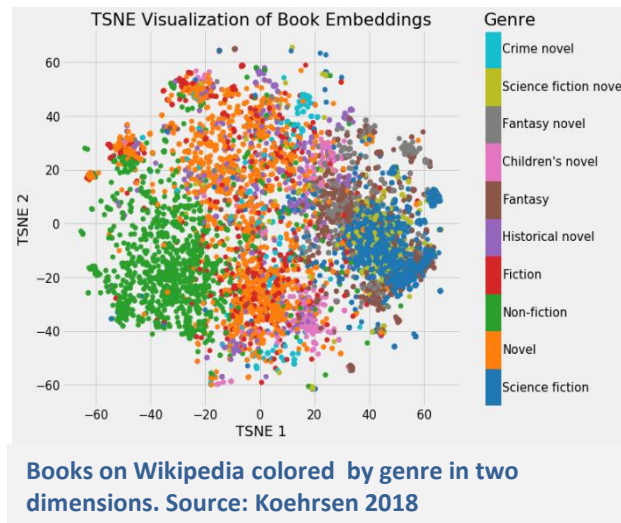


Μηχανική Μάθηση και Ανάκτηση Πληροφορίας

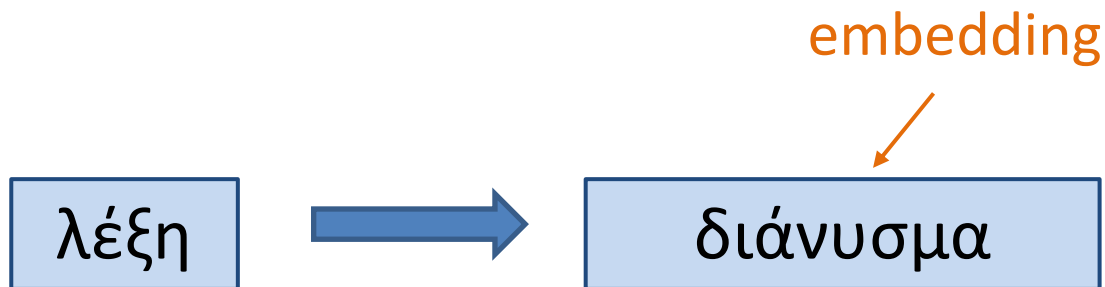
Word embeddings

Τι θα δούμε σήμερα

- Διανυσματική αναπαράσταση λέξεων (word embeddings)

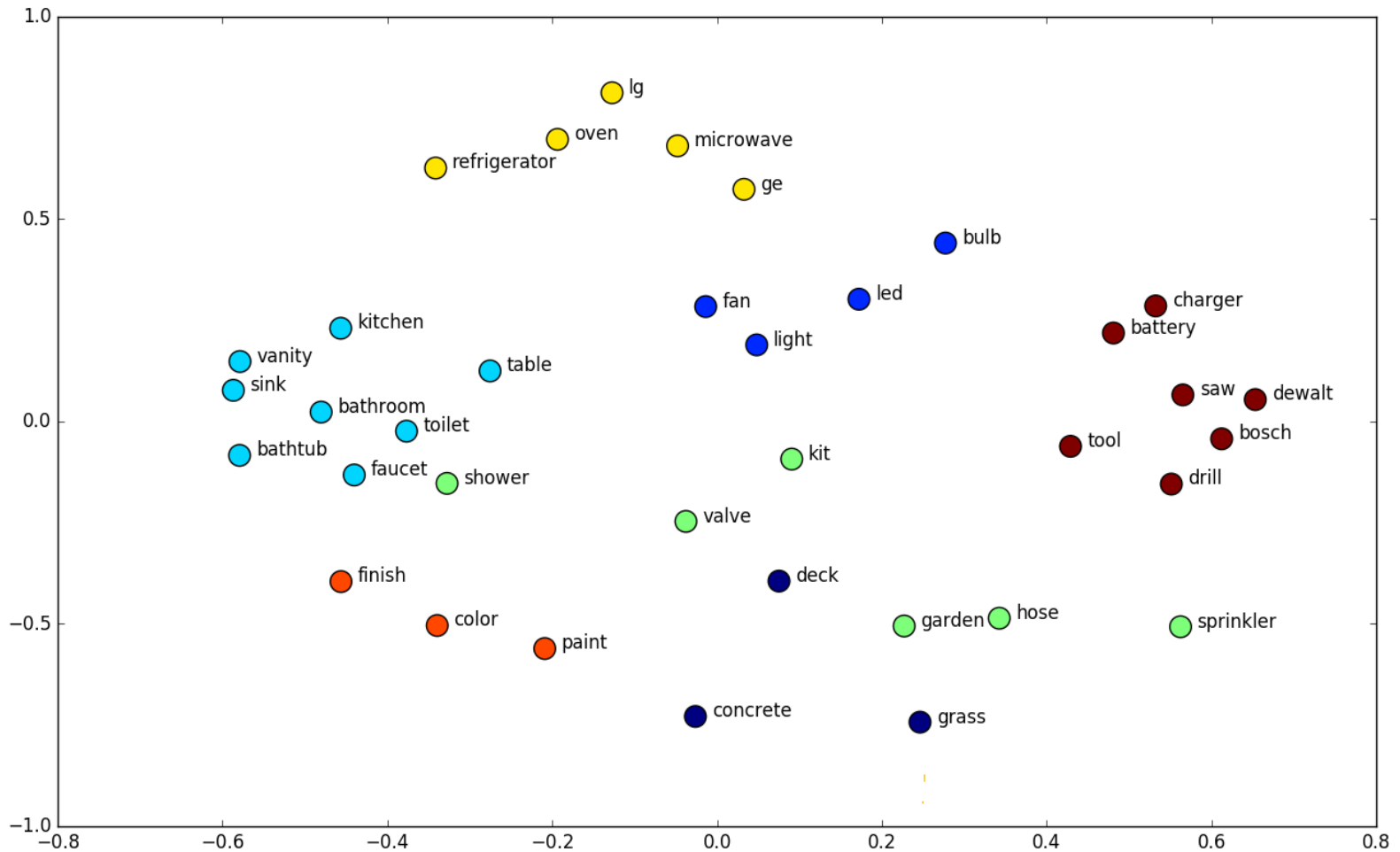


Στόχος: Διανυσματική *αναπαράσταση*
(representation) λέξεων – *κατανεμημένη (distributed)*
αναπαράσταση



Στόχος: παρόμοιες λέξεις -> παρόμοια διανύσματα

Παράδειγμα: 2-διάστατα embeddings

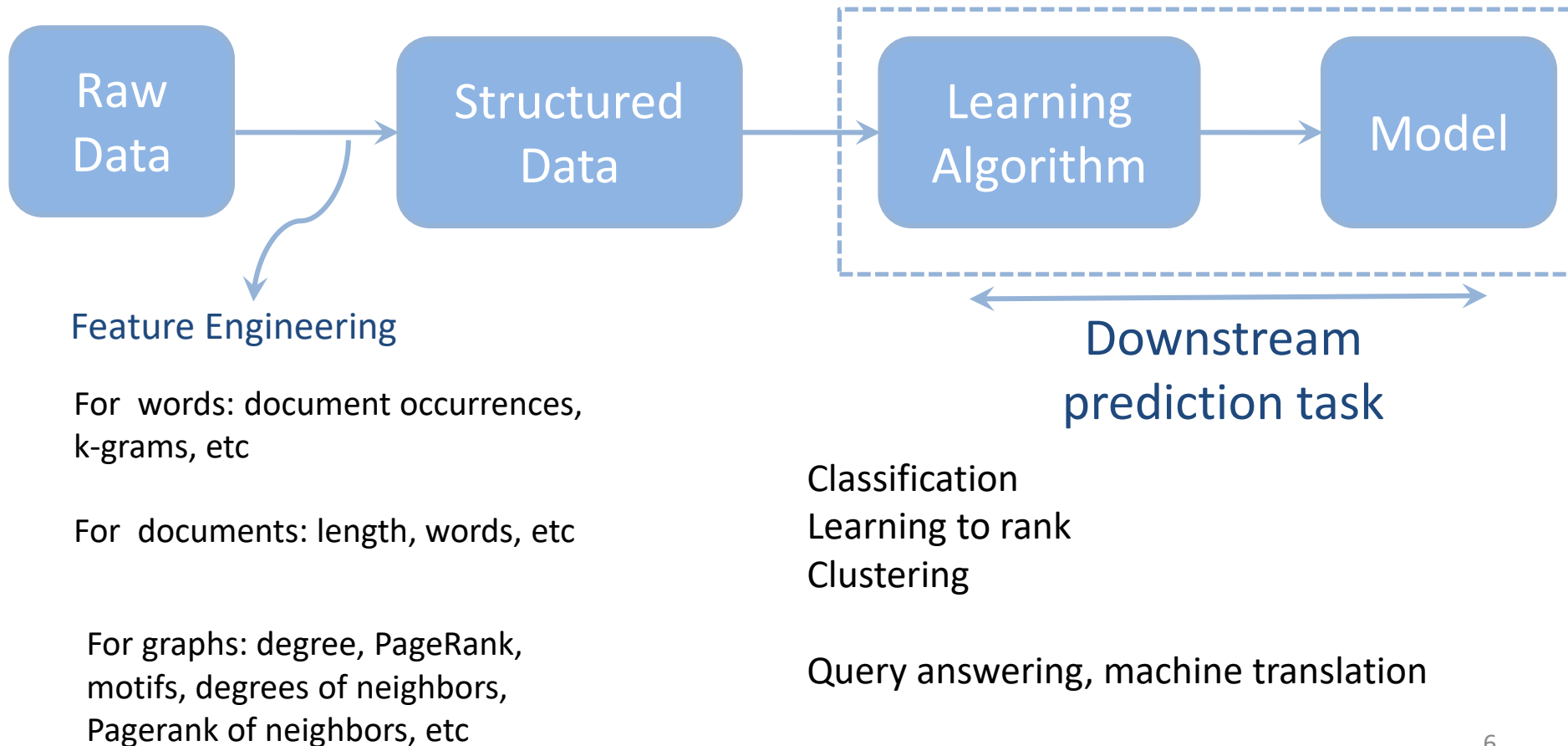


T-SNE plot

<http://suriyadeepan.github.io>

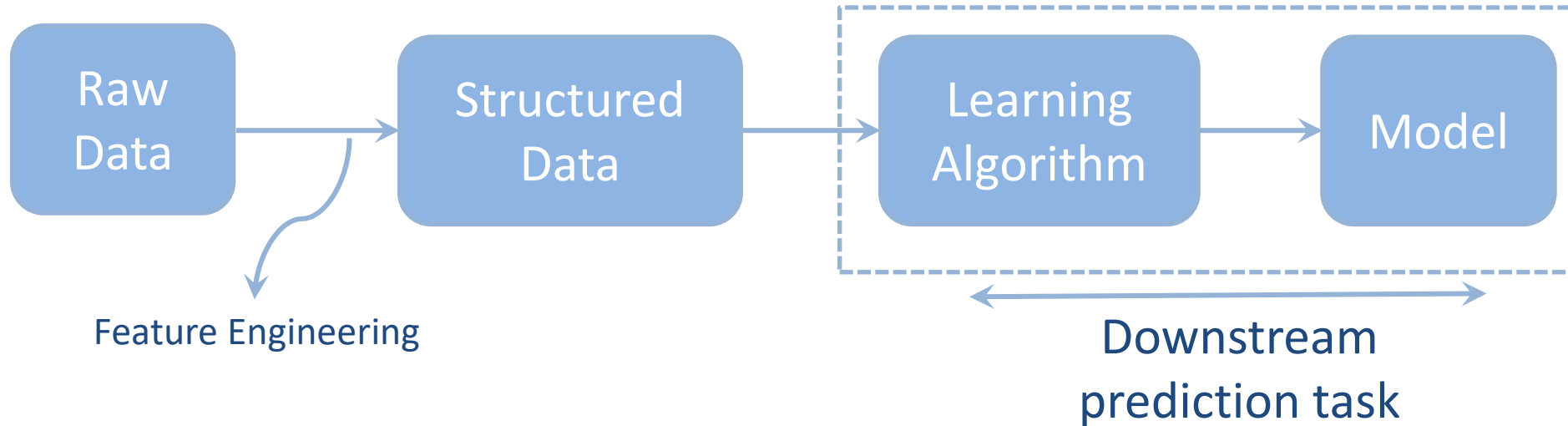
Embeddings: why?

Machine learning lifecycle



Embeddings: why?

