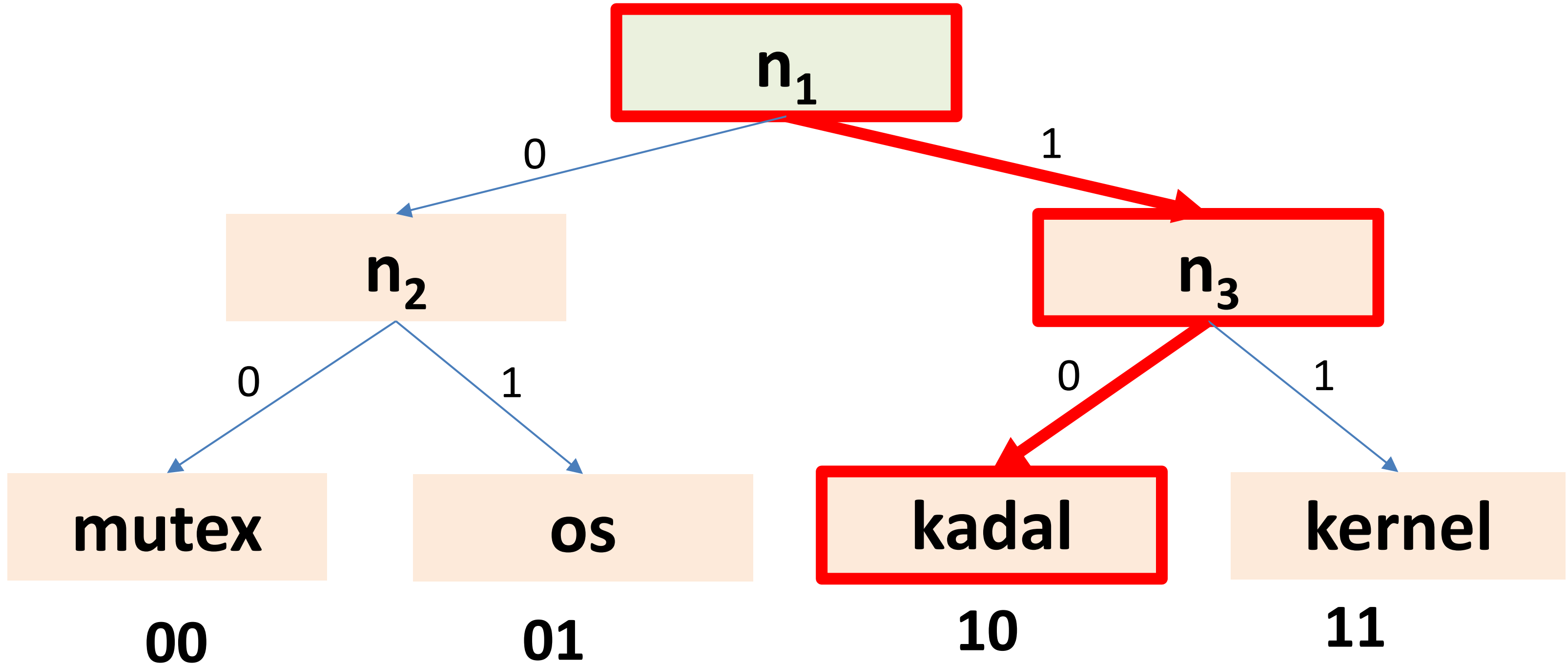
Hierarchical Softmax

Huffman Tree

Assign kode biner ke setiap kata di vocabulary

Gabung dua buah kata dengan frekuensi paling kecil; dan proses ini dilakukan terus.





