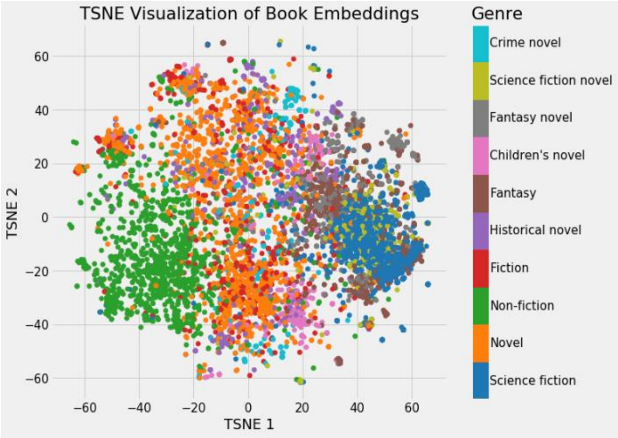
Τι θα δούμε σήμερα

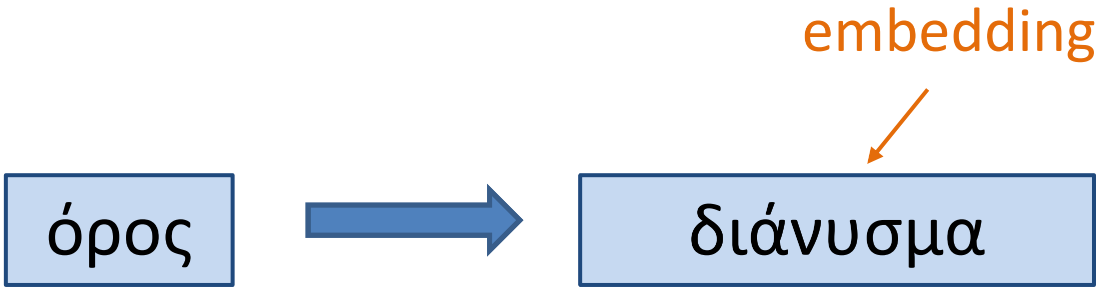
- Διανυσματική αναπαράσταση λέξεων (word embeddings)
- Μερικές πληροφορίες για την εργασία



Books on Wikipedia colored by genre in two dimensions. Source: Koehrsen 2018

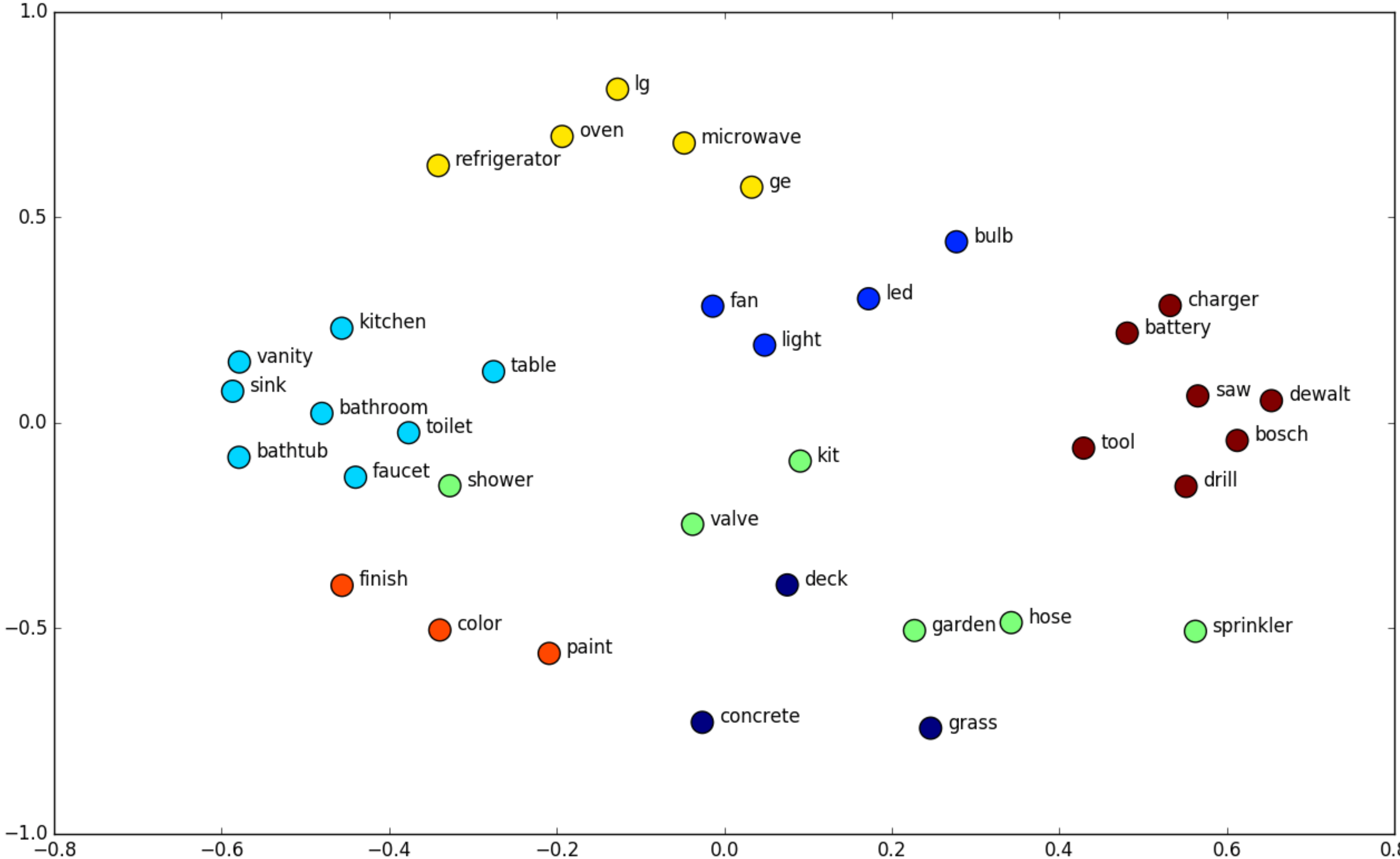
Word embeddings

Πρόβλημα: Διανυσματική *αναπαράσταση* (representation) όρων



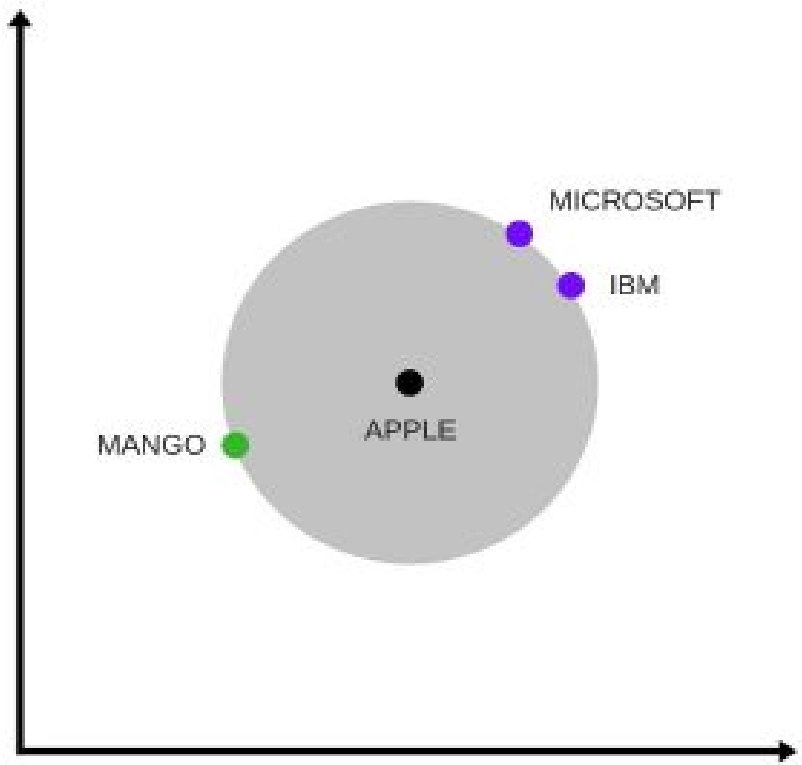
Στόχος: Όμοιοι όροι -> όμοια διανύσματα

Παράδειγμα: 2-διάστατα embeddings



http://suriyadeepan.github.io

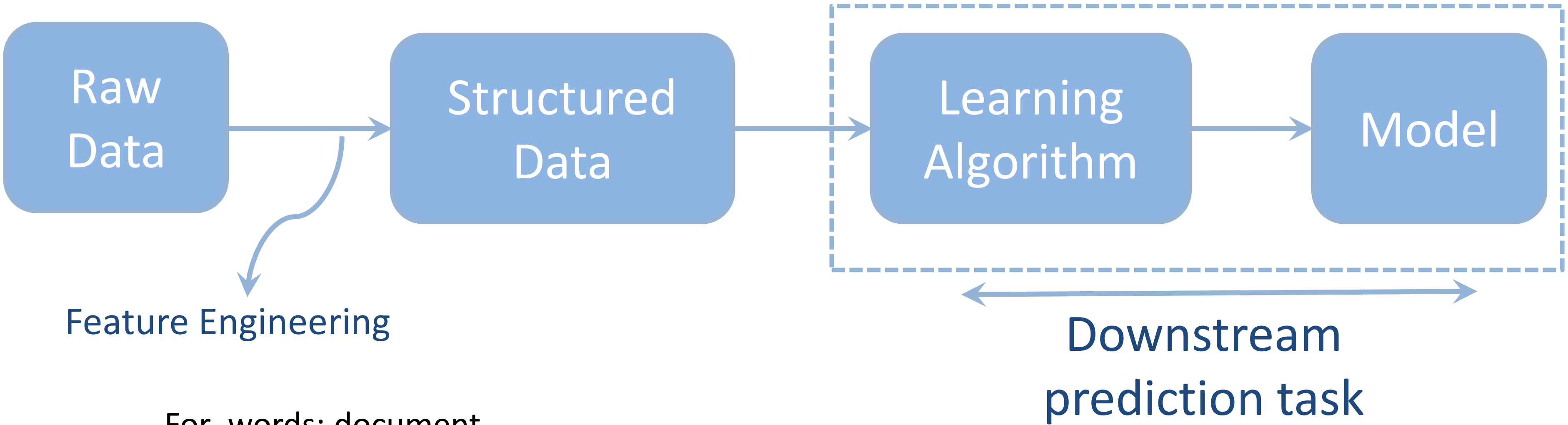
Apple: φρούτο και εταιρεία



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

Embeddings: why?

Machine learning lifecycle



For words: document occurrences, k-grams. etc

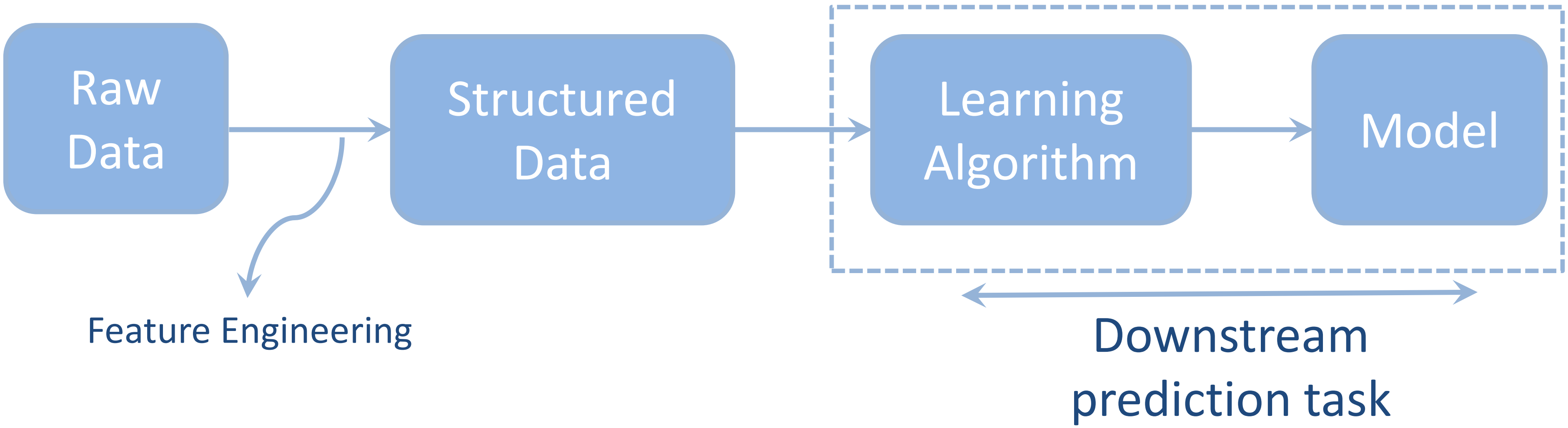
For documents: length, words, etc

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

Classification
Learning to rank
Clustering

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)

One-hot vectors

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

- Διατάσσουμε τις λέξεις αλφαβητικά
- Αναπαριστούμε κάθε λέξη με ένα $\mathrm{R}^{|V|x1}$ διάνυσμα που έχει παντού 0 και μόνο έναν 1 στη θέση που αντιστοιχεί στη θέση της λέξης στη διάταξη

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

- Καμία πληροφορία για ομοιότητα
- Πολλές διαστάσεις

Term-Document co-occurrence matrix

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

- Κατασκευάζουμε ένα |V|xM πίνακα με τις εμφανίσεις των λέξεων στα έγγραφα
- Αναπαριστούμε κάθε λέξη με ένα $\mathrm{R}^{|\mathrm{M}|x1}$

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

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

	d1	**d2**	**d3**	**d4**	**d5**
a	1	2	2	1	2
b	1	1	0	2	1
c	1	0	2	0	1
d	0	1	1	0	1
e	0	0	1	1	0
f	0	0	1	0	0