Machine learning lifecycle



~~For words: document occurrences, k-grams, etc~~

~~For documents: length, words, etc~~

~~For graphs: degree, PageRank, motifs, degrees of neighbors, Pagerank of neighbors, etc~~

Automatically learn the features (embeddings)

Πως θα πάρουμε αυτά τα διανύσματα;

One-hot vectors

Έστω ότι υπάρχουν $|V|$ διαφορετικές λέξεις (όροι) στο λεξικό μας

- Διατάσσουμε τις λέξεις αλφαβητικά
- Αναπαριστούμε κάθε λέξη με ένα $\mathbb{R}^{|V| \times 1}$ διάνυσμα που έχει παντού 0 και μόνο ένα 1 στη θέση που αντιστοιχεί στη θέση της λέξης στη διάταξη

$$w^{aardvark} = \begin{bmatrix} \mathbf{1} \\ 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \quad w^a = \begin{bmatrix} 0 \\ \mathbf{1} \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \quad w^{at} = \begin{bmatrix} 1 \\ 0 \\ \mathbf{0} \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \quad \dots \quad w^{zerba} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{1} \end{bmatrix}$$

- Κάθε λέξη ένα διαφορετικό σύμβολο
- Πολλές διαστάσεις ($|V|$ όσες οι λέξεις)
- Καμία πληροφορία για ομοιότητα

Term-Document co-occurrence matrix

Έστω ότι υπάρχουν $|V|$ διαφορετικές λέξεις (όροι) στο λεξικό μας και $|M|$ έγγραφα

- Κατασκευάζουμε ένα $|V| \times M$ πίνακα με τις εμφανίσεις των λέξεων στα έγγραφα
- Αναπαριστούμε κάθε λέξη με ένα $R^{|M| \times 1}$

Παράδειγμα:

d1: a b c

d2: a d a b

d3: a c d e c a f

d4: b e a b

d5: a b d c a

$|V| = 6, |M| = 5$

	d1	d2	d3	d4	d5
a	1	1	1	1	1
b	1	1	0	1	1
c	1	0	1	0	1
d	0	1	1	0	1
e	0	0	1	1	0
f	0	0	1	0	0

Word vector
for c



Term-Document co-occurrence matrix

Μπορούμε αντί για 0-1 να έχουμε το tf ή και το tf-idf βάρος

- Πολλές διαστάσεις
- Πρόβλημα κλιμάκωσης με τον αριθμό των εγγράφων

Window-based co-occurrence matrix

- Κατασκευάζουμε ένα $|V| \times |V|$ **affinity-matrix** για τις λέξεις: για δύο λέξεις, μετράμε τον αριθμό των φορών που αυτές οι δύο λέξεις εμφανίζονται μαζί σε έγγραφα
- Συγκεκριμένα, μετράμε τον αριθμό των φορών που κάθε λέξη εμφανίζεται μέσα σε ένα **παράθυρο** συγκεκριμένου μεγέθους W γύρω από τη λέξη ενδιαφέροντος

Παράδειγμα:

d1: a b c

d2: a d a b

d3: a c d e c a f

d4: b e a b

d5: a b d c a

	a	b	c	d	e	f
a	0	4	3	1	1	1
b	4	0	1	1	1	0
c	3	1	0	2	1	0
d	1	1	2	0	1	0
e	1	1	1	1	0	0
f	1	0	0	0	0	0

$W = 1$ (σε απόσταση 1)

Window-based co-occurrence matrix

Λέξεις όπως *apple*, *orange*, *mango*, κλπ μαζί με λέξεις όπως *eat*, *grow*, *cultivate*, *slice*, κλπ και το ανάποδο

Παράδειγμα:

d1: I enjoy flying.

d2: I like NLP.

d3: I like deep learning.

$$W = 1$$

$$X = \begin{matrix} & \begin{matrix} I & like & enjoy & deep & learning & NLP & flying & . \end{matrix} \\ \begin{matrix} I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{matrix} & \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

- Πολλές διαστάσεις

Θα μπορούσαμε να χρησιμοποιήσουμε μια τεχνική για να μειώσουμε τις διαστάσεις (dimensionality reduction) (πχ PCA analysis)

Singular Value Decomposition

From dimension n to
dimension r

$$A = U \Sigma V^T = \begin{bmatrix} \vec{u}_1 & \vec{u}_2 & \cdots & \vec{u}_n \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix} \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vdots \\ \vec{v}_n \end{bmatrix}$$

$[n \times n] \qquad \qquad \qquad [n \times n] \qquad \qquad \qquad [n \times n]$

- $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n$: singular values (square roots of eigenvals $AA^T, A^T A$)
- $\vec{u}_1, \vec{u}_2, \dots, \vec{u}_n$: left singular vectors (eigenvectors of AA^T)
- $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_n$: right singular vectors (eigenvectors of $A^T A$)
- Cut the singular values at some index r (get the largest r such values)

Singular Value Decomposition

A_r best approximation of A
(Frobernius norm)

$$\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2}$$

$$A = U \Sigma V^T = \begin{bmatrix} \vec{u}_1 & \vec{u}_2 & \cdots & \vec{u}_r \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{bmatrix} \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vdots \\ \vec{v}_r \end{bmatrix}$$

$[n \times r]$ $[r \times r]$ $[r \times n]$

- r : rank of matrix A
- $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r$: singular values (square roots of eigenvals AA^T , $A^T A$)
- $\vec{u}_1, \vec{u}_2, \dots, \vec{u}_r$: left singular vectors (eigenvectors of AA^T) (first r columns)
- $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_r$: right singular vectors (eigenvectors of $A^T A$) (first r rows)

$$A_r = \sigma_1 \vec{u}_1 \vec{v}_1^T + \sigma_2 \vec{u}_2 \vec{v}_2^T + \cdots + \sigma_r \vec{u}_r \vec{v}_r^T$$

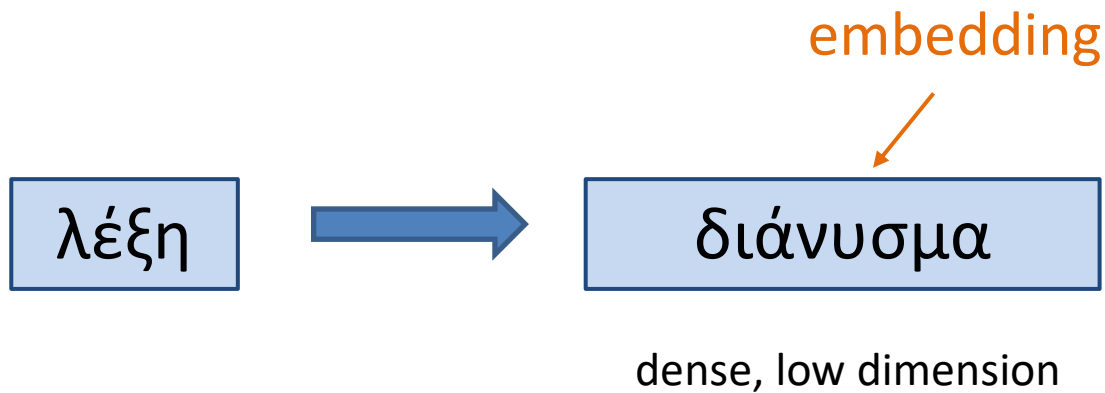
Αλλά

- Δύσκολο να ενημερώσουμε, πχ, αλλάζουν οι διαστάσεις συχνά
- Αραιός πίνακας
- Πολύ μεγάλες διαστάσεις

Θα δούμε μια τεχνική που βασίζεται σε επαναληπτικές μεθόδους

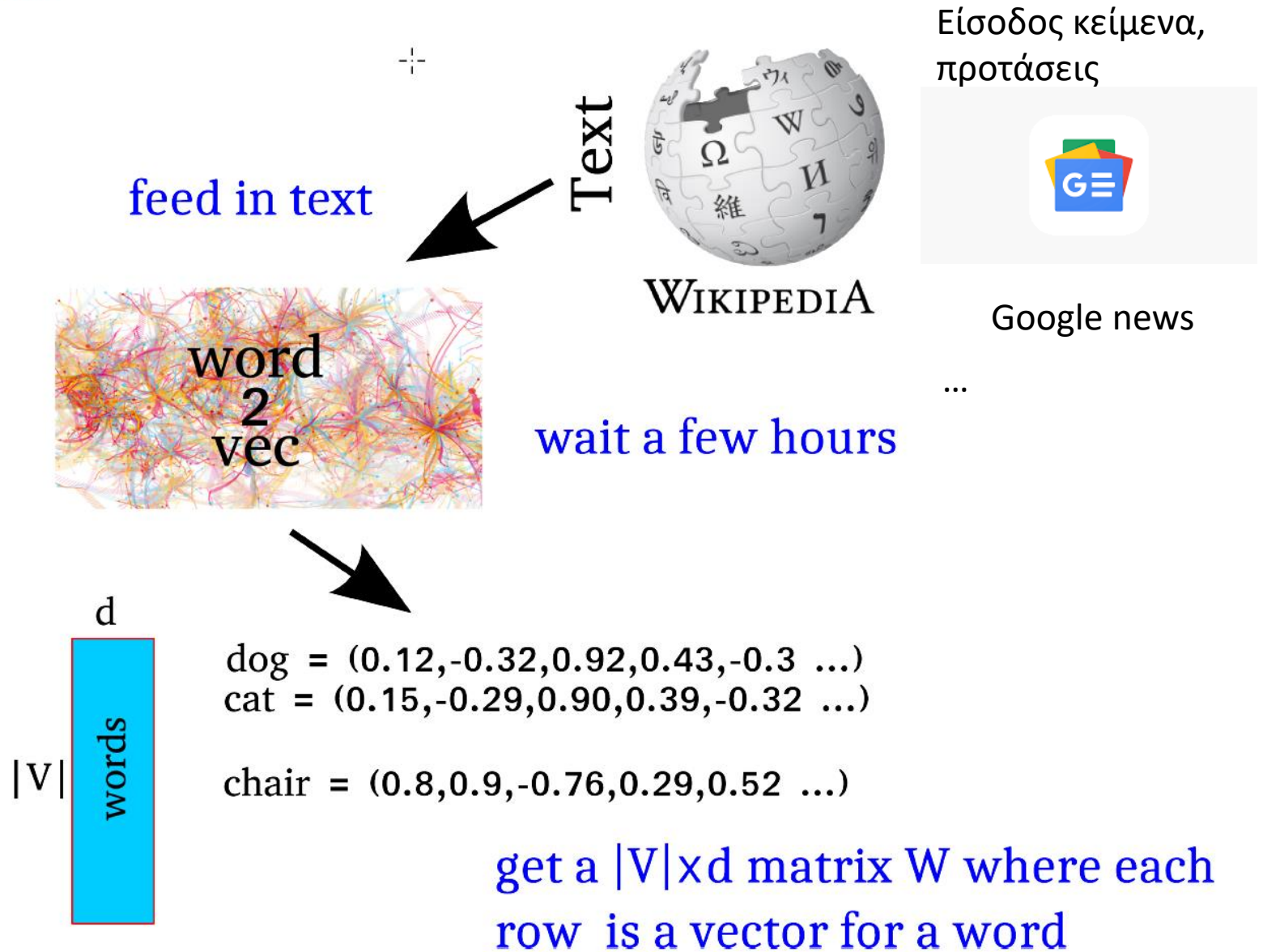
word2vec

Στόχος: Διανυσματική *αναπαράσταση*
(representation) λέξεων – *κατανεμημένη (distributed)*
αναπαράσταση