$$P(kadal|c) = P_{n_1}(right|c) \times P_{n_3}(left|c)$$
$$= (1 - P_{n_1}(left|c)) \times P_{n_3}(left|c)$$

$$P_n(left|c) = \sigma(v_n^T . center(c))$$

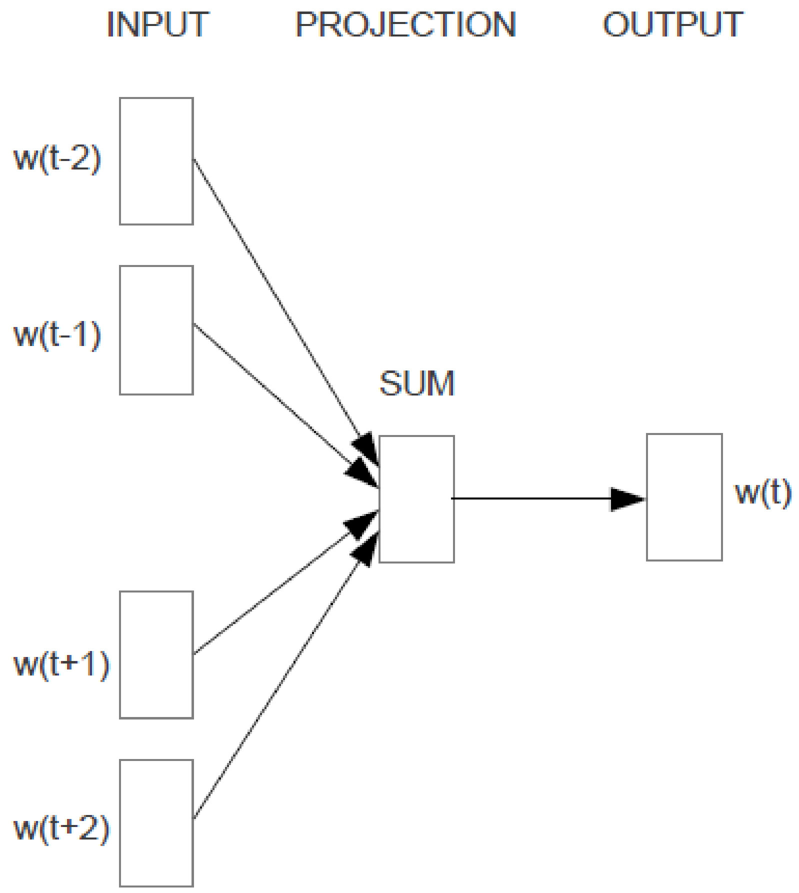
Trainable vector for node n

Hidden layer representation of the center c

Continuous Bag-of-Words

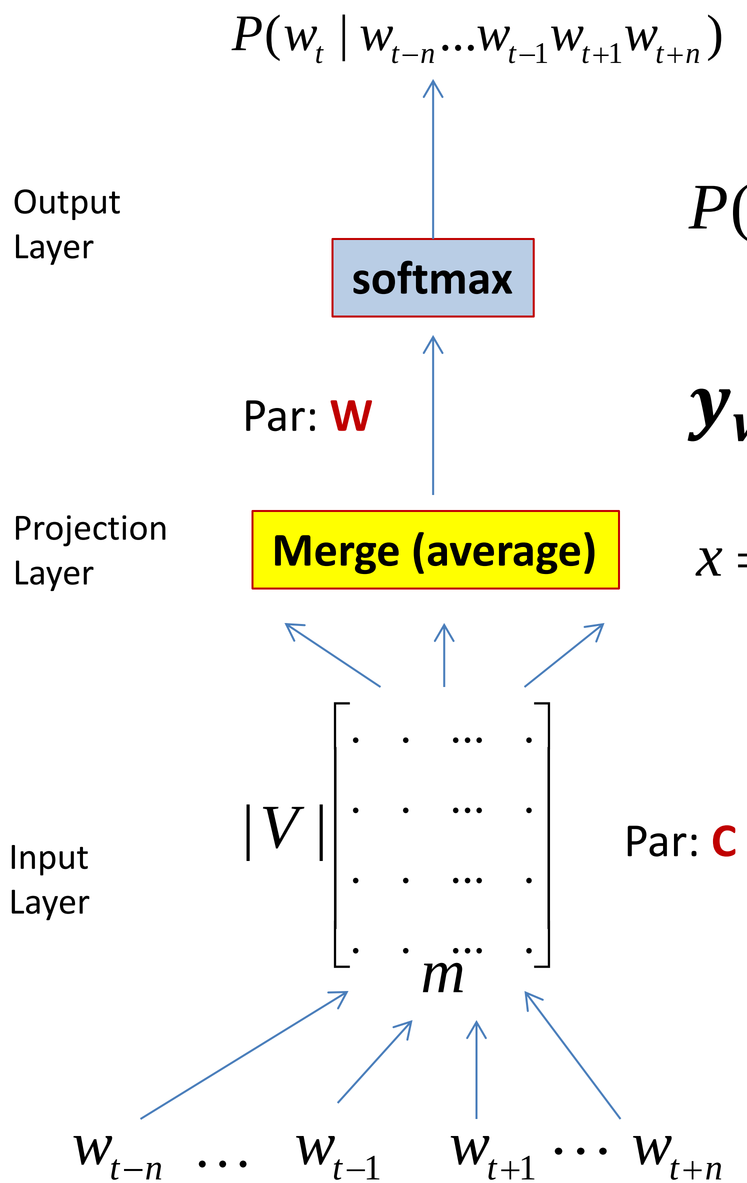
- **Mikolov's CBOW** looks at **n words** before and after the target words.
  - Non-linear hidden layer is also removed.
  - All word vectors get projected into the same position (**their vectors are averaged**)