Word vector for c

- Πολλές διαστάσεις
- Πρόβλημα κλιμάκωσης με τον αριθμό των εγγράφων

Window-based co-occurrence matrix

- Κατασκευάζουμε ένα $|V|x|V|$ affinity-matrix για τις λέξεις: για δύο λέξεις, μετράμε τον αριθμό των φορών που αυτές δύο λέξεις εμφανίζονται μαζί σε έγγραφα
- Συγκεκριμένα, μετράνε τον αριθμό των φορών που κάθε λέξη εμφανίζεται μέσα σε ένα *παράθυρο* συγκεκριμένου μεγέθους 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

W = 1 (σε απόσταση 1)

	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

Window-based co-occurrence matrix

- Κατασκευάζουμε ένα $|V|x|V|$ affinity-matrix για τις λέξεις: μετράμε τον αριθμό των φορών που δυο λέξεις εμφανίζονται μέσα σε ένα *παράθυρο* συγκεκριμένου μεγέθους

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

$X =$

	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

- Πολλές διαστάσεις

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

Singular Value Decomposition

From dimension d to dimension r

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

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

- $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$: singular values (square roots of eigenvals AA^T, A^TA)
- $\vec{u}_1, \vec{u}_2, \cdots, \vec{u}_n$: left singular vectors (eigenvectors of AA^T)
- $\vec{v}_1, \vec{v}_2, \cdots, \vec{v}_n$: right singular vectors (eigenvectors of A^TA)

- Cut the singular values at some index r (get the largest r such values)
- Get the first r columns of U to get the r-dimensional vectors

Singular Value Decomposition

Ar best approximation of A
(Frobernius norm)

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

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

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

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

Αλλά

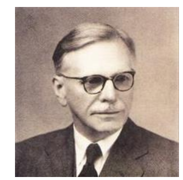
- Δύσκολο να ενημερώσουμε, πχ, αλλάζουν οι διαστάσεις συχνά
- Αραιός πίνακας
- Πολύ μεγάλες διαστάσεις

Θα δούμε μια τεχνική που βασίζεται σε επαναληπτικές μεθόδους

word2vec

Basic Idea

- You can get a lot of value by representing a word by means of its neighbors
- "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

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

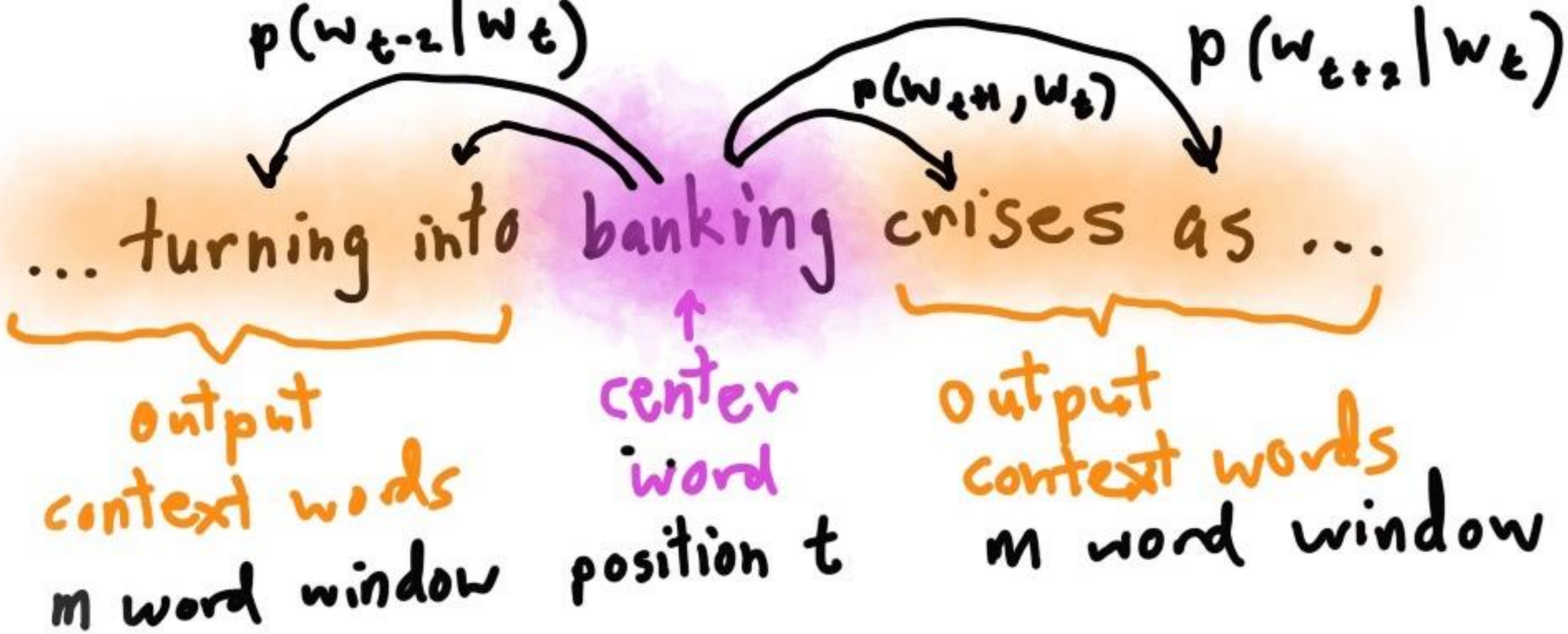
↖ These words will represent *banking* ↗

Basic idea

Define a model that aims to predict between a center word w_c and context words in some window of length m in terms of word vectors

$$\mathrm{P}(w_c \,|w_{c-m,} \dots, w_{c-1,}\ w_{c+1} \dots, w_{c+m})$$

Loss function 1-P that we want to minimize



p(w_{t-2}|w_t)
p(w_{t+1}, w_t)
p(w_{t+2}|w_t)
... turning into banking crises as ...
output context words
m word window
center word
position t
output context words
m word window

Basic idea

Define a model that aims to predict between a center word w_c and context words in some window of length m in terms of word vectors

$$\mathrm{P}(w_c \,|w_{c-m,} \ldots, w_{c-1,}\ w_{c+1} \ldots, w_{c+m})$$

Loss function 1-P that we want to minimize

Pairwise probabilities

Independence assumption (bigram model)

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=2}^{n} P(w_i|w_{i-1})$$

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

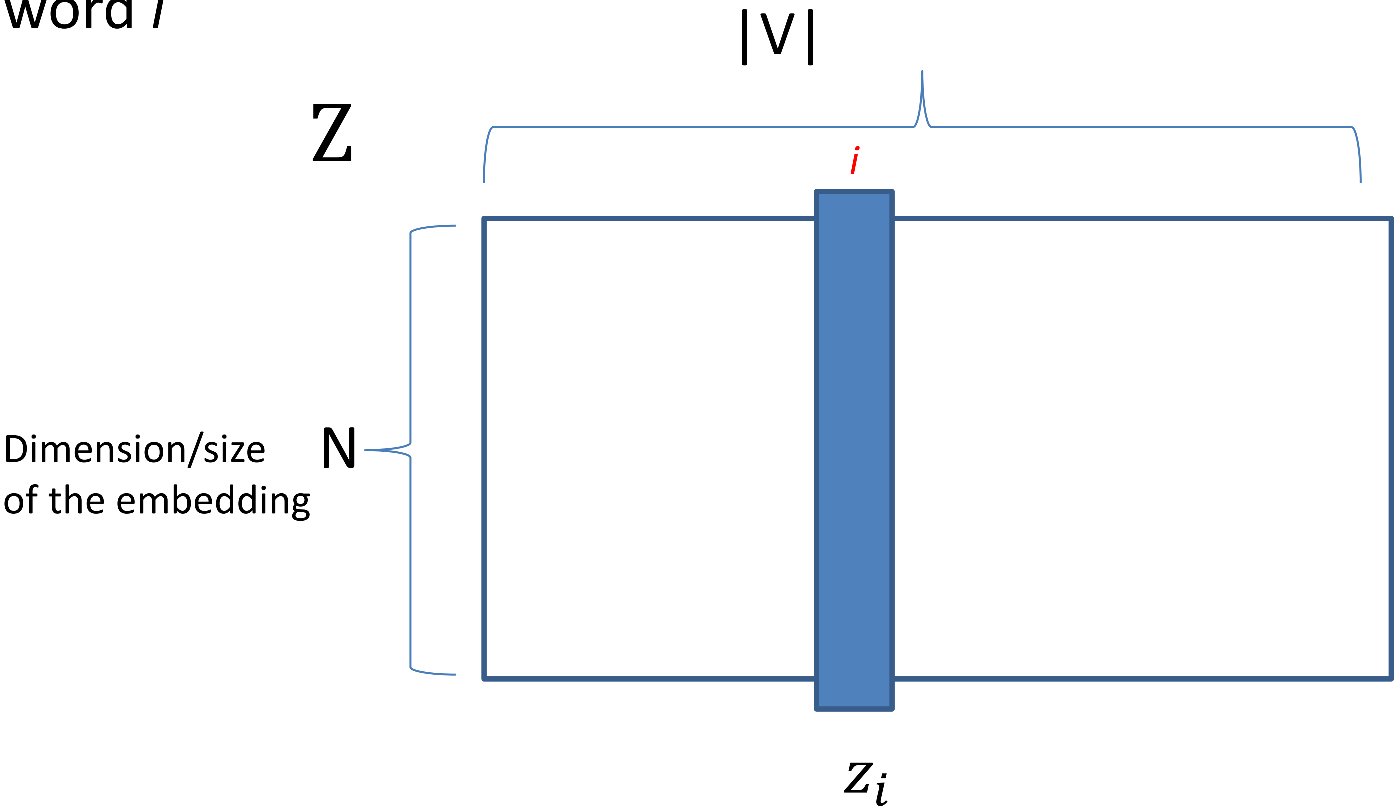
Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, Jeffrey Dean: *Distributed Representations of Words and Phrases and their Compositionality.* NIPS 2013: 3111-3119

|V| number of words
N size of embedding
m size of the window (context)

Note

Each word is assigned *a single N-dimensional vector*

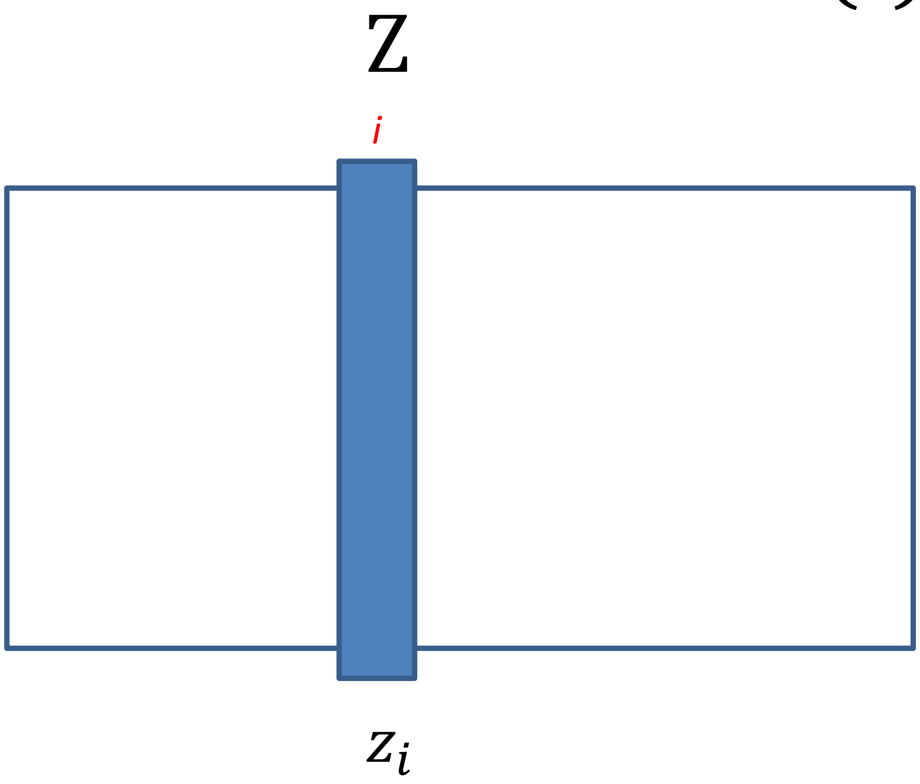
Learn embedding matrix Z: each column *i* is the embedding z_i of word *i*



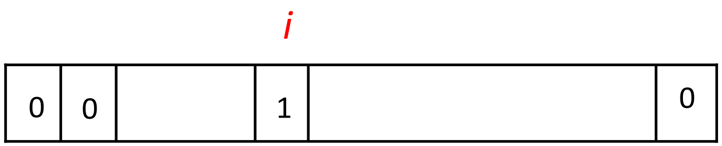
Note

Encoder is an embedding lookup

$$ENC(i) = Z\,I_i$$



$One\ hot\ vector\ I_i$



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

CBOW

|V| number of words
N size of embedding
m size of the window (context)

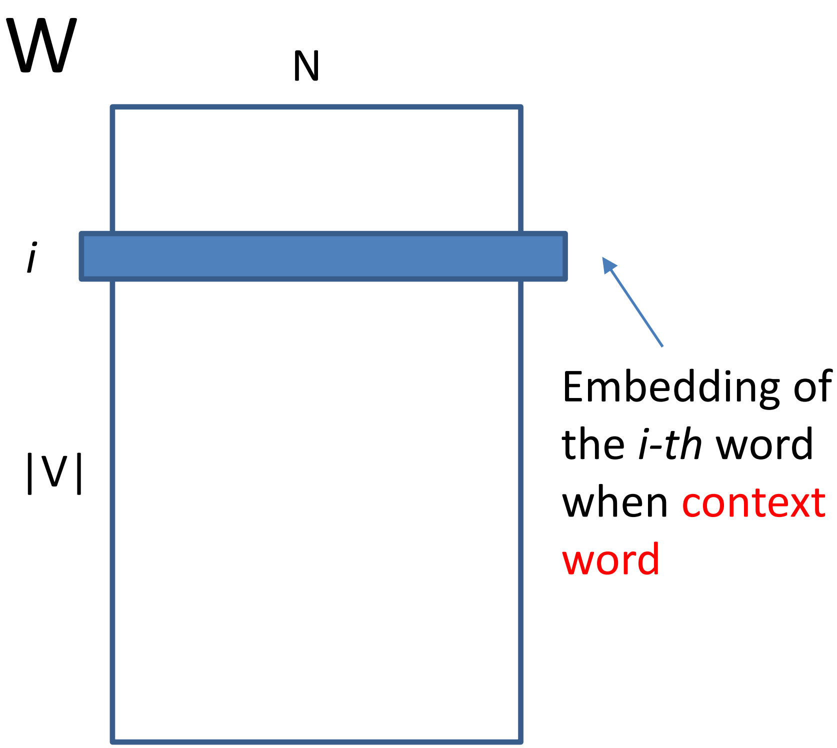
Use a window of context words to predict the center word