Στόχος: παρόμοιες λέξεις -> παρόμοια διανύσματα

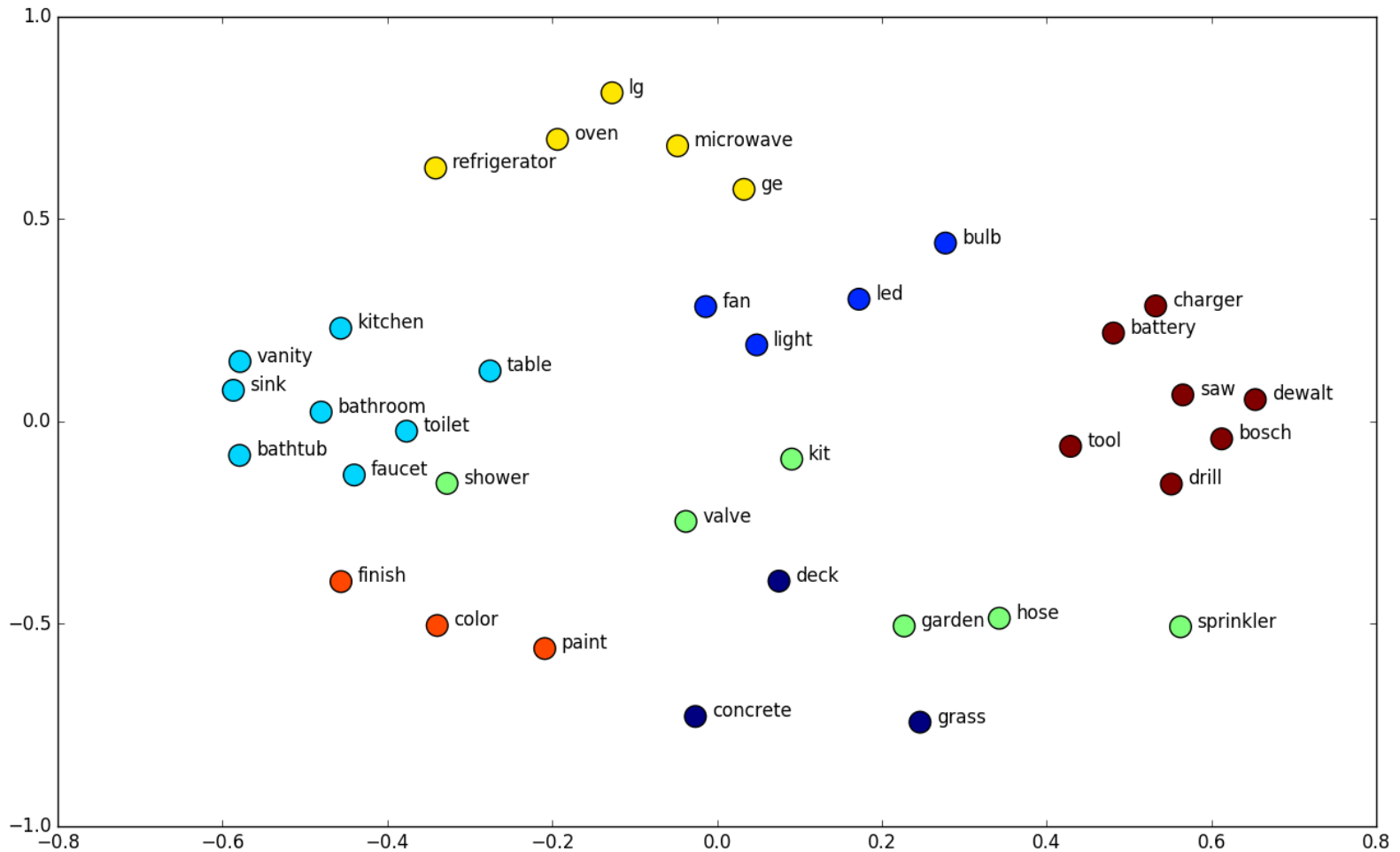
word2vec



word2vec

- dog
 - cat, dogs, dachshund, rabbit, puppy, poodle, rottweiler, mixed-breed, doberman, pig
- sheep
 - cattle, goats, cows, chickens, sheeps, hogs, donkeys, herds, shorthorn, livestock
- november
 - october, december, april, june, february, july, september, january, august, march
- jerusalem
 - tiberias, jaffa, haifa, israel, palestine, nablus, damascus katamon, ramla, safed
- teva
 - pfizer, schering-plough, novartis, astrazeneca, glaxosmithkline, sanofi-aventis, mylan, sanofi, genzyme, pharmacia

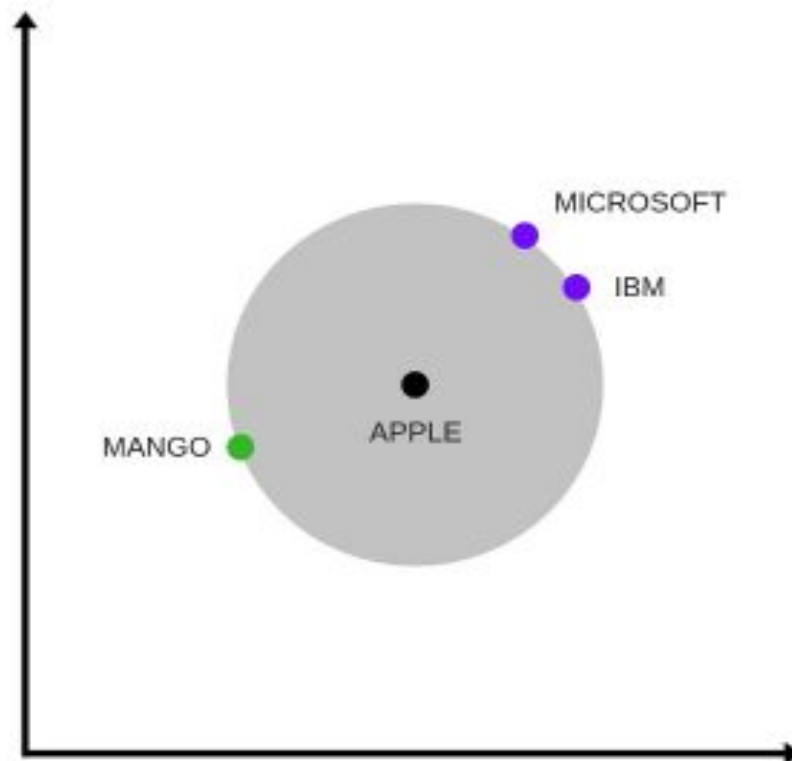
Παράδειγμα: 2-διάστατα embeddings



t-SNE plot

<http://suriyadeepan.github.io>

Apple: φρούτο και εταιρεία



<https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/>

Παράδειγμα: Έστω ότι έχουμε τον πίνακα W , πως θα βρούμε την πιο όμοια λέξη με το “dog”?

Σε προηγούμενα μαθήματα είδαμε

Lemmatization

Stemming

Λέξεις σημασιολογικά κοντινές

Ομοιότητα/απόσταση

1 2 3 4
5
6
||(1, 2, 3)||

- Similarity is calculated using *cosine similarity*:

$$\text{sim}(\vec{d}og, \vec{c}at) = \frac{\vec{d}og \cdot \vec{c}at}{\|\vec{d}og\| \|\vec{c}at\|}$$

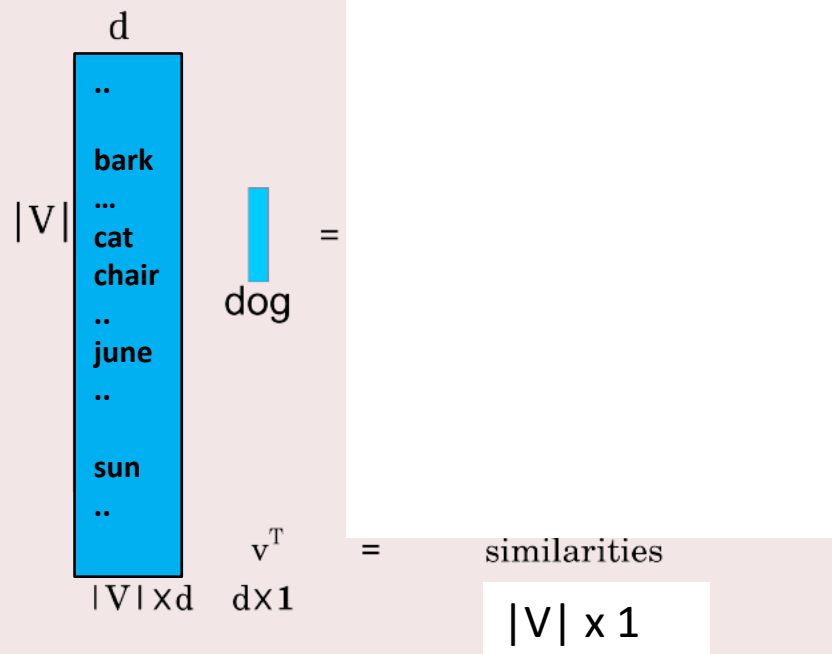
- For normalized vectors ($\|x\| = 1$), this is equivalent to a dot product:

$$\text{sim}(\vec{d}og, \vec{c}at) = \vec{d}og \cdot \vec{c}at$$

- **Normalize the vectors when loading them.**

TIP: Όπου μπορούμε χρησιμοποιούμε πράξεις πινάκων. Γιατί;

- Compute the similarity from word \vec{v} to all other words.
- This is a **single matrix-vector product**: $W \cdot \vec{v}^T$



- Result is a $|V|$ sized vector of similarities.
- Take the indices of the k -highest values.

Λέξη ποιο όμοια σε πολλές άλλες;

- “Find me words most similar to cat, dog and cow”.
- Calculate the pairwise similarities and sum them:

$$W \cdot \vec{cat} + W \cdot \vec{dog} + W \cdot \vec{cow}$$

- Now find the indices of the highest values as before.
- Matrix-vector products are wasteful. **Better option:**

$$W \cdot (\vec{cat} + \vec{dog} + \vec{cow})$$

Πως θα πάρουμε αυτόν τον πίνακα;

Basic Idea

- You can get a lot of value by representing a word by means of its neighbors (distributional semantics)
- “You shall know a word by the company it keeps”



(J. R. Firth 1957: 11)

- One of the most successful ideas of modern statistical NLP
- Context: words that appear in a fixed length window around the word

government debt problems turning into banking crises as has happened in
saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Use the many contexts of *w* to represent a word

Βασική ιδέα

Μία λέξη προσδιορίζεται από τις συμφραζόμενες της λέξεις (context)



Ο καθηγητής διδάσκει το **μάθημα** στους φοιτητές του στην αίθουσα.



Παράθυρο (window) = 3

- Center word Κάθε λέξη **δύο** αναπαραστάσεις: (1) center (2) context
- Context word Δηλαδή, έχουμε $2 |V| \times d$ πίνακες

- Το *center*-διάνυσμα της center λέξης πρέπει να είναι *όμοιο* με τα *context*-διανύσματα (δηλαδή, το άθροισμα των context διανυσμάτων) των context λέξεων
- Και προφανώς το *συμμετρικό*

Βασική ιδέα

Κάθε λέξη δύο διανυσματικές αναπαραστάσεις:

(1) center (2) context

Δηλαδή, έχουμε $2 |V| \times d$ πίνακες

- Το *center-διάνυσμα* της λέξης πρέπει να είναι *όμοιο* με τα *context-διανύσματα* (δηλαδή, το άθροισμα των context διανυσμάτων) των λέξεων που εμφανίζονται ως context της
- Το *context-διάνυσμα* της λέξης πρέπει να είναι *όμοιο* με τα *center-διανύσματα* των λέξεων που εμφανίζονται ως κεντρικές της

Learning: παραδείγματα κειμένου και προσπαθούμε να «μάθουμε» αυτά τα διανύσματα (βάρη), δηλαδή τους 2 πίνακες

Word2Vec

Predict between every word and its context words

Two algorithms

1. Skip-grams (SG)

Predict context words given the center word

2. Continuous Bag of Words (CBOW)

Predict center word from a bag-of-words context

Position independent (do not account for distance from center)

Two training methods

1. Hierarchical softmax
2. Negative sampling

Βασική ιδέα

Μία λέξη προσδιορίζεται από τις συμφραζόμενες της λέξεις (context)



Ο καθηγητής διδάσκει το **μάθημα** στους φοιτητές του στην αίθουσα.