- "Bag-of-Words" is because **the order of words in the history does not influence the projection**.



We seek a model for $P(w_t \mid w_{t-n} \dots w_{t-1} w_{t+1} \dots w_{t+n})$

Continuous Bag-of-Words



Feed-Forward Process

$$P(w_t \mid w_{t-n} \ldots w_{t-1} w_{t+1} \ldots w_{t+n}) = \frac{\exp(y_{w_t})}{\sum_{i \in V} \exp(y_i)}$$

$$\boldsymbol{y_{w_t} = W . x}$$

$$x = average(C(w_{t-n}), C(w_{t-1}), \ldots, C(w_{t+1}), C(w_{t+n}))$$

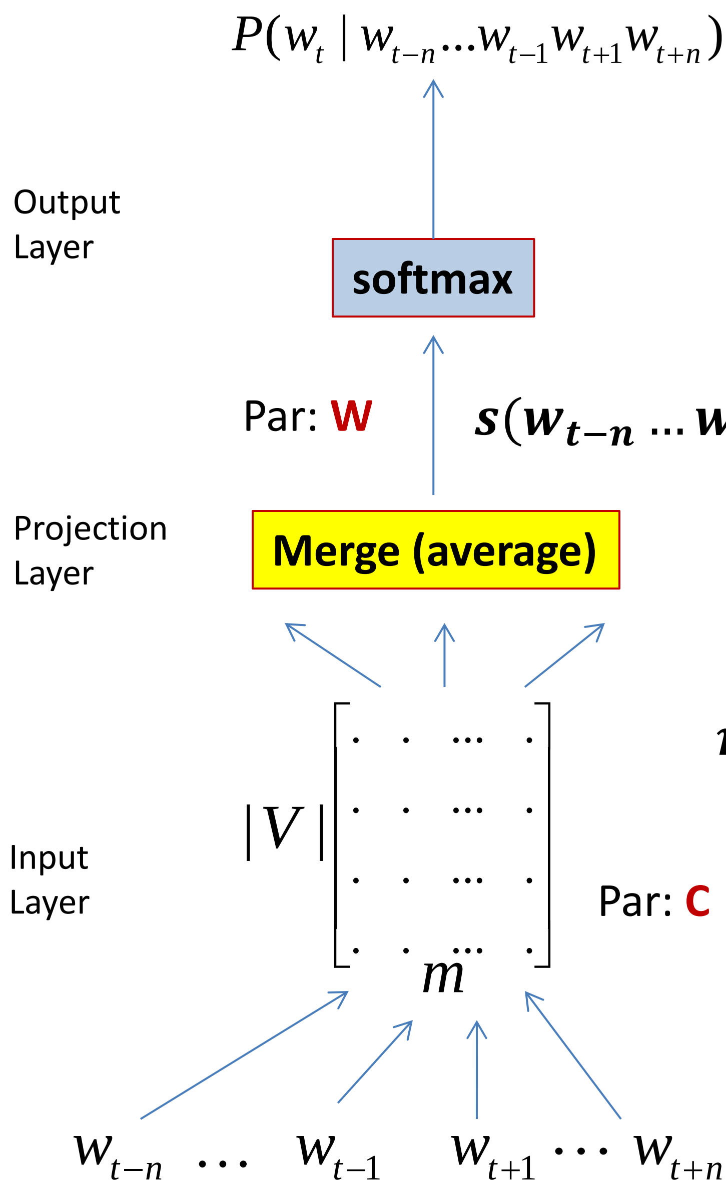
Total Parameters:

$$\theta = \{W, C\}$$

$$C \in R^{|V| \times m}$$

$$W \in R^{m \times |V|}$$

Continuous Bag-of-Words



Dengan kata lain, CBOW juga bisa dinyatakan dengan:

$$P(w_t|w_{t-n} \ldots w_{t+n}) = \frac{exp(s(w_{t-n} \ldots w_{t+n}, w_t))}{\sum_{i \in V} s(w_{t-n} \ldots w_{t+n}, w_i)}$$

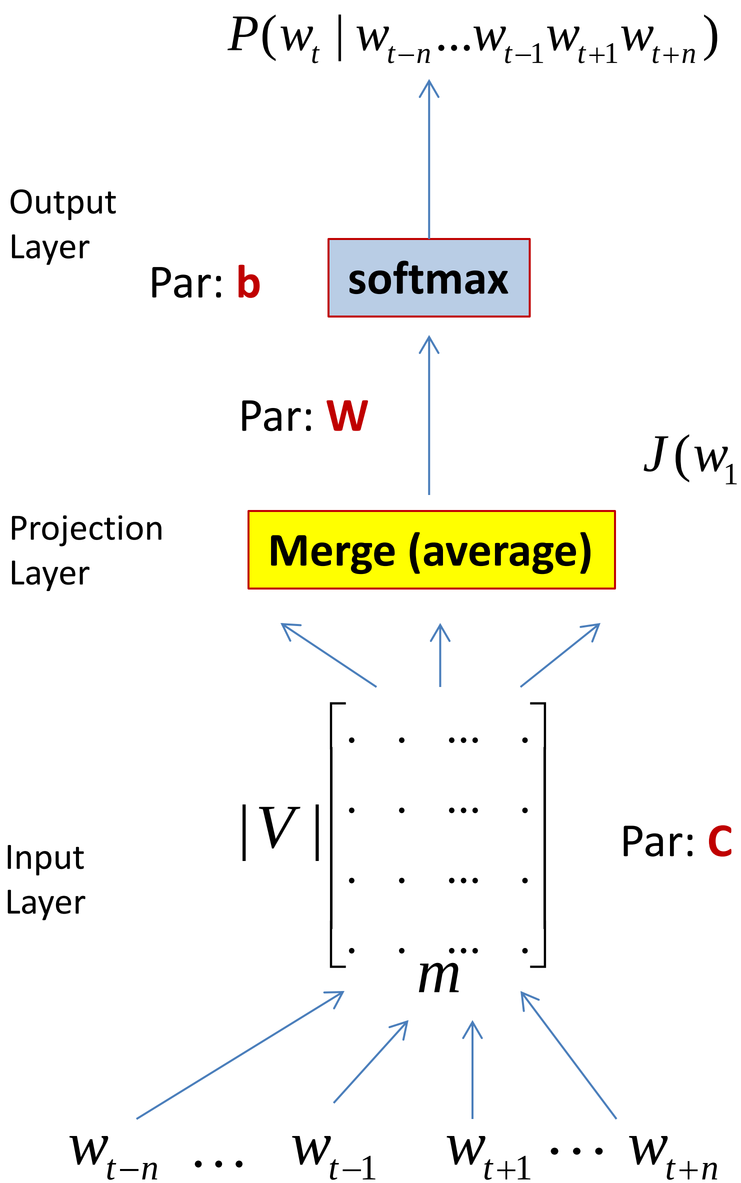
$$s(w_{t-n} \ldots w_{t+n}, w_i) = merge(w_{t-n} \ldots w_{t+n})^T . context(w_i)$$