Input: 2m context words
Output: center word
each represented as a one-hot vector

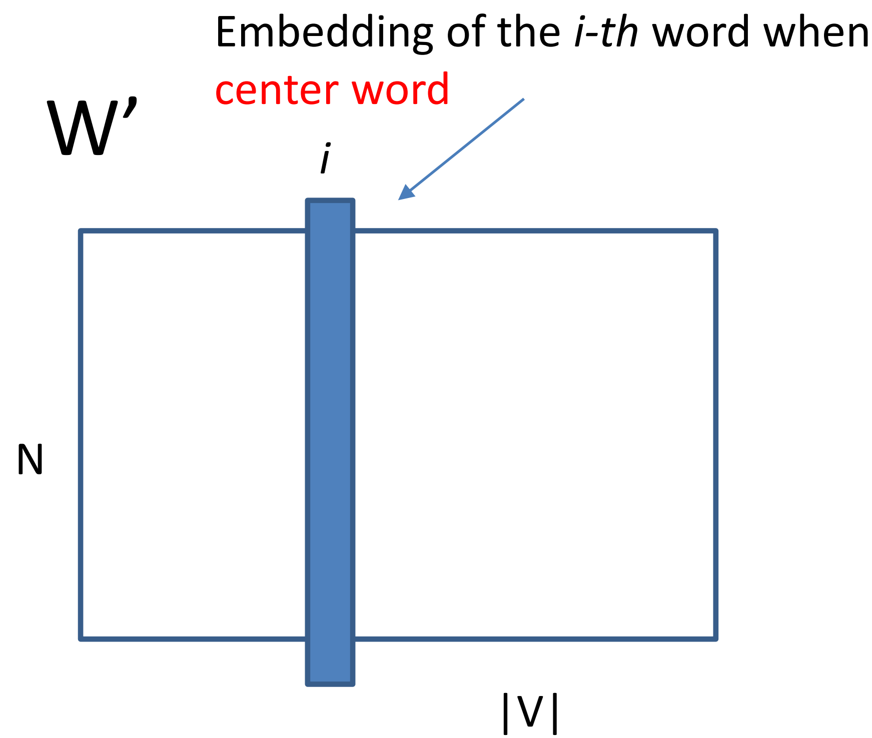
CBOW

Use a window of context words to predict the center word

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



|V| x N context embeddings when *input*



N x |V| center embeddings when *output*

CBOW

Use a window of context words to predict the center word

Intuition
The W'-embedding of the *center word* should be *similar* to 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

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

2. Compute the *embeddings of the context words*

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

3. *Average* these vectors

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

4. Generate a *score vector*

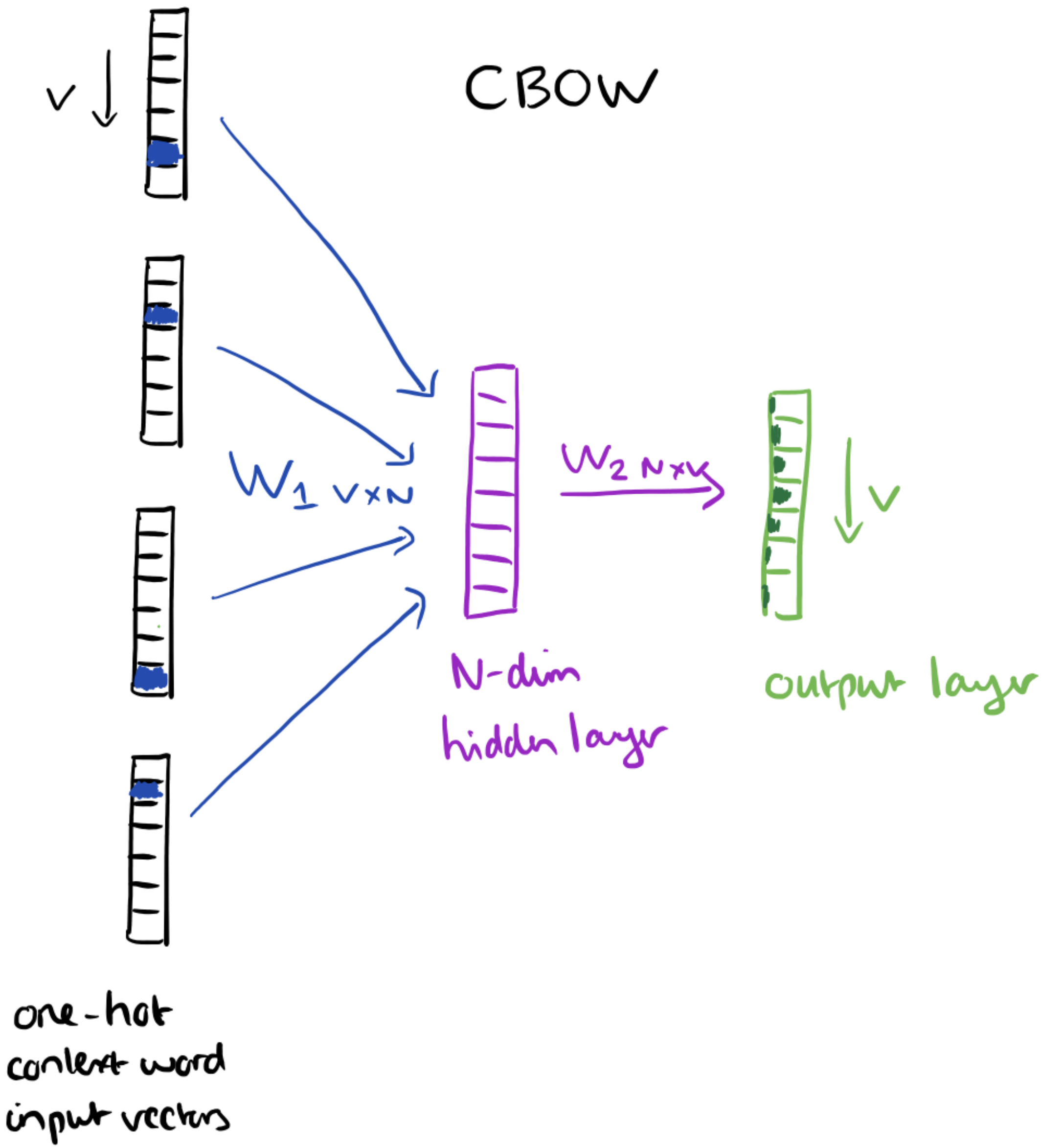
z = W' $\hat{v}$

dot product, (embedding of center word), similar vectors close to each other

5. Turn the *score vector to probabilities*

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

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


CBOW
V
W1 VxN
W2 NxV
V
N-dim
hidden layer
output layer
one-hot
context word
input vectors

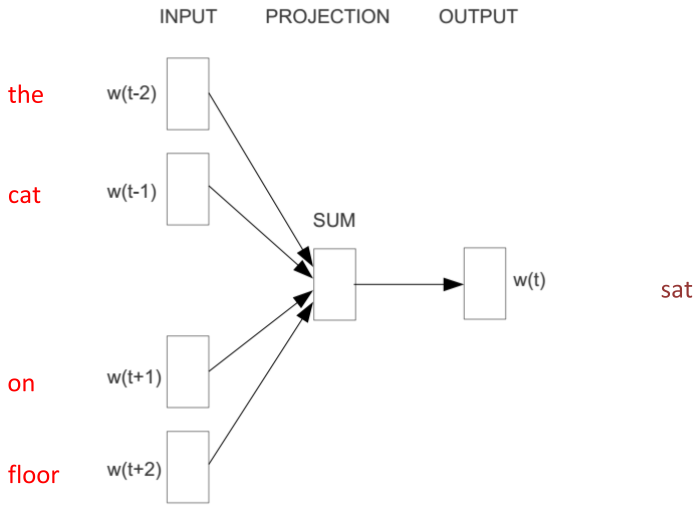
Softmax

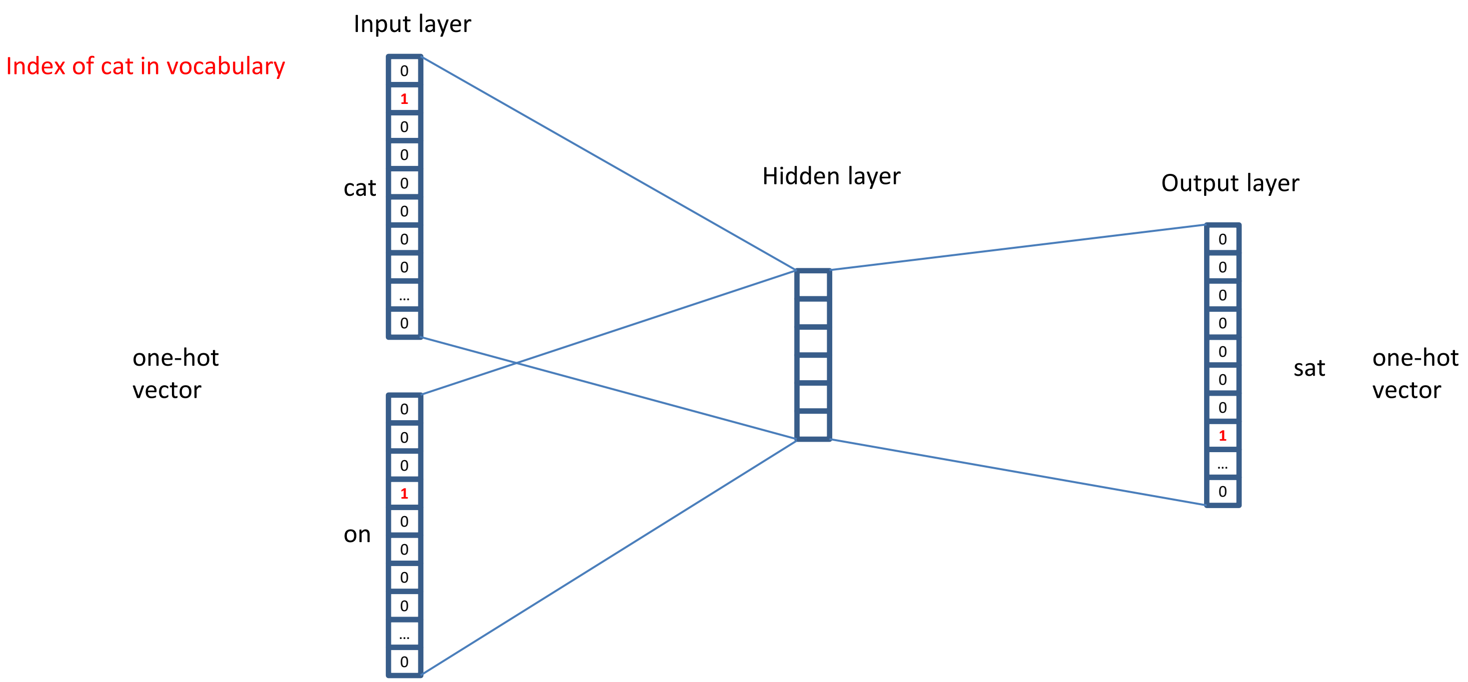
Exponentiate to make positive

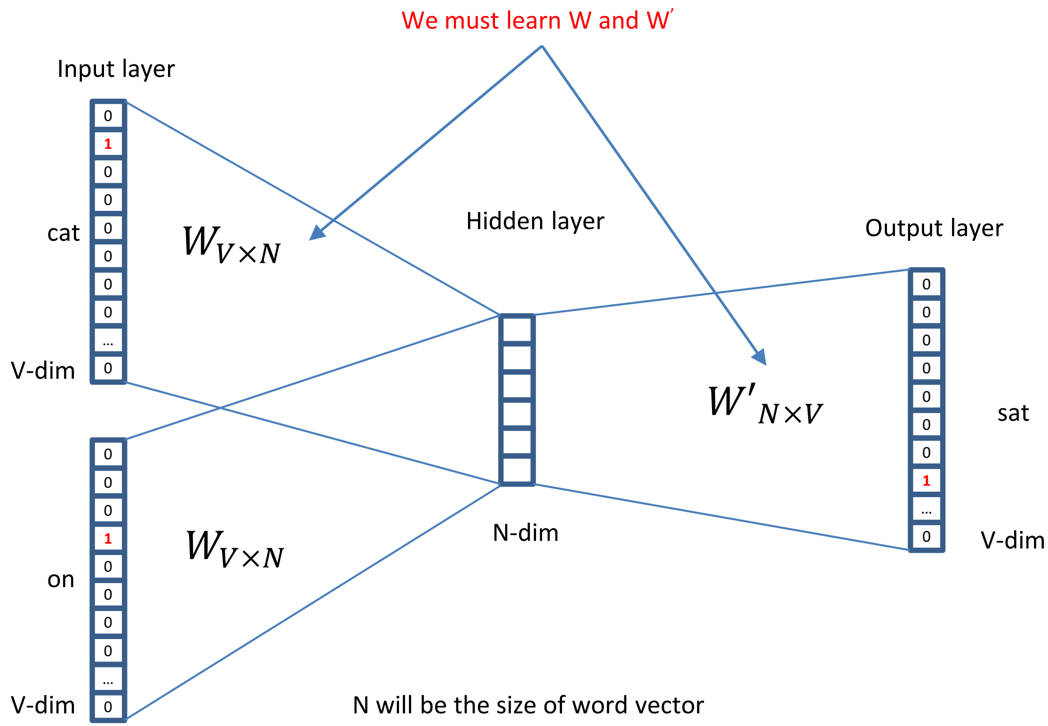
$$p_i = \frac{e^{u_i}}{\sum_j e^{u_j}}$$