Παράθυρο (window) = 3

καθηγητής διδάσκει το _____ στους φοιτητές του

_____ μάθημα _____

Συμβολισμοί

One-hot vectors

Έστω ότι υπάρχουν $|V|$ διαφορετικές λέξεις (όροι) στο λεξικό μας

- Διατάσσουμε τις λέξεις αλφαβητικά
- Αναπαριστούμε κάθε λέξη με ένα $\mathbb{R}^{|V| \times 1}$ διάνυσμα που έχει παντού 0 και μόνο έναν 1 στη θέση που αντιστοιχεί στη θέση της λέξης στη διάταξη
- Aka indicator vector

$$w^{aardvark} = \begin{bmatrix} \mathbf{1} \\ 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \quad w^a = \begin{bmatrix} 0 \\ \mathbf{1} \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \quad w^{at} = \begin{bmatrix} 1 \\ 0 \\ \mathbf{0} \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \quad \dots \quad w^{zerba} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{1} \end{bmatrix}$$

Look up

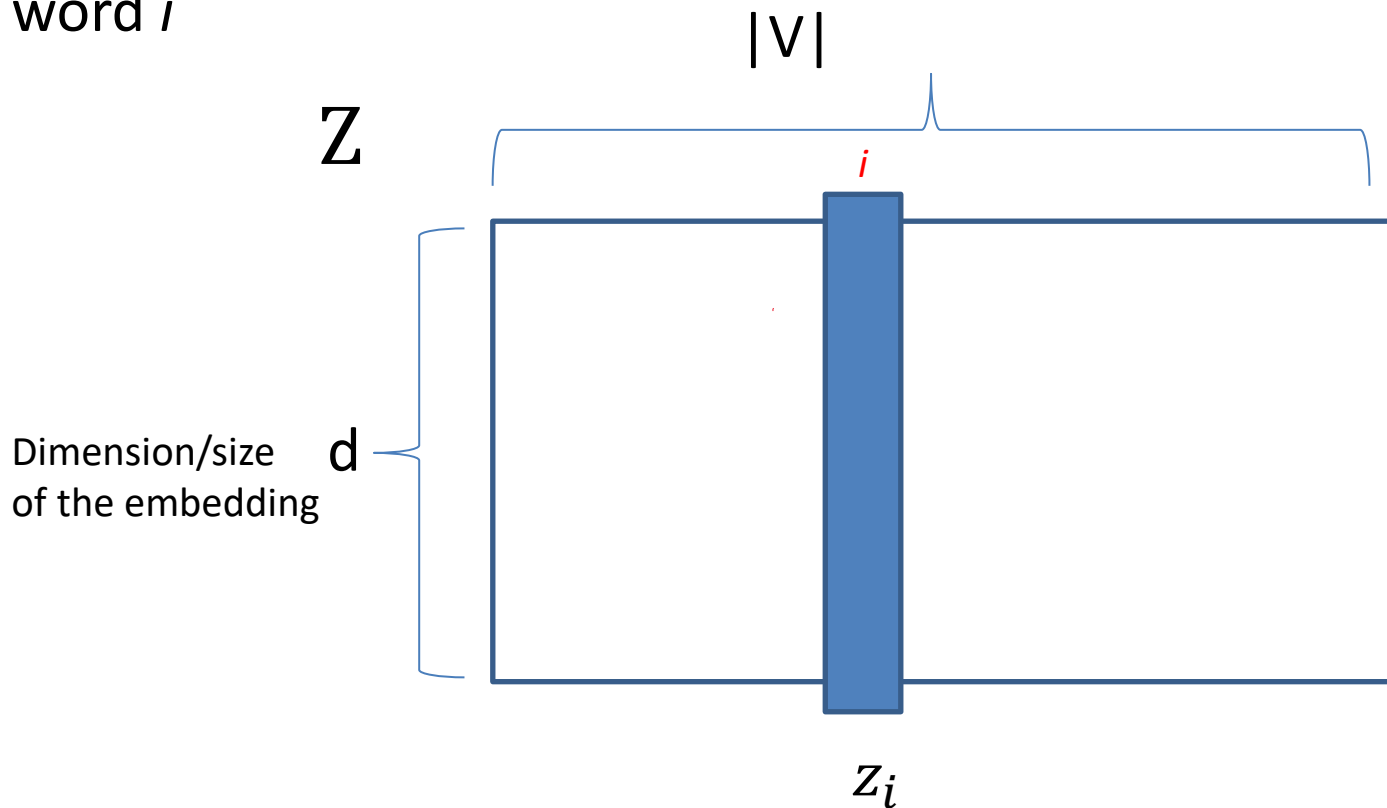
$|V|$ number of words

d size of embedding

m size of the window (context)

Each word is assigned *a d -dimensional vector*

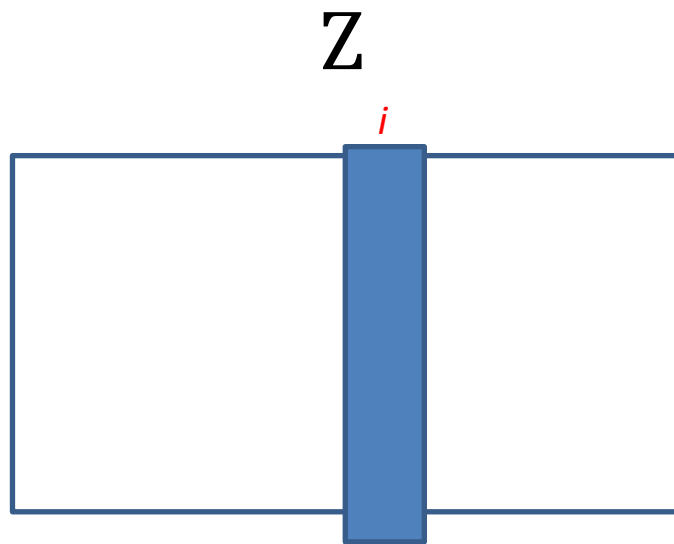
Learn embedding matrix Z : each column i is the embedding z_i of word i



Look up

Encoder is an embedding lookup

$$ENC(i) = Z I_i$$



Z_i
Embedding matrix

$d \times |V|$

One hot vector I_i

$$\begin{bmatrix} 0 \\ \cdot \\ \mathbf{1} \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix}$$

One-hot or indicator
vector, all 0s but
position i

$|V| \times 1$

=

$$\begin{bmatrix} w_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ w_d \end{bmatrix}$$

Embedding of
the i -th word

$d \times 1$

CBOW

CBOW

$|V|$ number of words

d size of embedding

m size of the window (context)

Use a window of context words to predict the center word

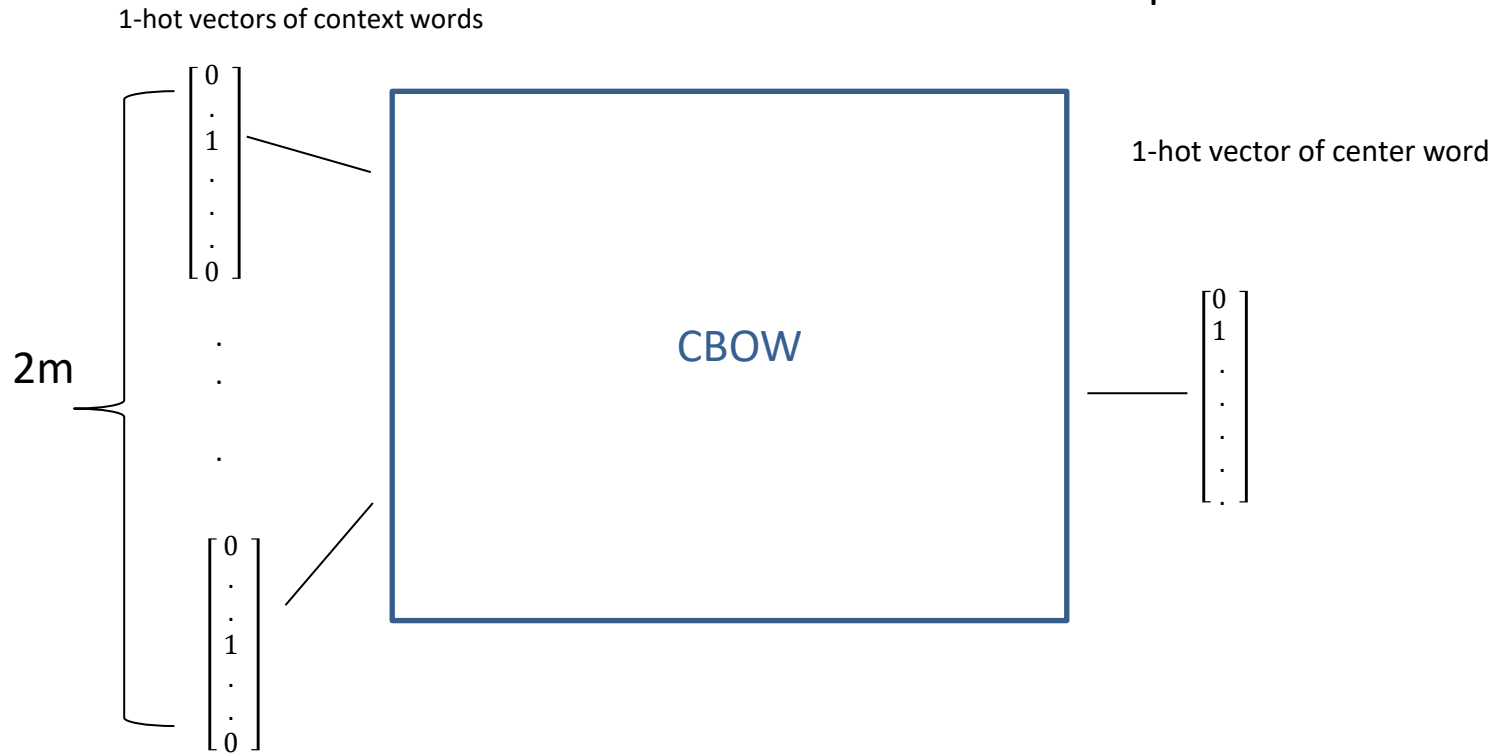
Input: $2m$ context words

Output: center word

each represented as a one-hot vector

Input

Output



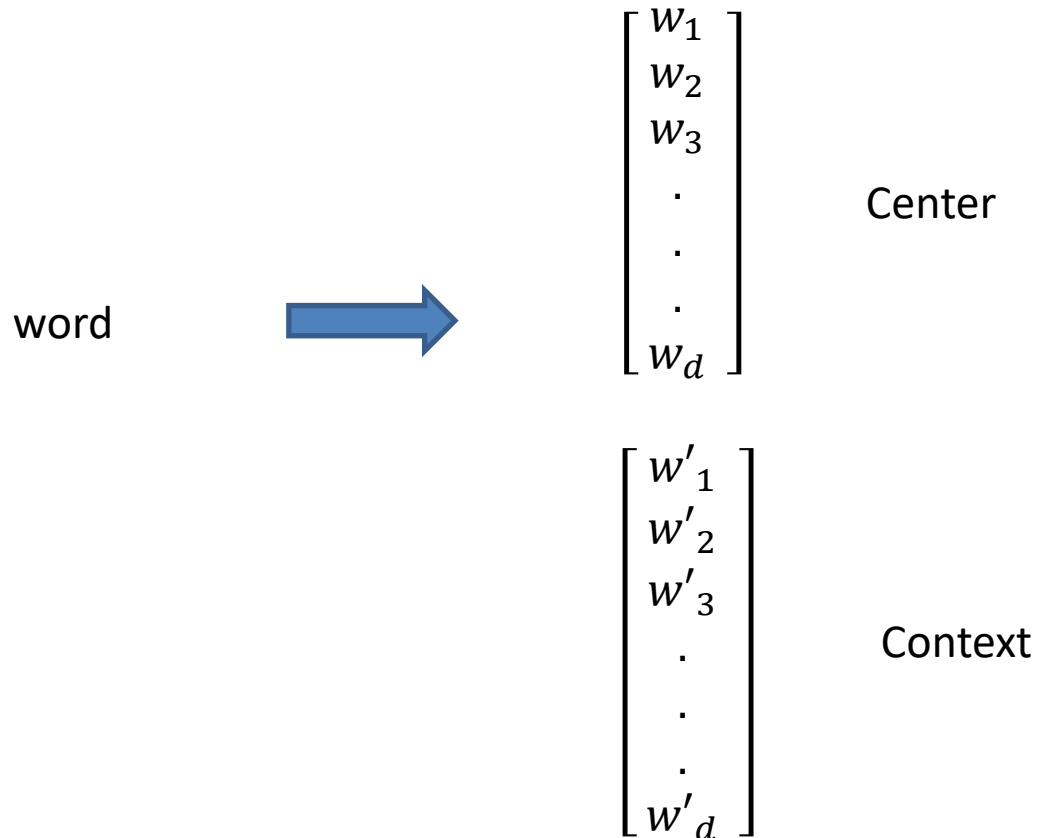
Εκπαίδευση (learning)

Δίνουμε *δεδομένα εκπαίδευσης*: (κείμενο) δηλαδή συμφραζόμενες λέξεις για τις οποίες ξέρουμε την κεντρική

και μαθαίνουμε αυτά τα διανύσματα (βάρη) (δηλαδή, τους 2 πίνακες) ώστε να *ελαχιστοποιούν το «λάθος»* στα δεδομένα εκπαίδευσης, δηλαδή να «βρίσκουν» την κεντρική

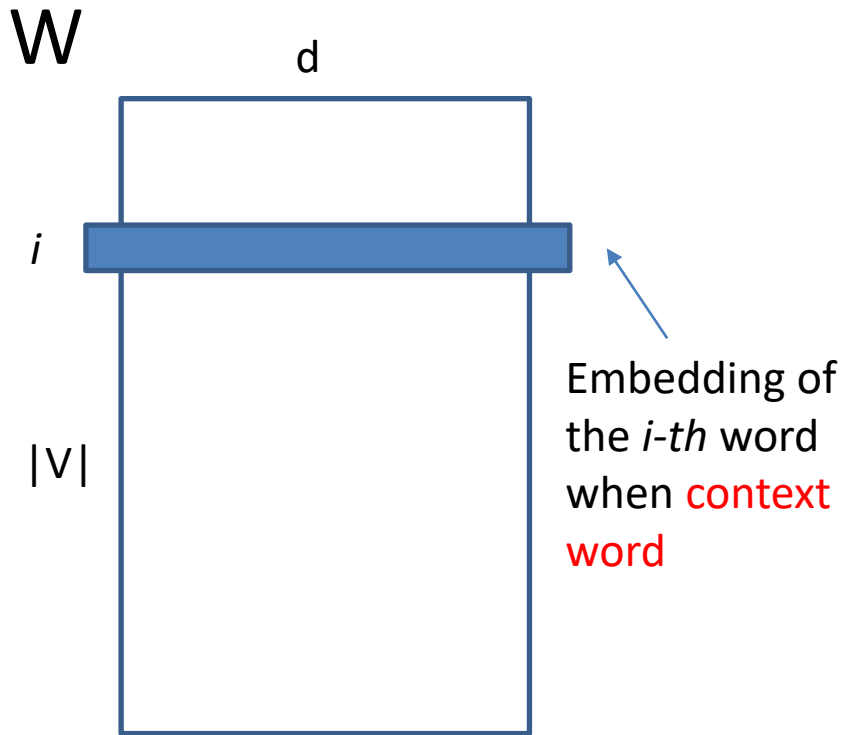
CBOW

Learns **two matrices** (two embeddings per word, one when context, one when center)