Kolom di matriks W

$$merge(w_{t-n} \ldots w_{t+n}) = \frac{1}{2n} \sum_{t-n \le j \le t+j} center(w_j)$$

Baris di matriks C

Continuous Bag-of-Words



Training

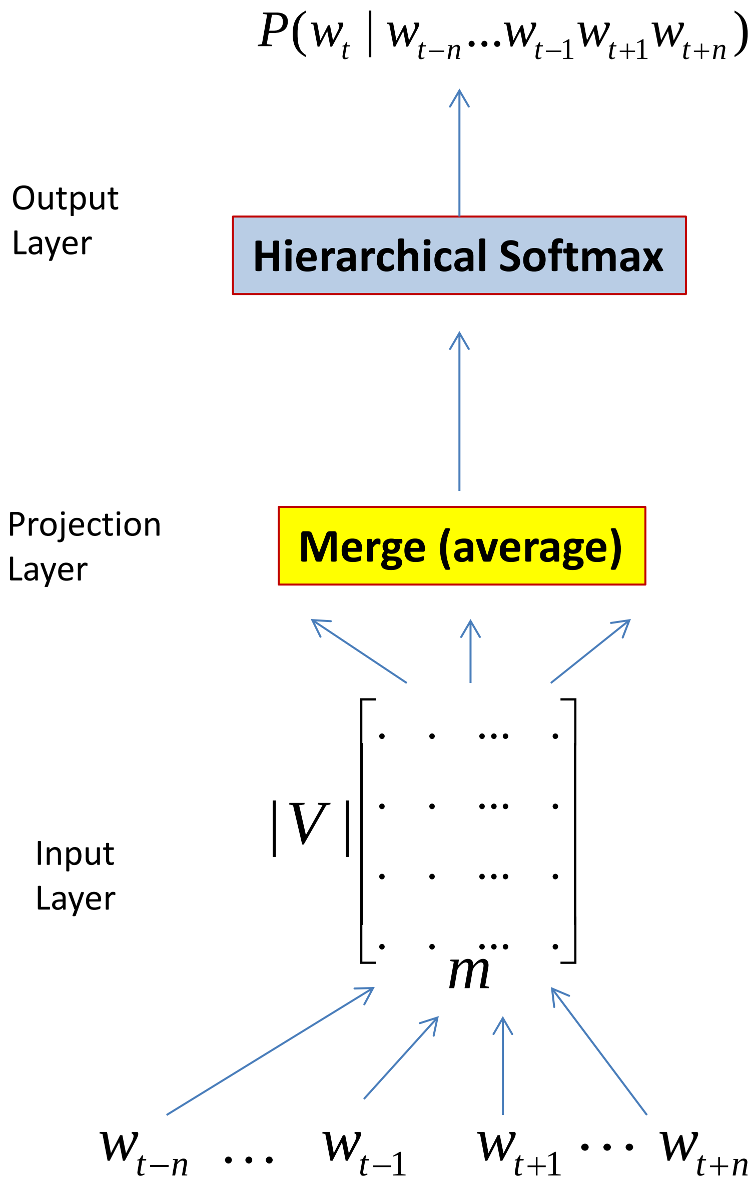
Training is achieved by looking **θ** that maximizes the following Cost Function:

Given training data $w_1, w_2, w_3, \dots, w_{T-1}, w_T$

$$J(w_1 \dots w_T; \theta) = \frac{1}{T} \sum_{t=1}^{T} \log P(w_t \mid w_{t-n} \dots w_{t-1} w_{t+1} \dots w_{t+n}) + \boxed{R(\theta)}$$

Regularization terms

Continuous Bag-of-Words



Training

Actually, if we use **vanilla softmax**, then the computational complexity per instance (**Q**) is still costly.

$$Q = N \times m + m \times |V|$$

To solve this problem, they use **Hierarchical Softmax layer**. This layer uses a **binary tree representation** of the output layer with |V| units.

$$Q = N \times m + m \times \log_2 |V|$$

(Morin & Bengio, 2005)

FastText (Bojanowski et al., TACL)

Subword Information

- Ekstensi dari Word2Vec -> bisa handle rare words atau kata yang out-of-vocabulary

- Perbedaannya adalah setiap kata tidak langsung diproyeksikan ke sebuah vector.
- Pada FastText, setiap kata “dipecah” dahulu menjadi “subwords”, dan setiap subword diproyeksikan ke vector.

Subword Information

- $\mathbf{w_t}$ = where

- Subwords dengan window = 3
  - <wh
  - whe
  - her
  - ere
  - re>
  - where

Kata asli itu sendiri diikutsertakan ke dalam set of ngrams

FastText – Skip-Gram Model

Given a word, what is the probability of its context word?

$$P(w_c|w_t) = softmax\big(s(w_c, w_t)\big) = \frac{\exp(s(w_c, w_t))}{\sum_{j=1}^{|Vocab|} \exp(s(w_j, w_t))}$$

Ingat bahwa, untuk kasus Word2Vec Skip-Gram biasa:

$$s(w_c, w_t) = center(\mathrm{w_t})^{\mathrm{T}}.\,context(\mathrm{w_c})$$

Untuk FastText, vektor kata adalah KOMBINASI dari vector Subwords !

Subword Information

- $\mathbf{w_t}$ = where
- Subwords dengan window = 3
  - <wh -> $z_{g1} = [0.3, 0.1, \ldots]$
  - whe -> $z_{g2} = [0.1, 0.4, \ldots]$
  - her -> $z_{g3} = [0.5, 0.9, \ldots]$
  - ere -> $z_{g4} = [0.5, 0.5, \ldots]$
  - re> -> $z_{g5} = [0.3, 0.1, \ldots]$
  - where -> $z_{g6} = [0.1, 0.3, \ldots]$

Center embedding dari subword “<wh”

Kata asli itu sendiri diikutsertakan ke dalam set of ngrams

Subword Information

- $\mathbf{w_t}$ = where

Representasi vector sebuah kata merupakan "sum" dari semua vector subwords.

- Subwords dengan window = 3
  - <wh -> $z_{g1} = [0.3, 0.1, \ldots]$
  - whe -> $z_{g2} = [0.1, 0.4, \ldots]$
  - her -> $z_{g3} = [0.5, 0.9, \ldots]$
  - ere -> $z_{g4} = [0.5, 0.5, \ldots]$
  - re> -> $z_{g5} = [0.3, 0.1, \ldots]$
  - where -> $z_{g6} = [0.1, 0.3, \ldots]$

$$s(w_c, w_t) = \sum_g z_g^{\mathrm{T}}.context(\mathrm{w_c})$$

Kata asli itu sendiri diikutsertakan ke dalam set of ngrams