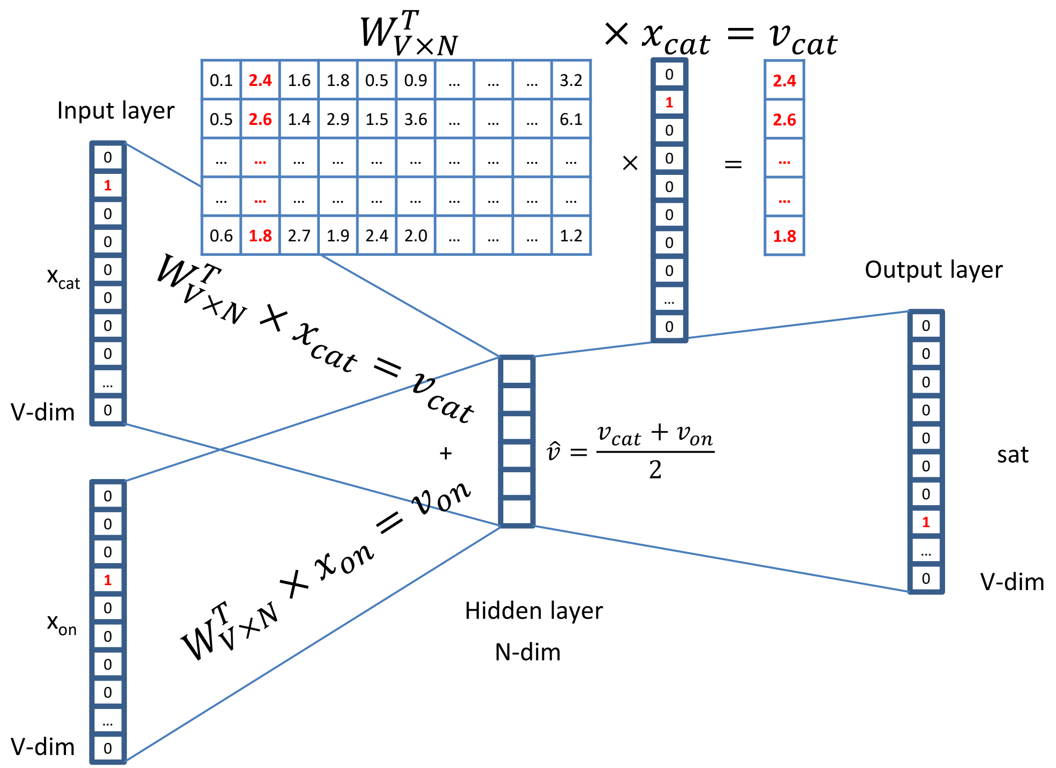
Normalize to give probability

- E.g. “The cat sat on floor”
  - Window size = 2

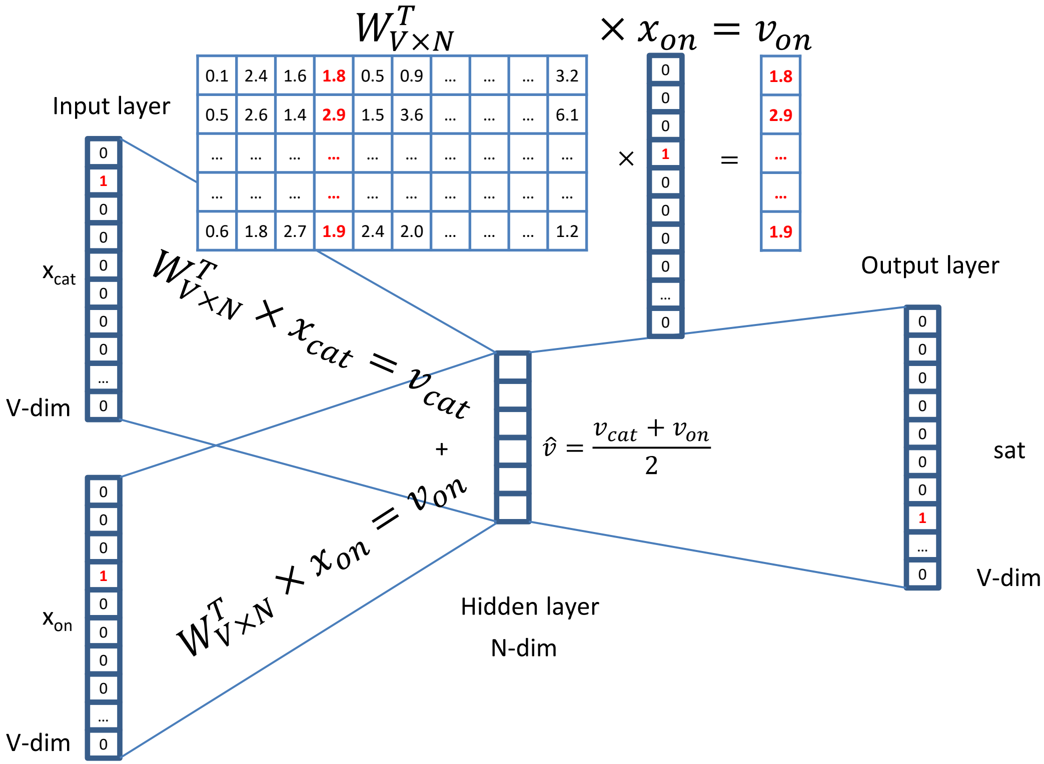



Input layer
Index of cat in vocabulary
cat
0
1
0
0
0
0
0
0
...
0
Hidden layer
Output layer
one-hot
vector
on
0
0
0
1
0
0
0
0
...
0
sat
one-hot
vector
0
0
0
0
0
0
0
1
...
0

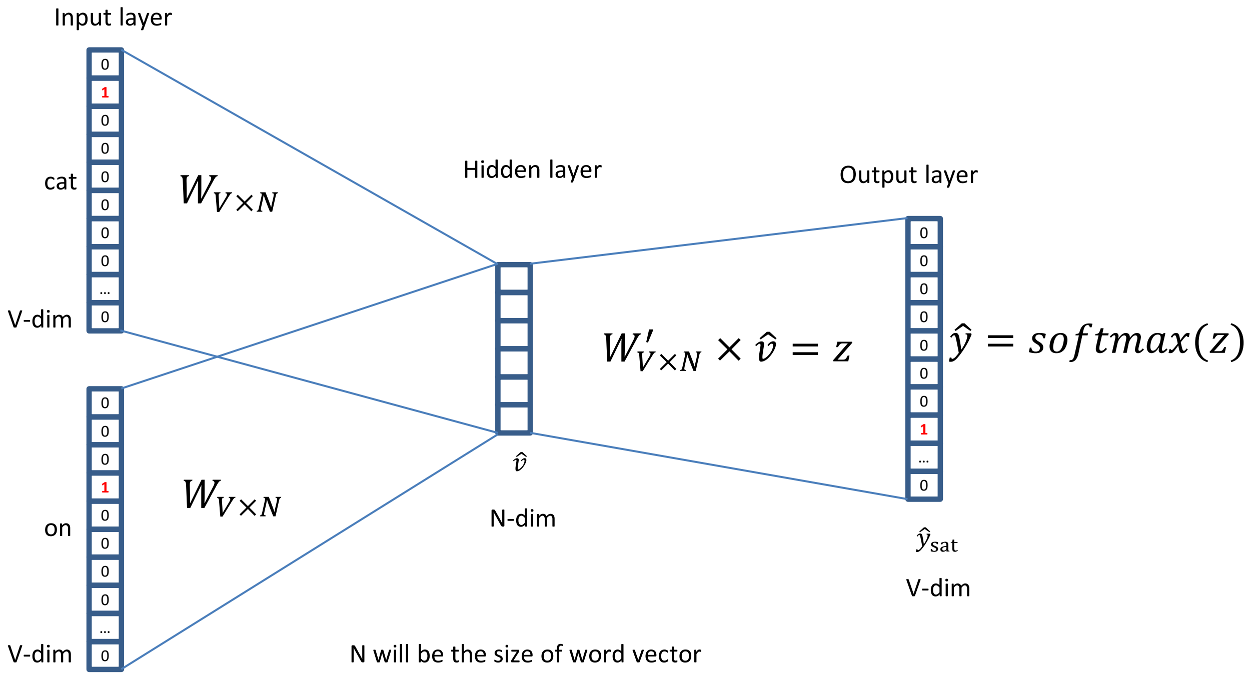

We must learn W and W′
Input layer
cat
$W_{V\times N}$
Hidden layer
Output layer
V-dim
$W'_{N\times V}$
sat
on
$W_{V\times N}$
N-dim
V-dim
V-dim
N will be the size of word vector

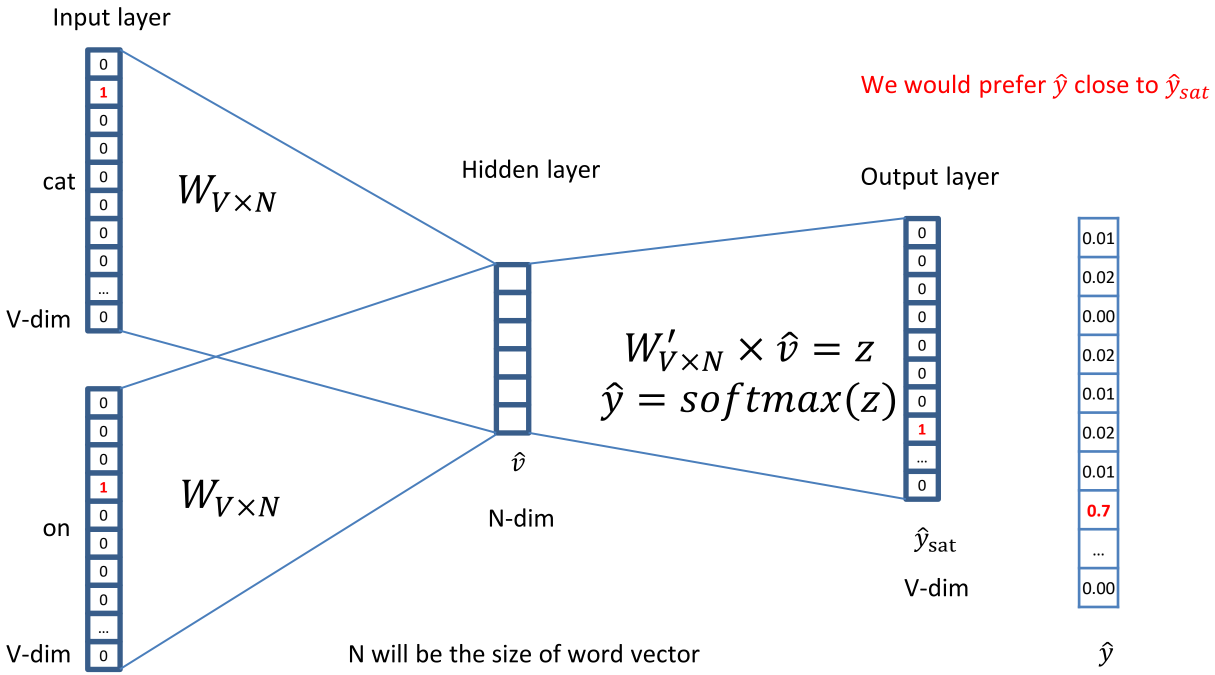


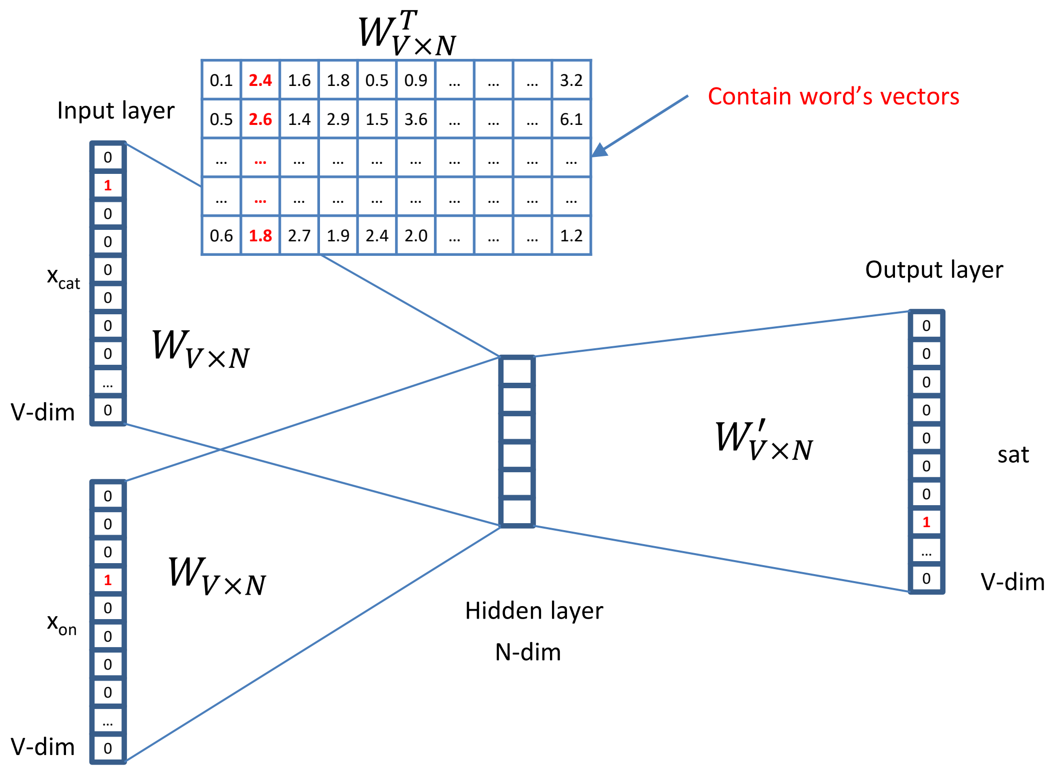
$W_{V\times N}^{T} \times x_{cat} = v_{cat}$
Input layer
x_{cat}
V-dim
x_{on}
V-dim
$W_{V\times N}^{T} \times x_{cat} = v_{cat}$
+
$W_{V\times N}^{T} \times x_{on} = v_{on}$
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim



$W^T_{V\times N}$
$\times x_{on} = v_{on}$
Input layer
0.1 2.4 1.6 1.8 0.5 0.9 ... ... 3.2
0.5 2.6 1.4 2.9 1.5 3.6 ... ... 6.1
0.6 1.8 2.7 1.9 2.4 2.0 ... ... 1.2
1.8 2.9 ... ... 1.9
x_{cat}
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
+
$W^T_{V\times N} \times x_{on} = v_{on}$
x_{on}
V-dim
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim


Input layer
cat
V-dim
on
V-dim
$W_{V \times N}$
$W_{V \times N}$
Hidden layer
$\hat{v}$
N-dim
$W'_{V \times N} \times \hat{v} = z$
Output layer
$\hat{y} = softmax(z)$
$\hat{y}_{\text{sat}}$
V-dim
N will be the size of word vector


We would prefer $\hat{y}$ close to $\hat{y}_{sat}$
Input layer
cat
V-dim
on
V-dim
$W_{V \times N}$
$W_{V \times N}$
Hidden layer
$\hat{v}$
N-dim
Output layer
$W'_{V \times N} \times \hat{v} = z$
$\hat{y} = softmax(z)$
$\hat{y}_{sat}$
V-dim
0.01
0.02
0.00
0.02
0.01
0.02
0.01
0.7
...
0.00
$\hat{y}$
N will be the size of word vector



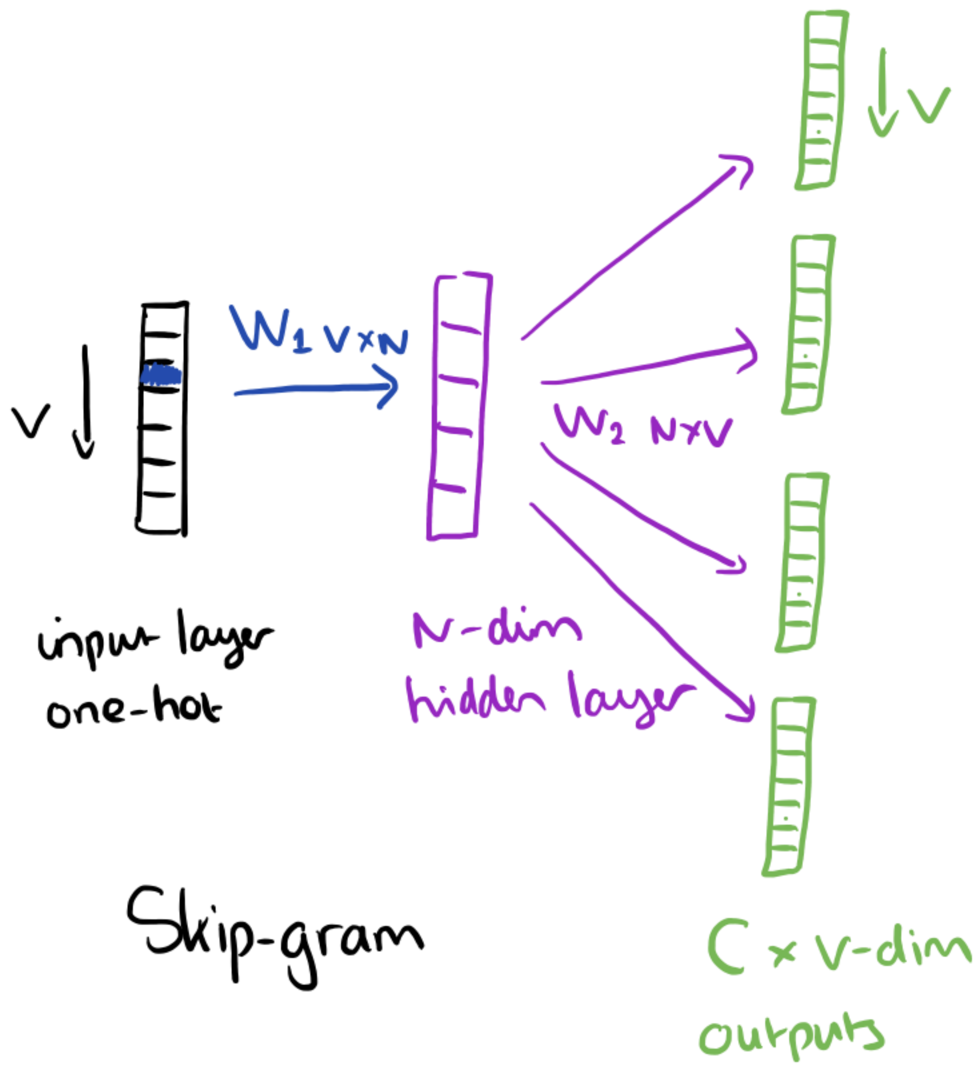
We can consider either W (context) or W' (center) as the word's representation.
Or even take the average.

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: N x |V|, input matrix, word representation as center word
W': |V| x N, output matrix, word representation as context word


V
W1 V×N
input layer
one-hot
N-dim
hidden layer
W2 N×V
V
C × V-dim
outputs

Skip-gram

Skipgram

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

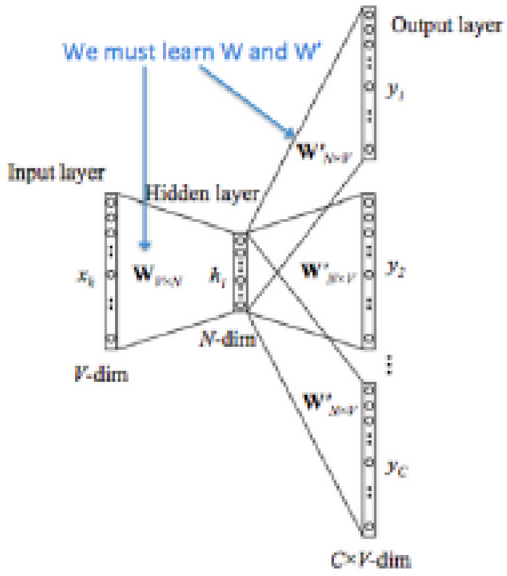
$y^{(j)}$ one hot vector for context words

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

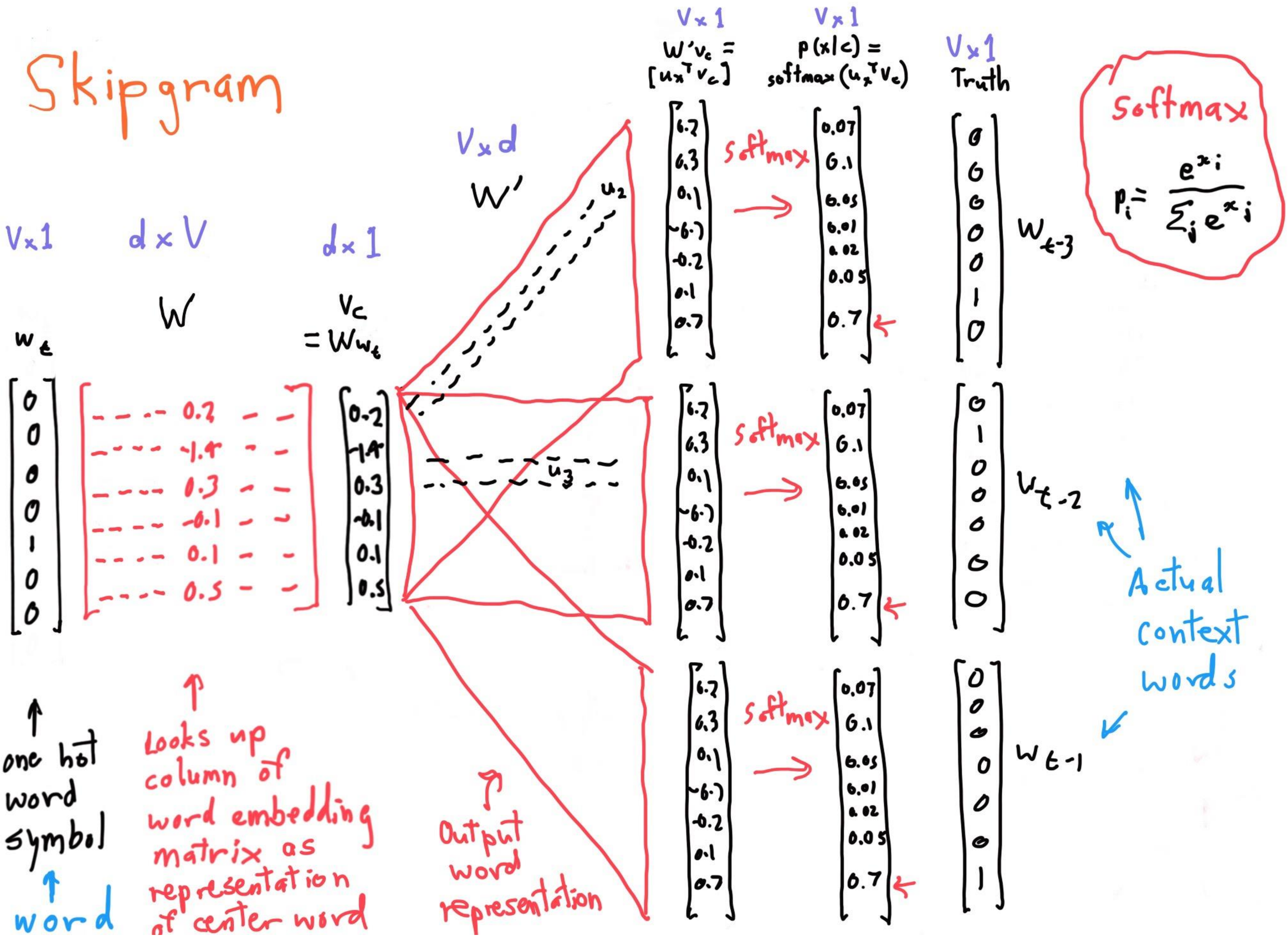
2. Get the *embedding of the center word*
$v_c = W\ x$

3. Generate a *score vector for each context word*
z = $W' v_c$

5. Turn the *score vector into probabilities*
$\hat{y}$ = *softmax(z)*



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


Skipgram
V×1
d×V
d×1
V×d
V×1
V×1
V×1
w_t
W
$v_c = Ww_t$
W'
$W'v_c = [u_x^T v_c]$
$p(x|c) = \text{softmax}(u_x^T v_c)$
Truth
u_2
u_3
softmax
softmax
softmax
w_{t-3}
w_{t-2}
w_{t-1}
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

Skipgram

- For each word $t = 1 \ldots T$, predict surrounding words in a window of "radius" m of every word.
- Objective function: Maximize the probability of any context word given the current center word:

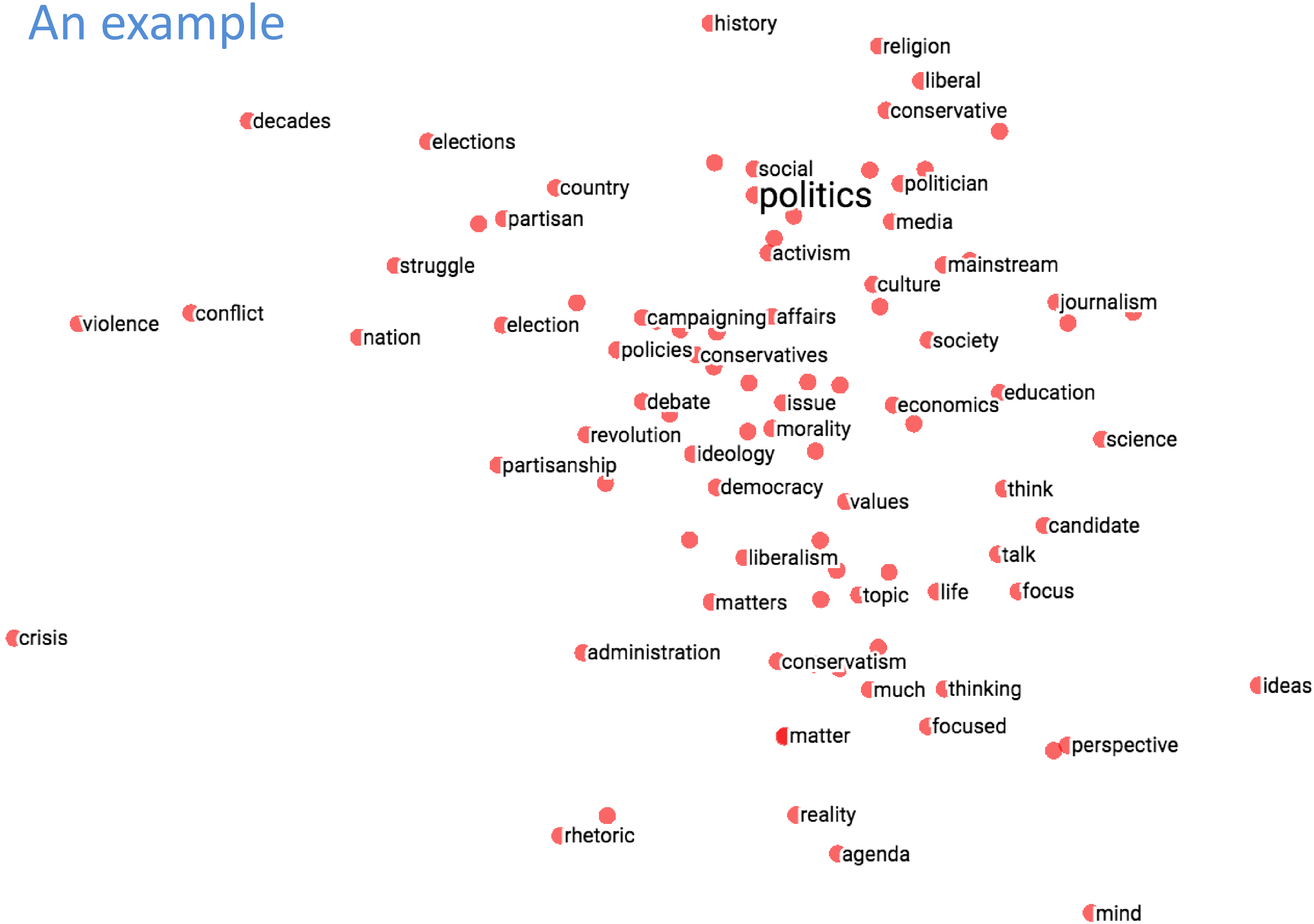
$$J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} p(w_{t+j} \mid w_t; \theta)$$