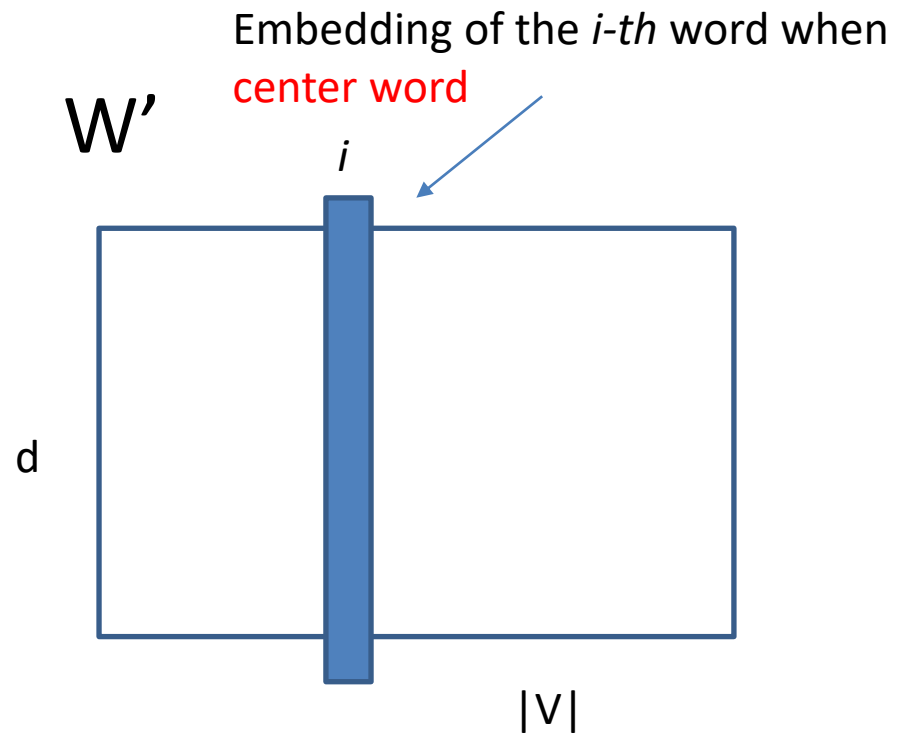


CBOW

Learns **two matrices** (two embeddings per word, one when context, one when center)



$|V| \times d$ context embeddings
when input



$d \times |V|$ center embeddings
when output

CBOW

Intuition

The W' -embedding of the *center word* should be *similar* to the (average of) the W -embeddings of its *context words*

- For similarity, we will use cosine (**dot product**)
- We will take the **average** of the W -embeddings of the context word

We want similarity close to one for the center word and close to 0 for all other words

CBOW

Given **window size** m

$x^{(c)}$ one hot vector for context words, y one hot vector for the center word

1. **INPUT:** the *one hot vectors* for the $2m$ context words

$$x^{(c-m)}, \dots, x^{(c-1)}, x^{(c+1)}, \dots, x^{(c+m)}$$

2. **GET THE EMBEDDINGS** of the context words

$$v_{c-m} = Wx^{(c-m)}, \dots, v_{c-1} = Wx^{(c-1)}, v_{c+1} = Wx^{(c+1)}, \dots, v_{c+m} = Wx^{(c+m)}$$

3. **TAKE THE SUM** these vectors (average)

$$\hat{v} = \frac{v_{c-m} + v_{c-m+1} + \dots + v_{c+m}}{2m}, \hat{v} \in R^N$$

4. **COMPUTE SIMILARITY:** dot produce W' (all center vectors) and context \hat{v} (generate score vector z)

$$z = W' \hat{v}$$

5. Turn the *score vector to probabilities*

$$\hat{y} = \text{softmax}(z)$$

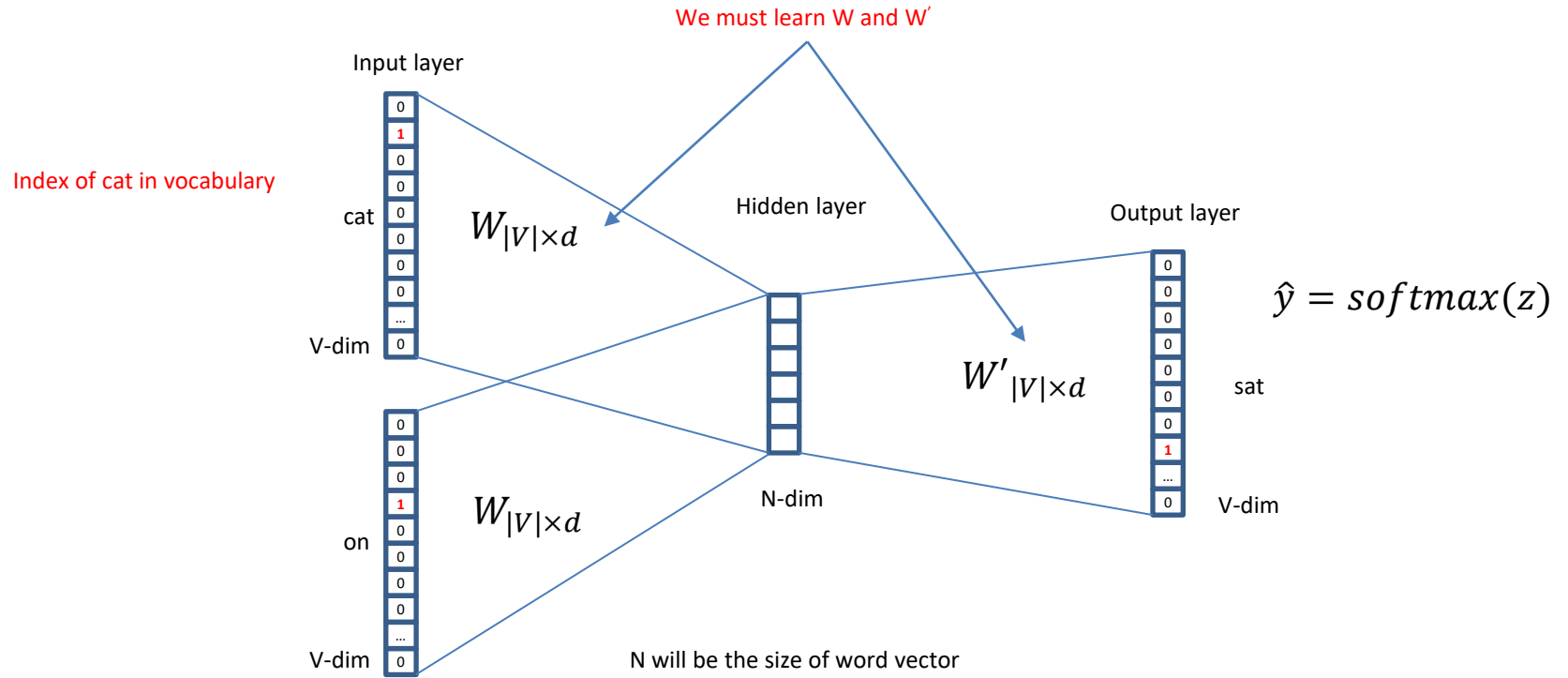
We want this to be close to 1 for the center word

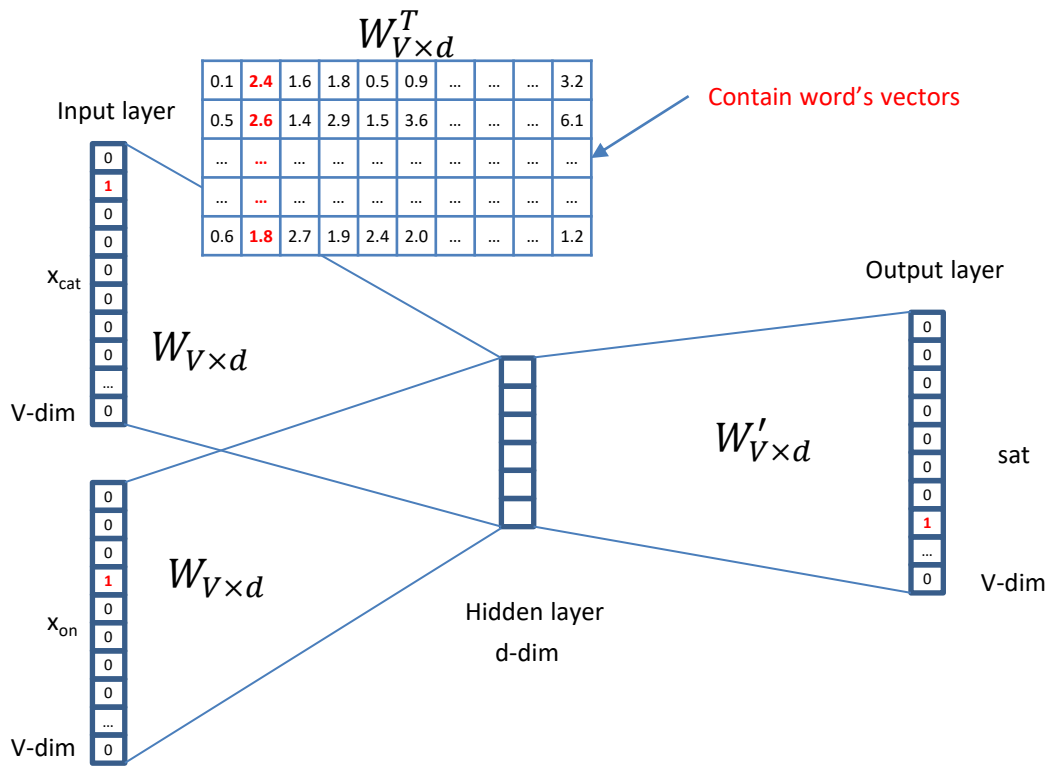
Απεικόνιση κατανομών οποιοδήποτε τιμών σε κατανομές πιθανοτήτων

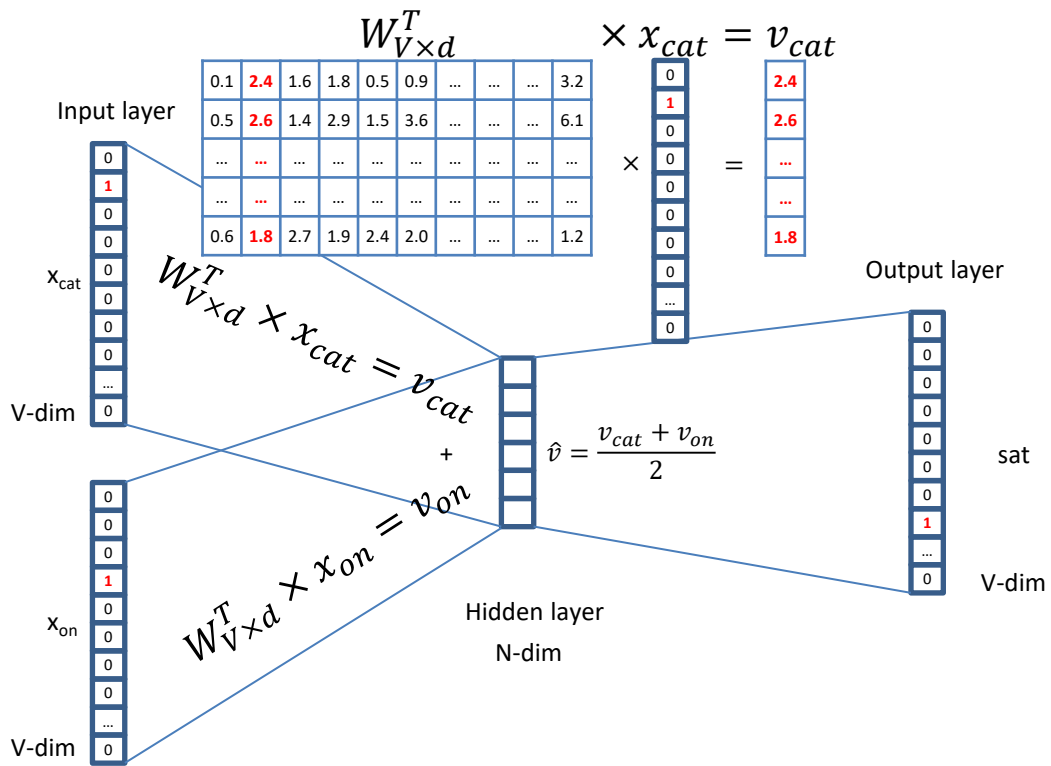
$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

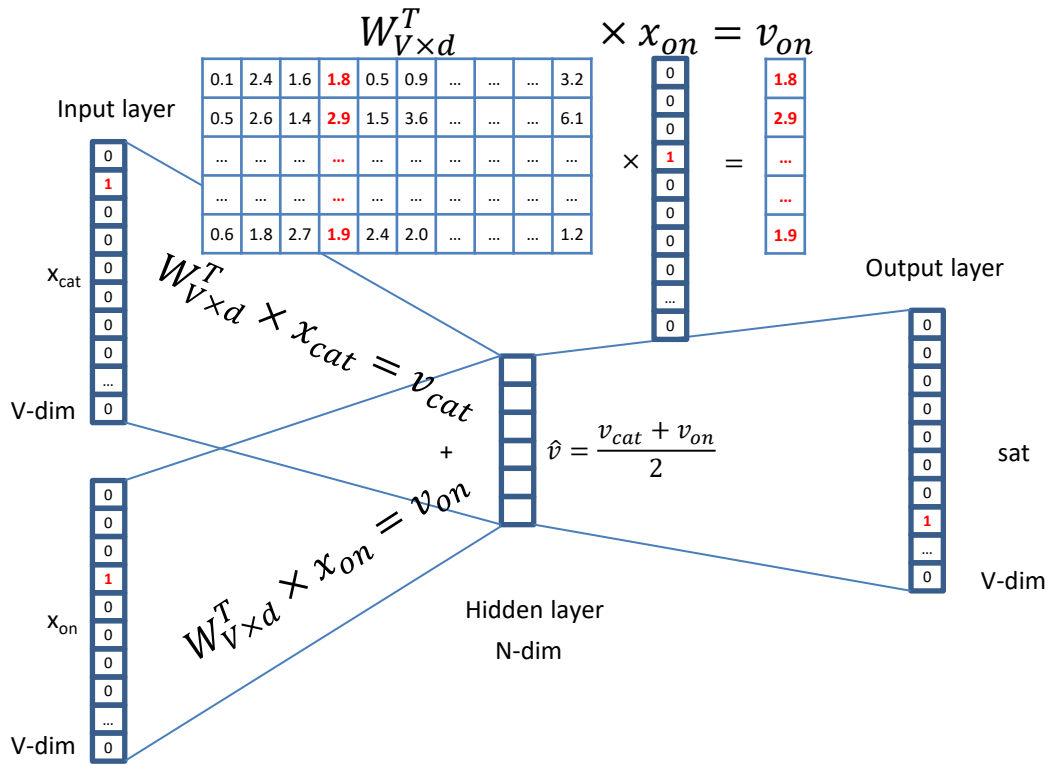
- Εκθετικό για να είναι θετικό
- Ενισχύει την πιθανότητα της μεγαλύτερης τιμής (max)
- Κάποια πιθανότητα και στις μικρότερες τιμές (soft)

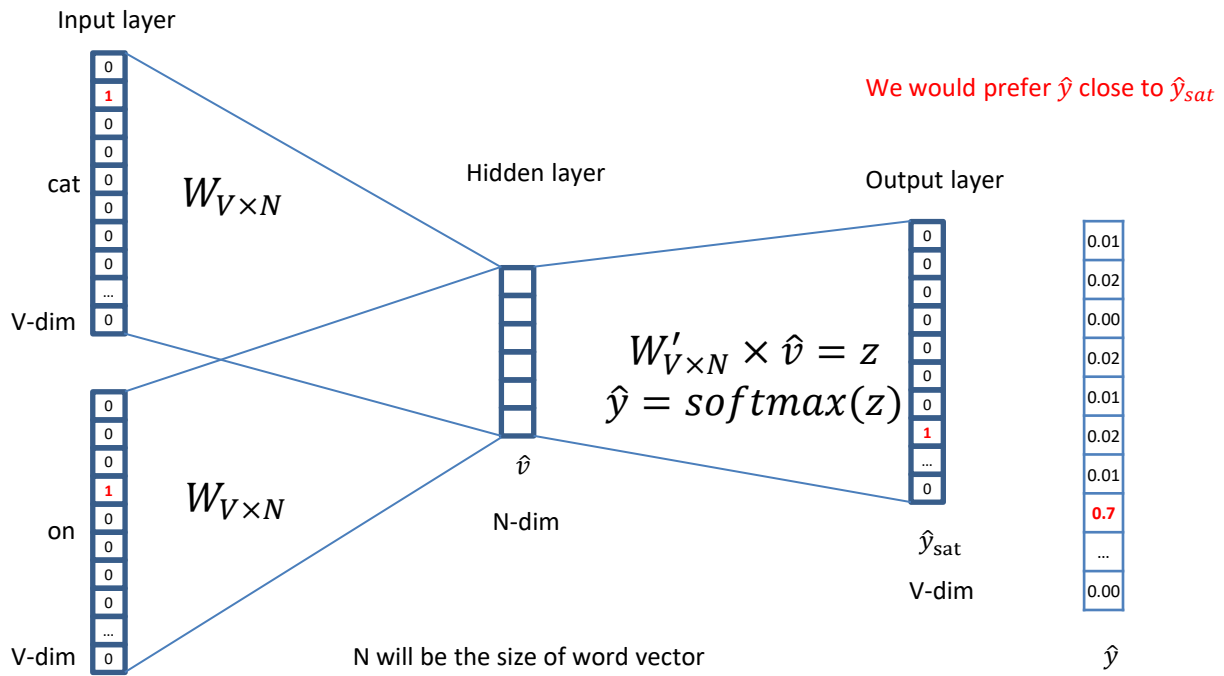
cat sat on window size = 1











- We can consider either W (context) or W' (center) as the word's representation.
 - Or even take the average.

- Use the model to predict a missing word

SKIPGRAM

Skipgram

Given the center word, predict (or, generate) the context words

Input: center word

Output: 2m context word

each represented as a one-hot vectors

Learn two matrices

W : $d \times |V|$, input matrix, word representation as center word

W' : $|V| \times d$, output matrix, word representation as context word

Input

1-hot vector of center word

$$\begin{bmatrix} 0 \\ 1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

—



Output

1-hot vectors of context words

$$\begin{bmatrix} 0 \\ \cdot \\ 1 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix}$$

·

·

·

$$\begin{bmatrix} 0 \\ \cdot \\ \cdot \\ 1 \\ \cdot \\ \cdot \\ 0 \end{bmatrix}$$

} 2m

Εκπαίδευση (learning)

Δίνουμε δεδομένα εκπαίδευσης (κείμενο, δηλαδή κεντρικές λέξεις για τις οποίες ξέρουμε τις συμφραζόμενες)

και μαθαίνουμε αυτά τα διανύσματα (βάρη) (δηλαδή, τους 2 πίνακες) ώστε να ελαχιστοποιούν το «λάθος» στα δεδομένα εκπαίδευσης, δηλαδή να «βρίσκουν» τις συμφραζόμενες