Negative Log Likelihood

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

where θ represents all variables we will optimize

An example


history
religion
liberal
conservative
decades
elections
social
politics
politician
country
partisan
media
activism
struggle
mainstream
culture
journalism
conflict
violence
nation
election
campaigning
affairs
policies
conservatives
society
education
debate
issue
economics
revolution
morality
science
ideology
partisanship
democracy
think
values
candidate
liberalism
talk
life
focus
matters
topic
crisis
administration
conservatism
ideas
much
thinking
matter
focused
perspective
reality
rhetoric
agenda
mind

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**

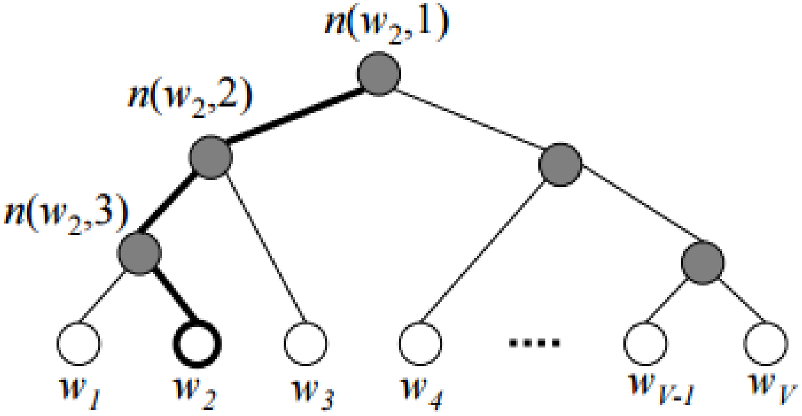
1. Hierarchical softmax
2. Negative sampling

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, Jeffrey Dean: *Distributed Representations of Words and Phrases and their Compositionality.* NIPS 2013: 3111-3119

Training methods: hierarchical softmax

Στόχος: Αντί να μάθουμε ένα διάνυσμα ανά λέξη, δηλαδή |V| διανύσματα, να μάθουμε $log_2(|V|)$ διανύσματα

Πως; Ένα δυαδικό δέντρο
Ένα φύλλο ανά λέξη
Μαθαίνουμε την αναπαράσταση των εσωτερικών κόμβων
Αναπαράσταση λέξης: concat των αναπαραστάσεων των κόμβων στο μονοπάτι από τη ρίζα στη λέξη



Training methods: negative sampling

Στόχος: Να βελτιώσουμε την ποιότητα των αναπαραστάσεων με χρήση αρνητικών δειγμάτων

- Για κάθε ένα θετικό, K αρνητικά δείγματα
- Χρήση unigram μοντέλου για να τα κατασκευάσουμε

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

- 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

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

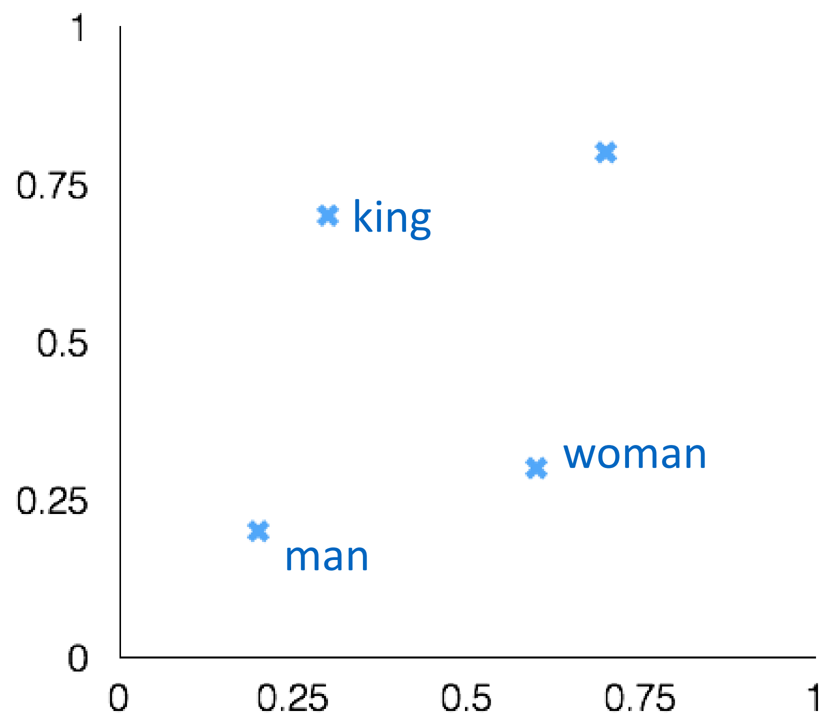
Test for linear relationships, examined by Mikolov et al.

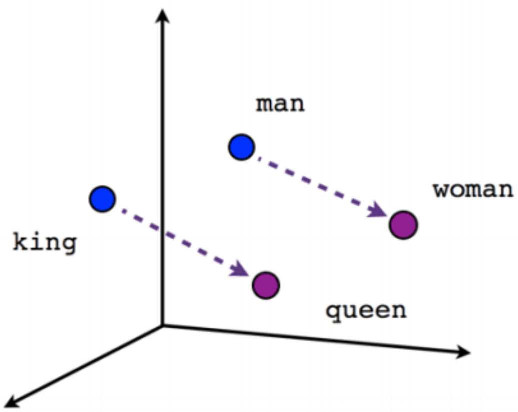
a:b :: c:?

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

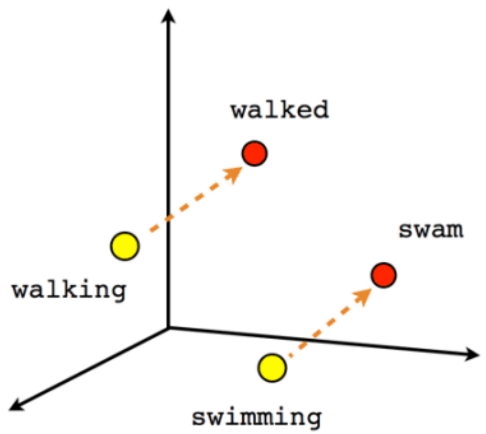
man:woman :: king:?

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

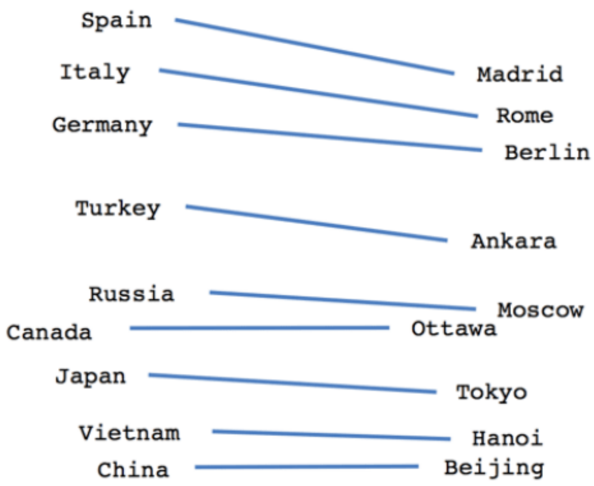




Male-Female



Verb tense



Country-Capital

- Στην *εργασία*, σας ζητείτε να χρησιμοποιείστε word embeddings
  - Πρέπει να επιλέξετε πως
  - Θα αξιολογηθεί και η καταλληλότητα/πρωτοτυπία/χρησιμότητα

- Στη συνέχεις θα δούμε
  - pretrained embeddings
  - μερικές εφαρμογές

Σύντομη περιγραφή της εργασίας

1. Θα *συλλέξετε* έναν αριθμό από Wikipedia άρθρα
Αυτή θα είναι η συλλογή σας.

2. Θα *υλοποιήσετε* ένα σύστημα αναζήτησης (ΣΑΠ) αυτών των άρθρων:
Ο χρήστης θα δίνει μία ή περισσότερες λέξεις κλειδιά και το σύστημα θα επιστρέφει τα πιο συναφή άρθρα σε διάταξη με βάση τη συνάφεια τους στην ερώτηση

Για να υλοποιήσετε το σύστημα θα χρησιμοποιείστε τη *Lucene*. Περισσότερα την επόμενη ώρα.

Global vs. local embedding

global	local
cutting	tax
squeeze	deficit
reduce	vote
slash	budget
reduction	reduction
spend	house
lower	bill
halve	plan
soften	spend
freeze	billion

Πάνω σε ποια συλλογή (corpus) φτιάχνουμε τα embeddings;
Προτάσεις από ποια κείμενα θα χρησιμοποιήσουμε;

[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)

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

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

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/

Extra slides

Hierarchical softmax and negative sampling

Hierarchical softmax

Instead of learning O(|V|) vectors, learn O(log(|V|) vectors

How?

- Build a binary tree with leaves the words, *and learn one vector for each internal node*.
- The value for each word w is the product of the values of the internal nodes in the path from the root to w.

The probability of a word being the context word is defined as:

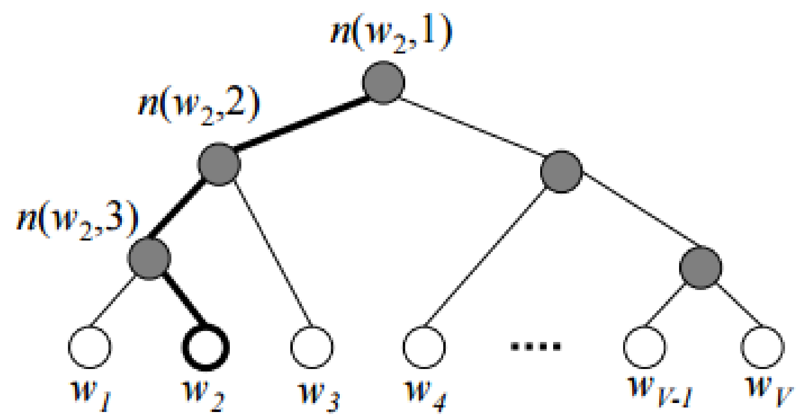
compares the similarity of the ***input vector*** v_w to each internal node vector

$$p(c|w) = \prod_{j=1}^{L(w)-1} \sigma([\![n(c, j+1) = ch(n(w,j))]\!] \cdot {v_{n(c,j)}}^T v_w)$$

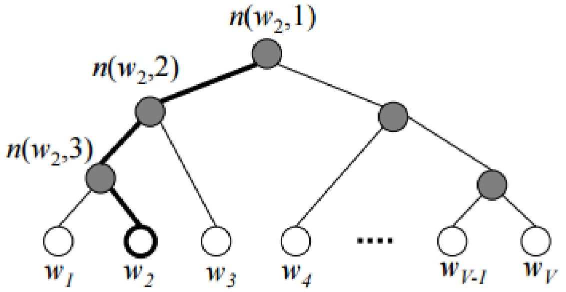
returns 1 if the path goes left, - 1 if it goes right

where:

- $n(w,j)$ – is the j-th node on the path from the root to *w*. $n(w,1) = \text{root}$, $n(w, L(w)) = \text{parent of w}$
- $L(w)$ – is the length of the path from root to *w*. $L(w_2) = 3$
- $ch(n)$ – is the left child of node n.
- $[\![x]\!] = \begin{cases} 1 & if\ x\ is\ true \\ -1 & otherwise \end{cases}$
- $\sigma(x) = \frac{1}{1+e^{-x}}$



Suppose we want to compute the probability of w_2 being the output word.



- The probabilities of going right/left in a node *n* are:
  - $p(n, left) = \sigma\left({v_n}^T v_w\right)$
  - $p(n, right) = 1 - \sigma\left({v_n}^T v_w\right) = \sigma\left(-{v_n}^T v_w\right)$

$$p(w_2 = c) = p(n(w_2, 1), left) \cdot p(n(w_2, 2), left) \cdot \ p(n(w_2, 3), right)$$

$$= \sigma\left({v_{n(w_2,1)}}^T v_w\right) \cdot \sigma\left({v_{n(w_2,2)}}^T v_w\right) \cdot \sigma\left(-{v_{n(w_2,3)}}^T v_w\right)$$

Complexity improved even further using a Huffman tree:

- Designed to compress binary code of a given text.
- A full binary suffix tree that guarantees a minimal average weighted path length when some words are frequently used.

Negative Sampling

- For each positive example we draw ***K*** *negative examples*.
- The negative examples are drawn according to the unigram distribution of the data

$$P_D(c) = \frac{\#(c)}{|D|}$$

$\mathrm{p}(D=1|w,c)$ is the probability that $(w,c) \in D$.

$\mathrm{p}(D=0|w,c) = 1 - \mathrm{p}(D=1|w,c)$ is the probability that $(w,c) \notin \mathrm{D}$.

For negative samples: $p(D=1|w,c)$ must be low $\Rightarrow$ $p(D=0|w,c)$ will be high.

$$\arg\max_{\theta} \prod_{(w,c)\in D} p(D=1|c,w;\theta) \prod_{(w,c)\in D'} p(D=0|c,w;\theta)$$

$$= \arg\max_{\theta} \sum_{(w,c)\in D} \log \sigma(v_w \cdot v_c) + \sum_{(w,c)\in D'} \log \sigma(-v_w \cdot v_c)$$

For one sample:

$$\log \sigma(v_w \cdot v_c) + \sum_{i=1}^{k} \log \sigma(-v_w \cdot v_c)$$

Extra slides

Neural nets (from our graduate class with P. Tsaparas)

(Thanks to Philipp Koehn for the material borrowed from his slides)

INTRODUCTION TO NEURAL NETWORKS

Classification

- **Classification** is the task of ***learning a target function*** **f** that maps attribute set **x** to one of the predefined class labels **y**

categorical categorical continuous class

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

One of the attributes is the class attribute
In this case: Cheat

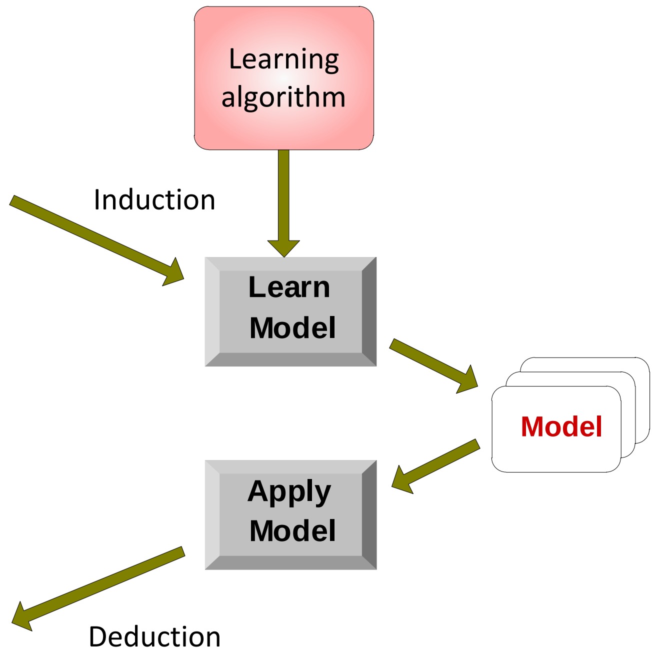
Two class labels (or classes): Yes (1), No (0)

Figure 4.2. Classification as the task of mapping an input attribute set $\mathbf{x}$ into its class label y.

Illustrating Classification Task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set



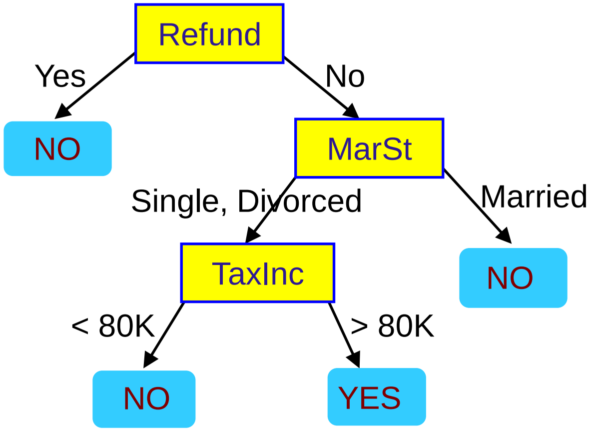
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set

Example of a Model

Tid	Refund	Marital Status	Taxable Income	Cheat
	categorical	categorical	continuous	class
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

Training Data



Model: Decision Tree

Classification in Networks

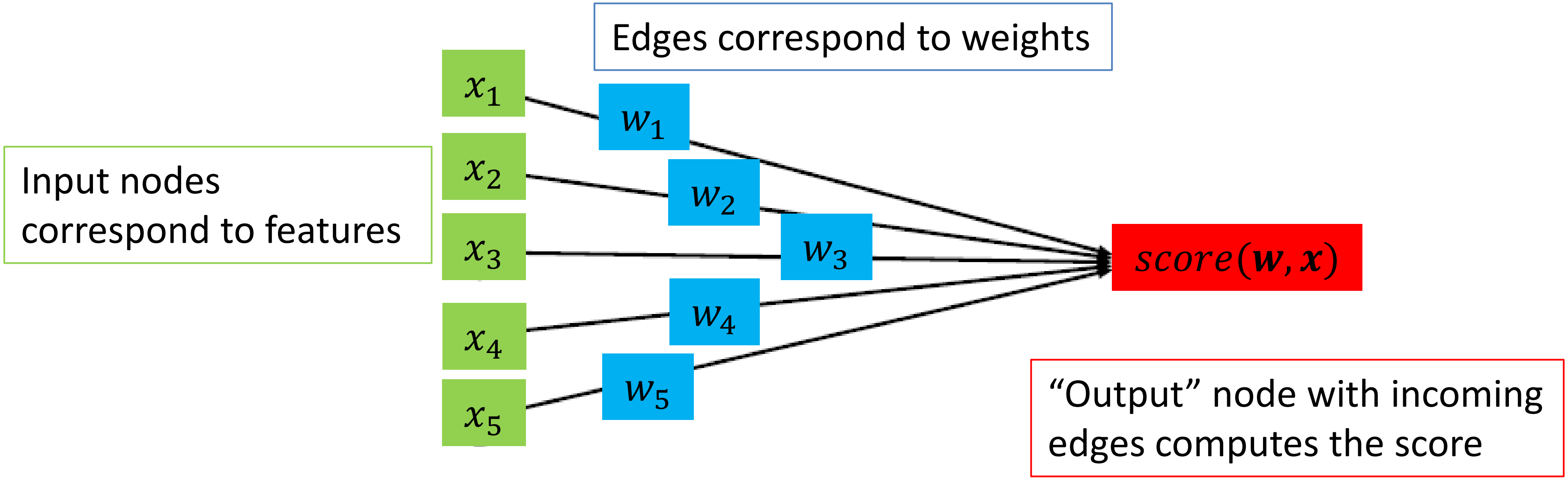
- There are various problems in network analysis that can be mapped to a classification problem:
  - Link prediction: Predict 0/1 for missing edges, whether they will appear or not in the future.
  - Node classification: Classify nodes as democrat-republican/spammers-legitimate/other categories
    - Use node features but also neighborhood and structural features
    - Label propagation
  - Edge classification: Classify edges according to type (professional/family relationships), or according to strength.
  - More…
- Recently all of this is done using Neural Networks.

Linear Classification

- A simple model for classification is to take a linear combination of the feature values and compute a score.
- Input: Feature vector $\boldsymbol{x} = (x_1, \ldots, x_n)$
- Model: Weights $\boldsymbol{w} = (w_1, \ldots, w_n)$
- Output: $score(\boldsymbol{w}, \boldsymbol{x}) = \sum_i w_i x_i$
- Make a decision depending on the output score.
  - E.g.: Decide "Yes" if $score(\boldsymbol{w}, \boldsymbol{x}) > 0$ and "No" if $score(\boldsymbol{w}, \boldsymbol{x}) < 0$

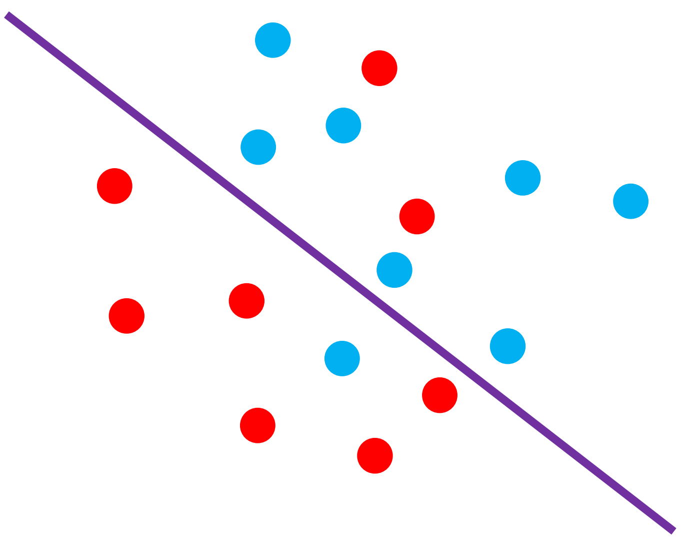
Linear Classification

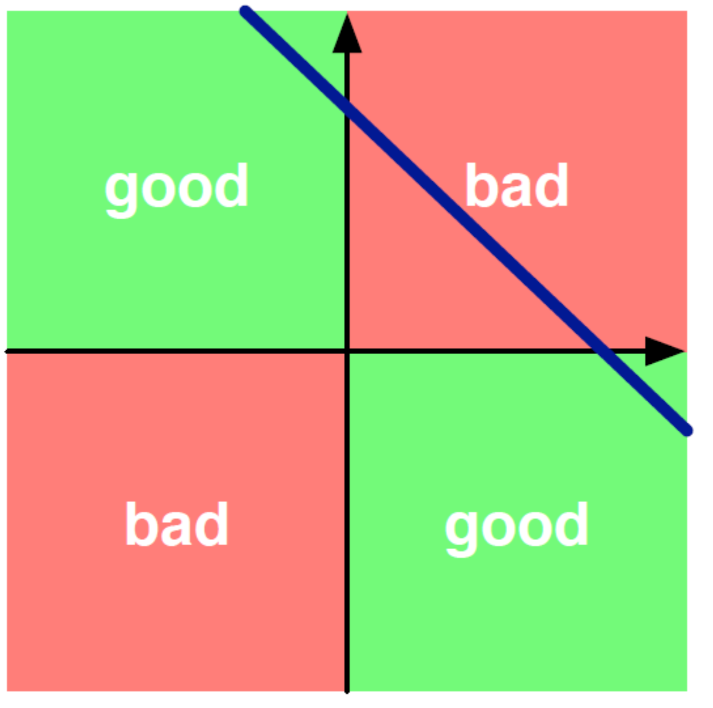
- We can represent this as a network



Linear models

- Linear models partition the space according to a hyperplane

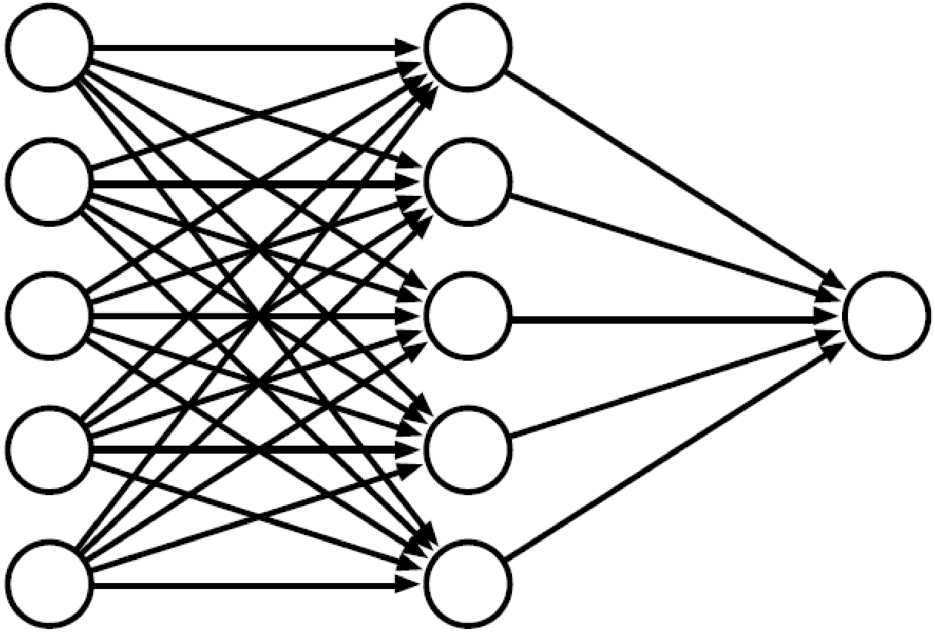




- But they cannot model everything

Multiple layers

- We can add more layers:
  - Each arrow has a weight
  - Nodes compute scores from incoming edges and give input to outgoing edges



Did we gain anything?