Skipgram

Given the center word, predict (or, generate) the context words

$y^{(j)}$ one hot vector for context words

1. Input: *one hot vector* of the center word

x

2. Get the *embedding of the center word*

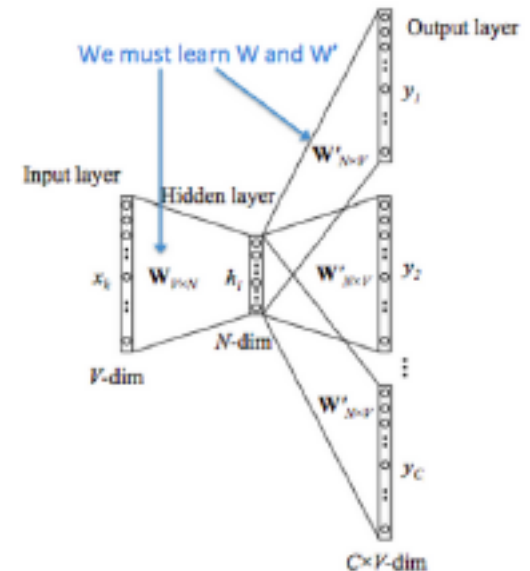
$$v_c = W x$$

3. Generate a *score vector for each context word*

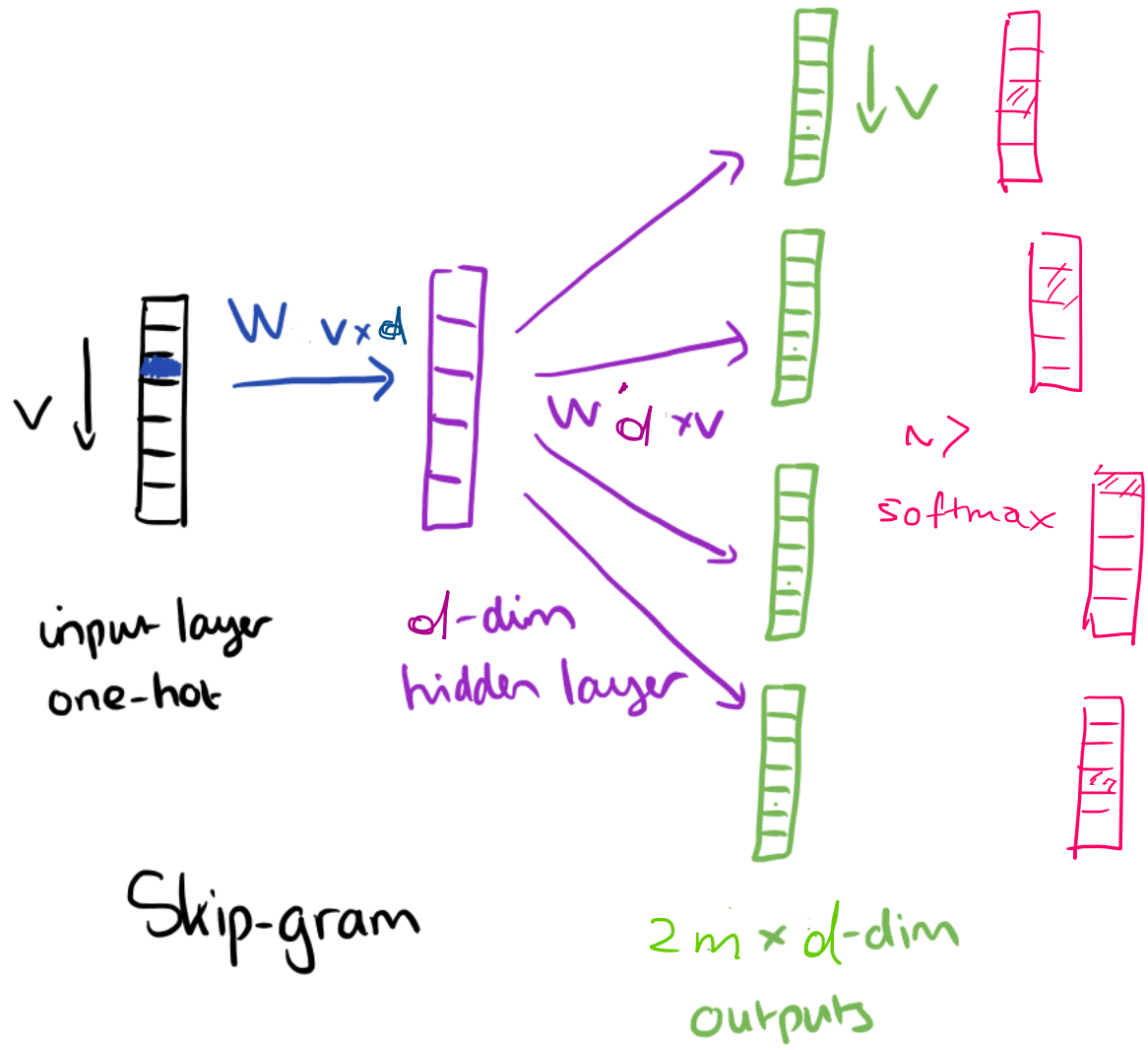
$$z = W' v_c$$

5. Turn the *score vector into probabilities*

$$\hat{y} = \text{softmax}(z)$$

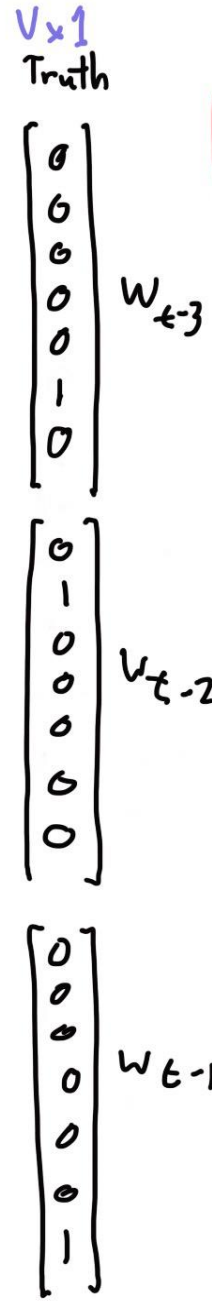
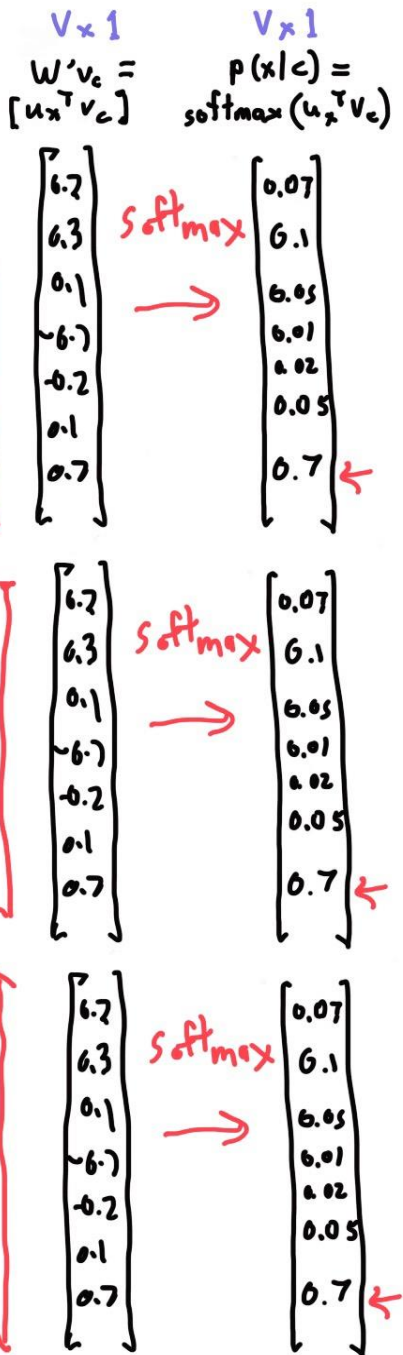
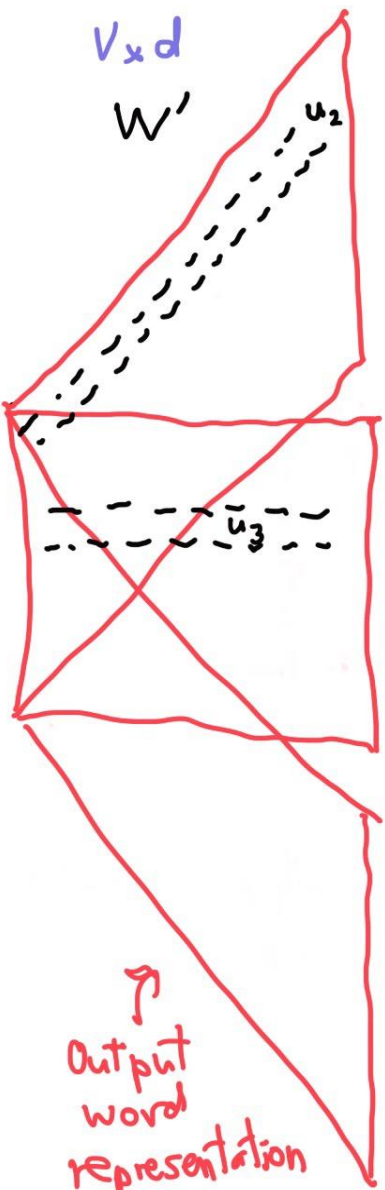
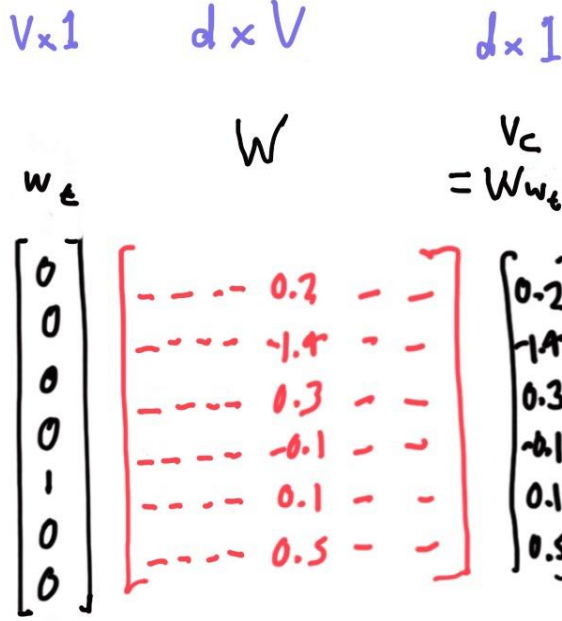


We want this to be close to 1 for the context words



Skip-gram

Skipgram



softmax

$$p_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Actual context words

↑
 one hot word symbol
 ↑
 word

↑
 Looks up column of word embedding matrix as representation of center word

↘
 Output word representation

Αποτελέσματα

These representations are *very good* at encoding **similarity** and **dimensions of similarity**!

- Analogies

A is to B what C is to D

Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

Syntactically

- $X_{apple} - X_{apples} \approx X_{car} - X_{cars} \approx X_{family} - X_{families}$
- Similarly for verb and adjective morphological forms

Semantically

- $X_{shirt} - X_{clothing} \approx X_{chair} - X_{furniture}$
- $X_{king} - X_{man} \approx X_{queen} - X_{woman}$

Test for linear relationships, examined by Mikolov et al.

a:b :: c:?



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

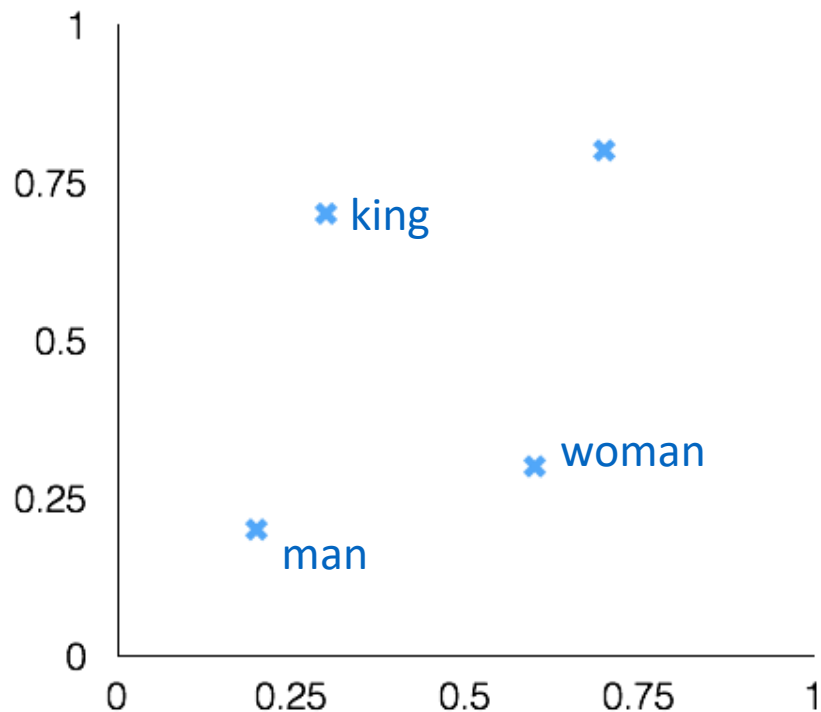
man:woman :: king:?

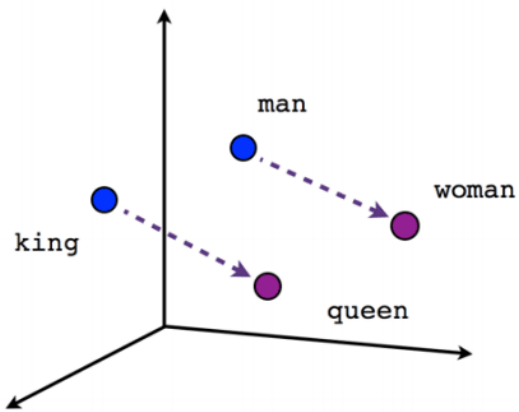
+ king [0.30 0.70]

- man [0.20 0.20]

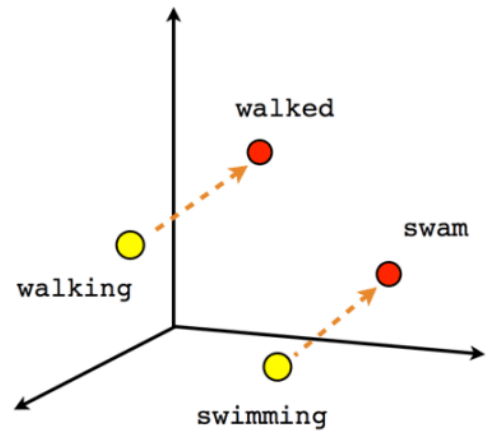
+ woman [0.60 0.30]

queen [0.70 0.80]

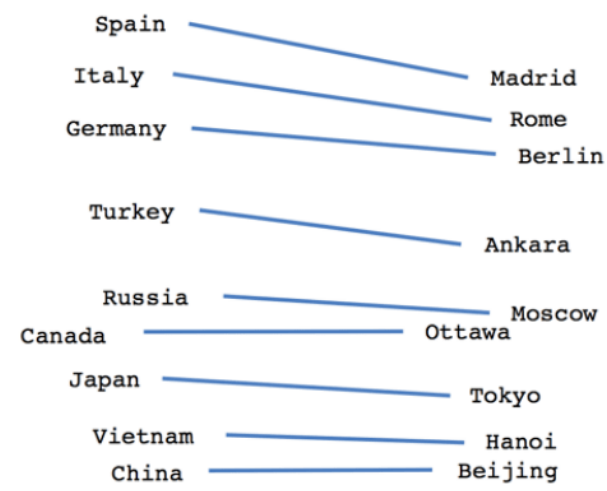




Male-Female



Verb tense



Country-Capital

Κάποιες εφαρμογές

Global vs. local embedding

global	local	
cutting	tax	Πάνω σε ποια συλλογή (corpus) φτιάχνουμε τα embeddings; Προτάσεις από ποια κείμενα θα χρησιμοποιήσουμε;
squeeze	deficit	
reduce	vote	
slash	budget	
reduction	reduction	
spend	house	
lower	bill	
halve	plan	
soften	spend	
freeze	billion	

[Diaz 2016] Terms similar to ‘cut’ for a word2vec model trained on a general news corpus and another trained only on documents related to ‘gasoline tax’.

1. Train and create embeddings based on a local collection

Python implementation in gensim

<https://radimrehurek.com/gensim/models/word2vec.html>

Tensorflow

https://www.tensorflow.org/tutorials/text/word_embeddings

2. Use pretrained embeddings

Pretrained embeddings for 157 languages

<https://fasttext.cc/docs/en/crawl-vectors.html>

Google

<https://code.google.com/archive/p/word2vec/>

Finding the degree of similarity between two words.

```
model.similarity('woman','man')  
0.73723527
```

Finding odd one out.

```
model.doesnt_match('breakfast cereal dinner lunch'.split())  
'cereal'
```

Amazing things like woman+king-man =queen

```
model.most_similar(positive=['woman','king'],negative=['man'],top  
n=1)  
queen: 0.508
```

Probability of a text under the model

```
model.score(['The fox jumped over the lazy dog'.split()])  
0.21
```

ανεκτική ανάκτηση: (1) επέκταση ερωτήματος ή/και (2)
context-dependent διόρθωση λάθους, όπου θα μπορούσαμε να
χρησιμοποιήσουμε και το query log και γενικά query
suggestions

Improve language translation



bilingual embedding with chinese in green and english in yellow

By aligning the word embeddings for the two languages

Χρήση στη διάταξη των εγγράφων του αποτελέσματος μιας ερώτησης

Είδαμε διάταξη με $q^T d$

Μπορούμε να χρησιμοποιήσουμε *embeddings*;

Πολλές εναλλακτικές, για παράδειγμα (aboutness)

$$\sum_{w \in q} wd'$$

Όπου w το *embedding των λέξεων της ερώτησης* και d' το *embedding του εγγράφου* (π.χ., το μέσο των *embedding των λέξεων του εγγράφου*)

- Στόχος: Σχέση ερώτησης με το όλο το περιεχόμενο του εγγράφου
- Input (center word) embedding ή output (context) word embedding; in-query, out-document
- Σε συνδυασμό με άλλα κριτήρια

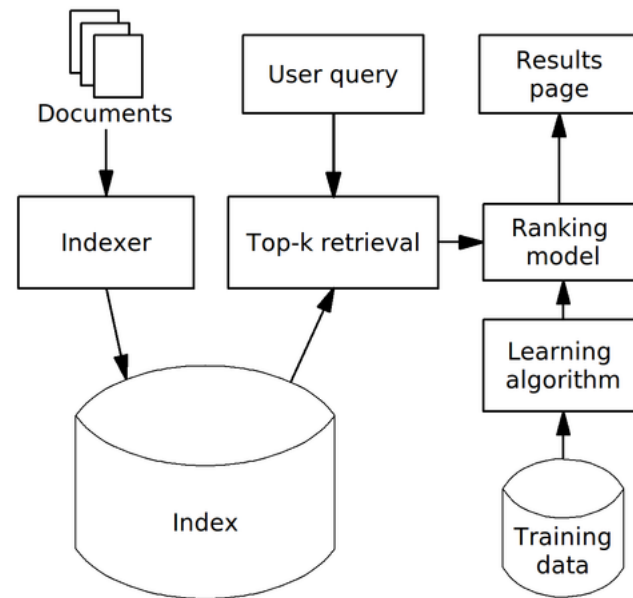
Learning to rank

Learning to rank

- Input: Documents, query
- Learn some (partial) order of the documents based on the relevance of each document for the query
 - Document features (tf, embeddings, k-grams, etc)
 - Query features
 - *How do we specify the ranking? (next)*
- Learn a *ranking function* f (a ranking model)
 - by solving a minimization problem with respect to a *loss function* which is a measure of *accuracy with respect to the training data*

Learning to rank in IR

- Get the top- k documents (e.g., document that have at least one of the query terms)
- Use the ranking models to create the final ranking the top- k documents



Learning to rank

1. Point-wise approach

- Training data
 - For each query-document pair (q, d) there is a numerical, or ordinal score
- The learning-to-rank problem can be approximated by *a regression problem*:
 - Given a (q, d) pair predicts its score
- Existing supervised machine learning algorithms can be used for this purpose.

Learning to rank

2. Pair-wise approach

- Training data
 - For each query q ,
 - a pair of documents (d_1, d_2) , where d_1 is better than d_2
- The learning-to-rank problem can be seen as *a binary classification problem*:
 - Given a (d_1, d_2) , pair output 1 if d_1 is better and 0 otherwise
- Existing supervised machine learning algorithms such as probabilistic classifiers (e.g., logistic regression) can be used for this purpose.

Learning to rank

3. List-wise approach

- Training data not a single, or a couple of documents but an ordered list.
For example:
For each query q ,
a list of documents d_i associated with scores/judgements r_i

- Learning-to-rank

$$L_{LN}(r(q), \hat{r}(q))$$

minimize the sum of the differences between the predicted $\hat{r}(q)$ and the actual score $r(q)$.

Transformers

Transformers

- A class of *deep learning* models
- An instance of the *decoder-encoder* model
- First introduced in the “*Attention is All you Need*” paper by Google researchers in 2017
- Original designed for *language translation*, particularly from English to German. But, generalized to other language tasks
- Many *applications*: example, question answering, sentiment analysis, entity recognition

Encoder/decoder architecture

Two models

- **Encoder**: maps an *input sequence* to a sequence of continuous *representations*, which is then fed into a decoder.
- **Decoder**: receives the *output of the encoder* together with the *decoder output at the previous time step*, to generate an *output sequence*

Jointly trained to minimize the conditional log-likelihood.

Once trained, the encoder/decoder can

- generate an output given an input sequence, or
- can score a pair of input/output sequences.

Encoder/decoder architecture

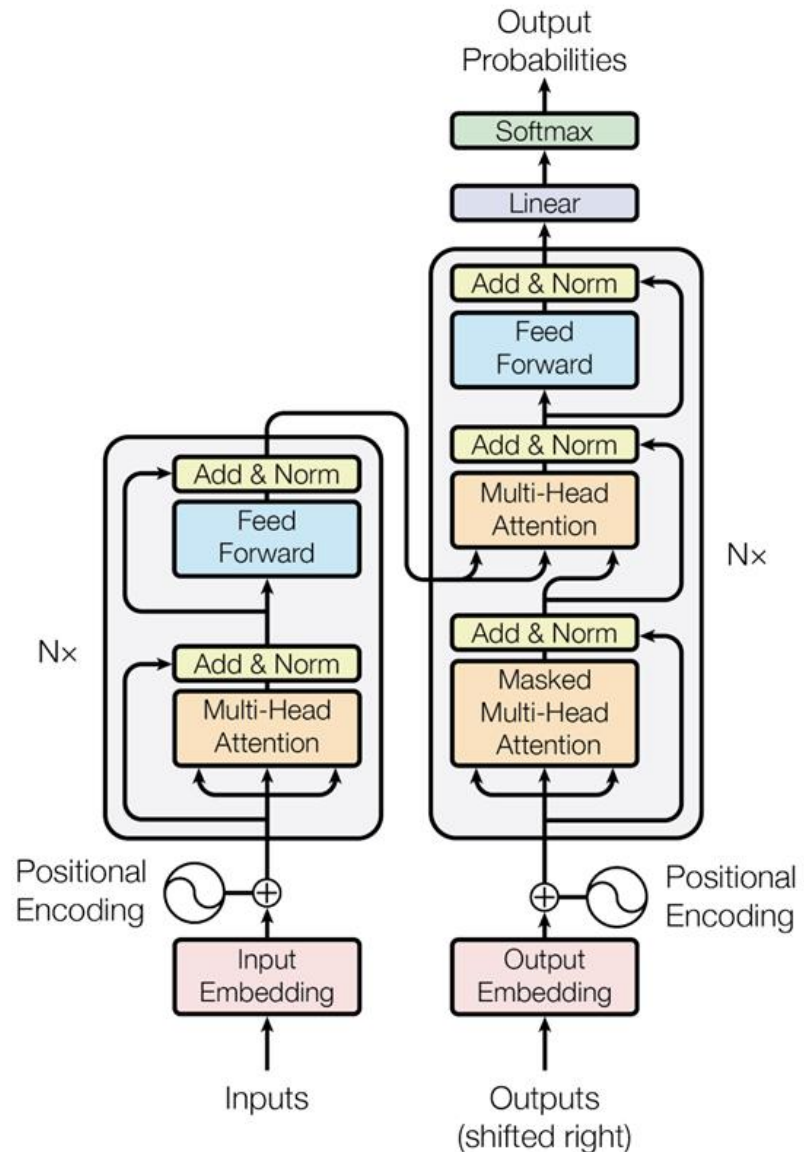
Initially, $N = 6$ layers

the **Encoder** has two sub layers:

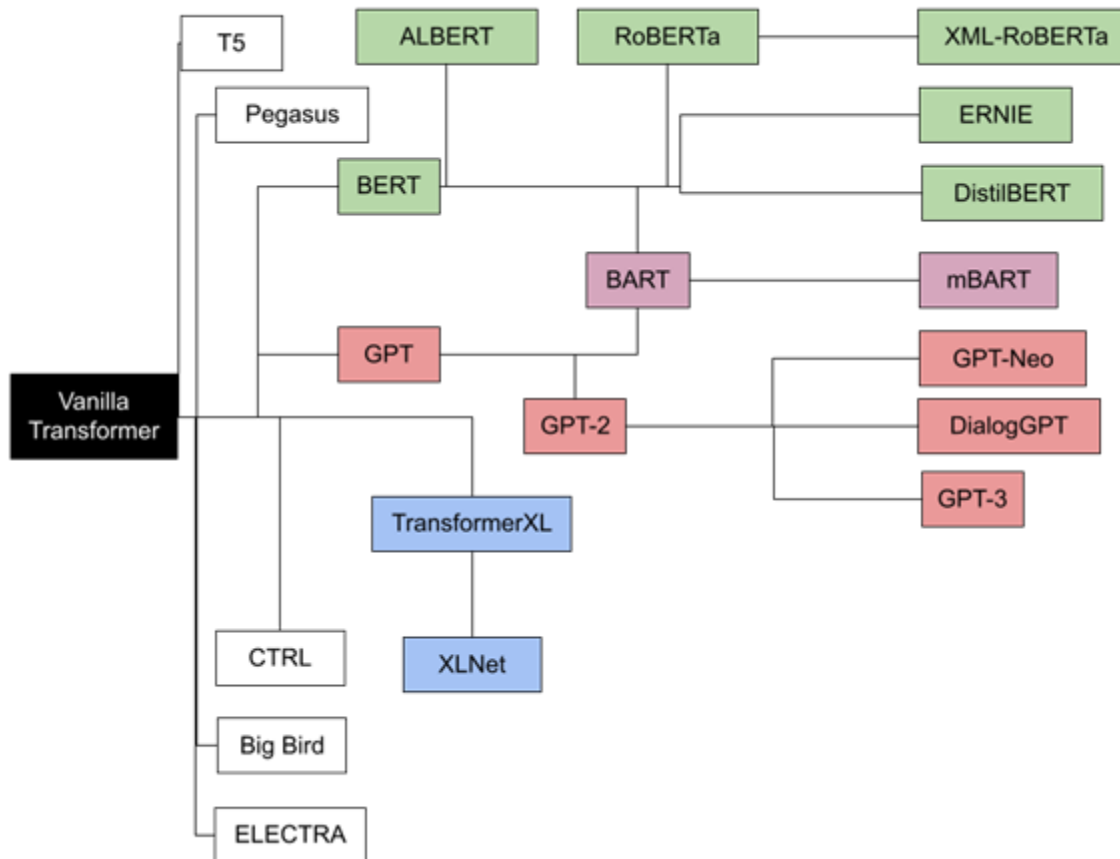
- a multi-head attention layer, and
- a simple feed forward network.

The **Decoder** adds a third sublayer: another multi-head attention layer over the output of the Encoder.

- masked to prevent attention to subsequent positions.
- positional encodings



Family of Transformers



[Transformer models: an introduction and catalog — 2022 Edition | by Xavier Amatriain | Mar, 2022 | Medium](https://xamat.medium.com/transformers-models-an-introduction-and-catalogue-2022-edition-2d1e9039f376)
<https://xamat.medium.com/transformers-models-an-introduction-and-catalogue-2022-edition-2d1e9039f376>

Transfer learning

- [Pretrained Transformer models](#) can adapt to tasks they have not been trained on
- *Re-use* the pretrained model on a new task, by adapting it with a much smaller data set (fine tuning)
- Useful for [other tasks](#), e.g., vision, audio and music applications

- Integrated into the main AI frameworks (e.g., Pytorch)
- [Huggingface](#), a startup commercializing an open-source Transformers library.
- [GPT-3](#) is a Transformer model introduced by OpenAI in May 2020 as a follow up to their earlier GPT and GPT-2, commercialized through a partnership between OpenAI and Microsoft.

Transformers for IR tasks

<https://github.com/Albert-Ma/awesome-pretrained-models-for-information-retrieval>

Pretrained transformer for Greek language

<https://huggingface.co/nlpauieb/bert-base-greek-uncased-v1>

Embeddings for Greek Language

<http://archive.aueb.gr:7000/resources/>

Neural (semantic) search in Lucene (solr)

Given a dense vector q that models the query (information need) calculate the distance (Euclidean, dot product, etc.) between q and every vector d that represents a document

DenseVectorField type

Knn Query Parser

Nearest Neighborhood Search

Proximity Neighborhood Graph

- Vertices (documents) are mapped to nodes
- Edges between “similar” vertices

Navigable small world graph

<https://sease.io/2022/01/apache-solr-neural-search.html>

Προαιρετική άσκηση

Χρησιμοποιείτε κάποια word embeddings για να ανακαλύψτε bias

πχ με χρήση αναλογιών

Μια γνωστή τέτοια αναλογία:

Man is to computer what woman is to homemaking

End of lecture

Χρησιμοποιήθηκε υλικό από

- CS276: Information Retrieval and Web Search, Christopher Manning and Pandu Nayak, Lecture 14: Distributed Word Representations for Information Retrieval
- <https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/>

Μια περιγραφή του skipgram:

Chris McCormick

<http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>

Δείτε και το

<https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/>