Non-linearity

- Instead of computing a linear combination

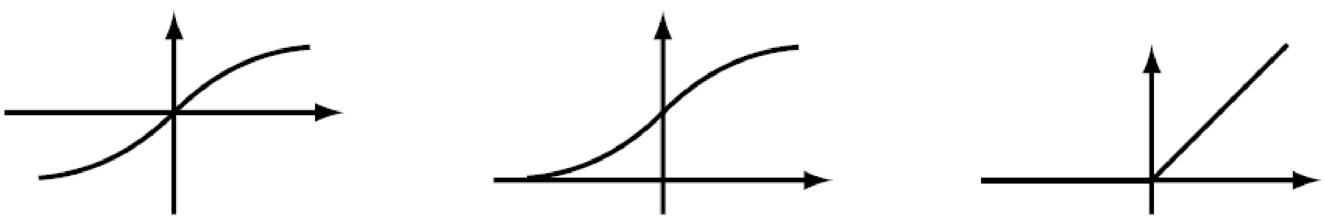
$$score(\boldsymbol{w}, \boldsymbol{x}) = \sum_i w_i x_i$$

- Apply a non-linear function on top:

$$score(\boldsymbol{w}, \boldsymbol{x}) = g\left(\sum_i w_i x_i\right)$$

- Popular functions:

$\tanh(x)$ $\quad$ $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$ $\quad$ $\text{relu}(x) = \max(0, x)$

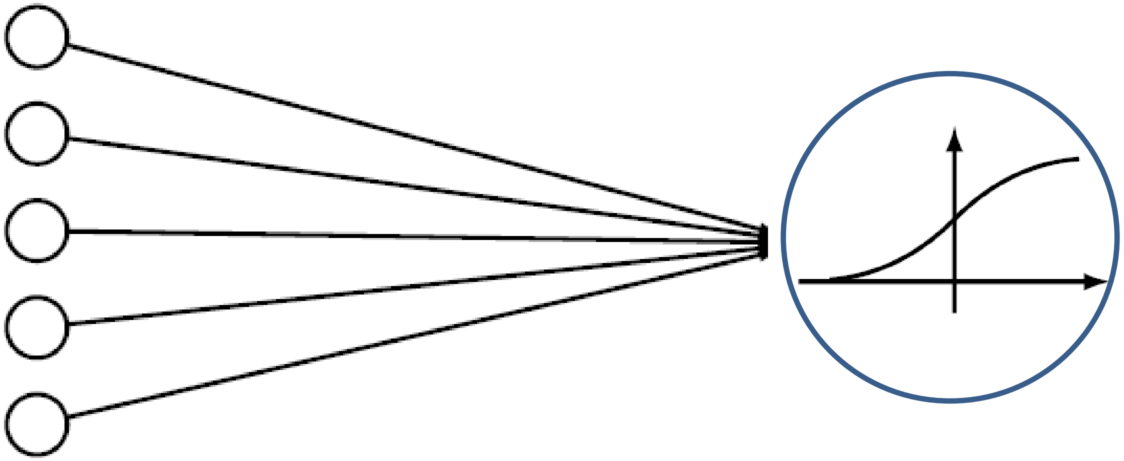


(sigmoid is also called the "logistic function")

These functions play the role of a soft “switch” (threshold function)

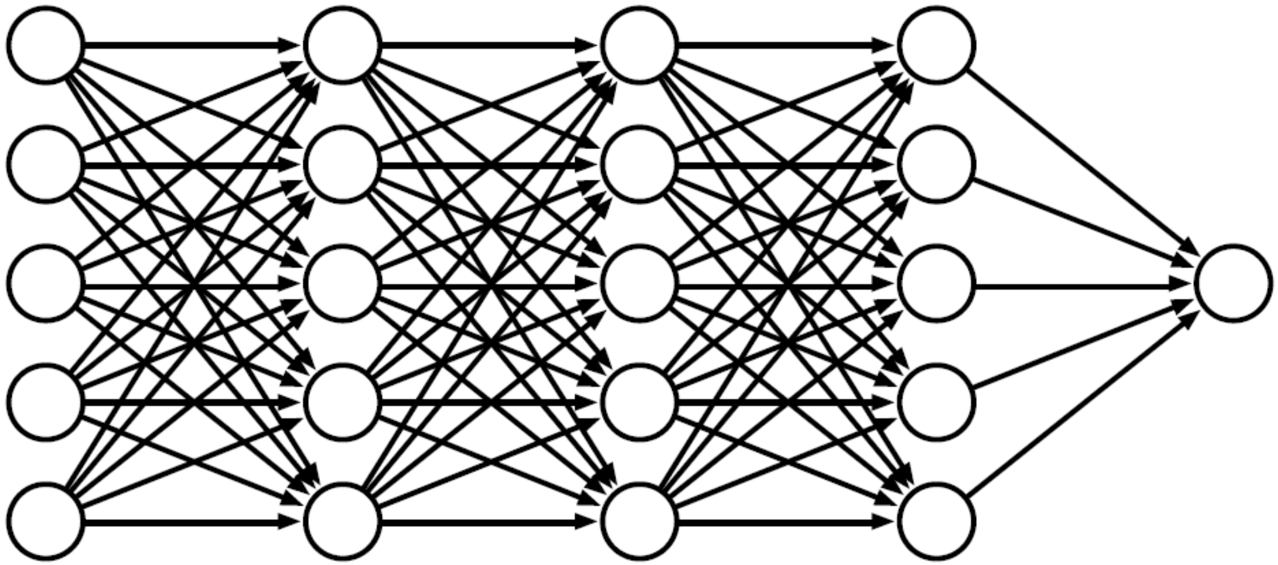
Side note

- Logistic regression classifier:
  - Single layer with a logistic function



Deep learning

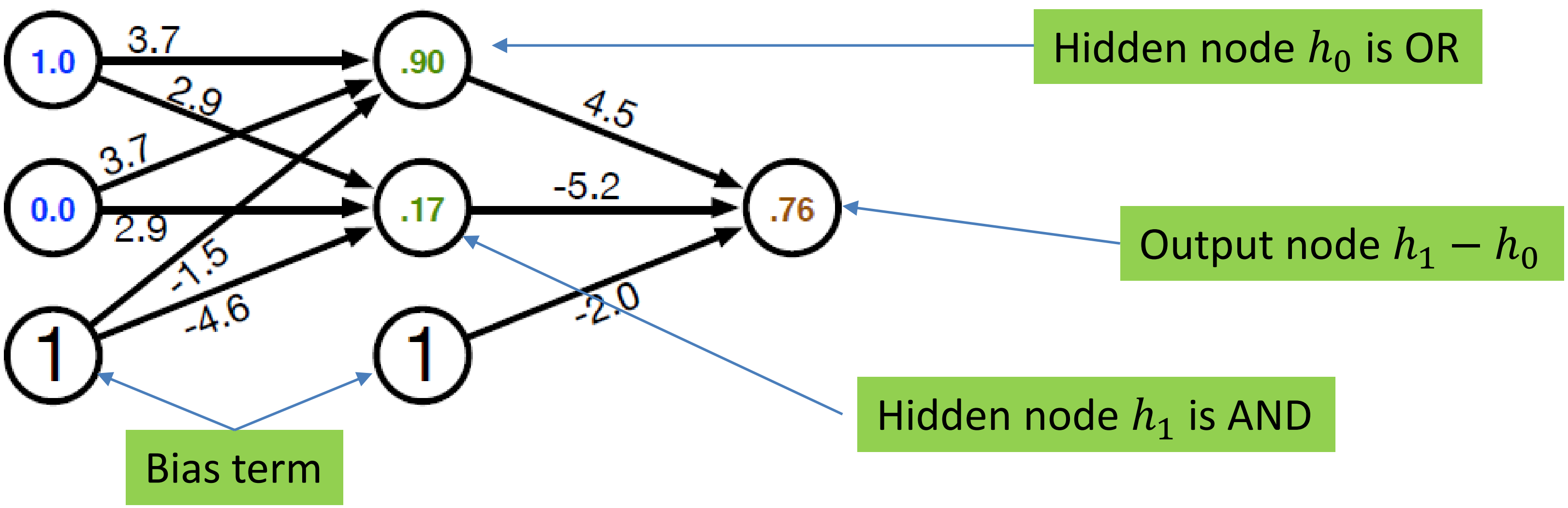
- Networks with multiple layers



- Each layer can be thought of as a processing step
- Multiple layers allow for the computation of more complex functions

Example

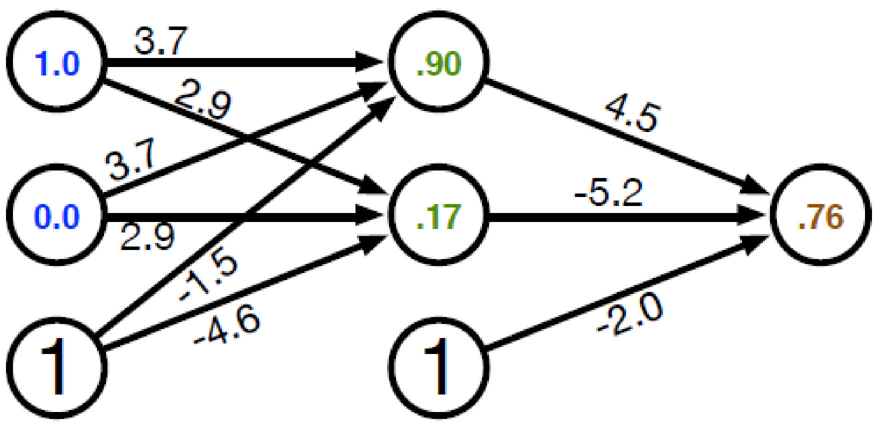
- A network that implements XOR



Input x_0	Input x_1	Hidden h_0	Hidden h_1	Output y_0
0	0	0.12	0.02	$0.18 \rightarrow 0$
0	1	0.88	0.27	$0.74 \rightarrow 1$
1	0	0.73	0.12	$0.74 \rightarrow 1$
1	1	0.99	0.73	$0.33 \rightarrow 0$

Error

- The computed value is 0.76 but the correct value is 1
  - There is an error in the computation

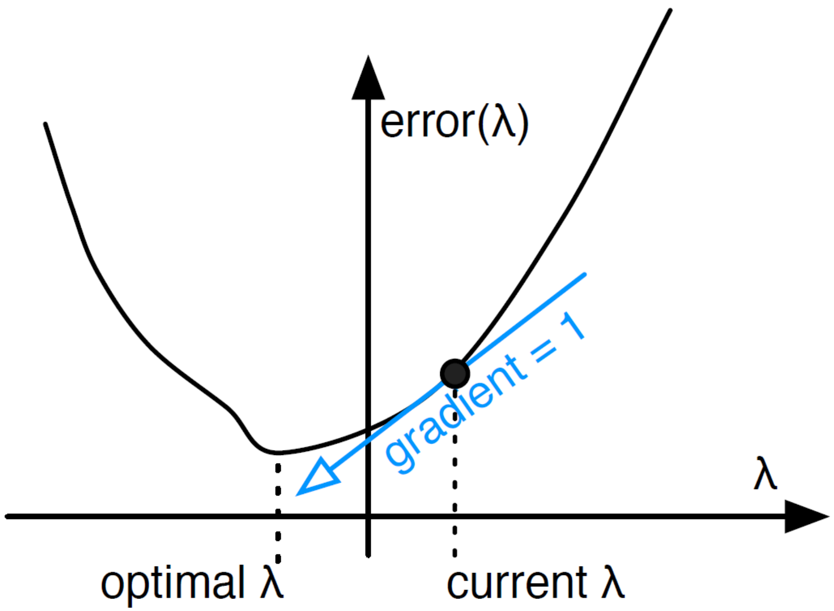


  - How do we set the weights so as to minimize this error?

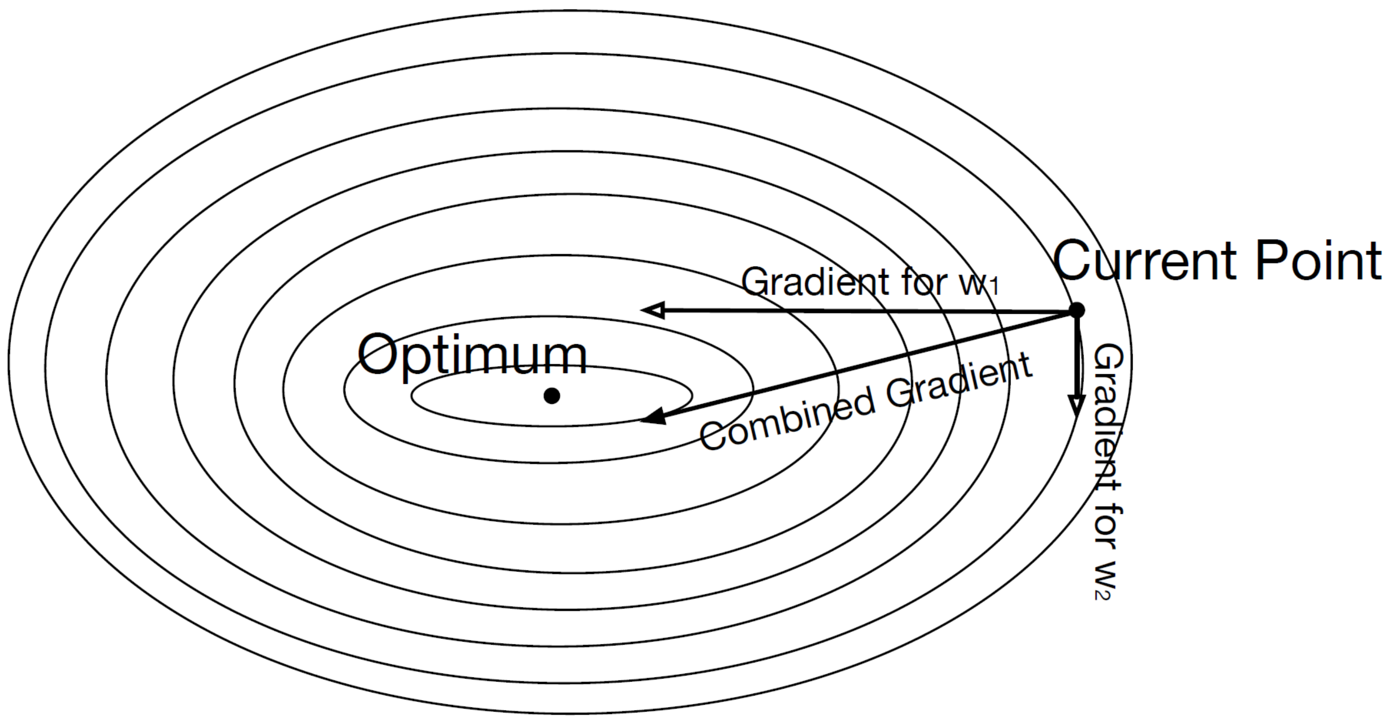
Gradient Descent

- The error is a function of the weights
- We want to find the weights that minimize the error
- Compute gradient: gives the direction to the minimum
- Adjust weights, moving at the direction of the gradient.

Gradient Descent

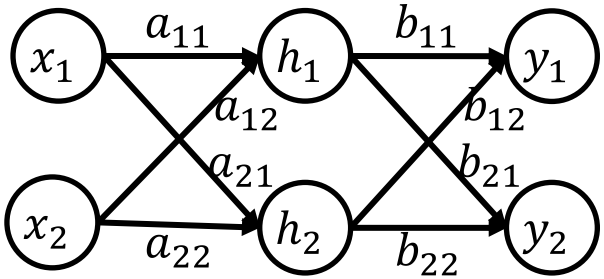


Gradient Descent



Backpropagation

- How can we compute the gradients? Backpropagation!
- Main idea:
  - Start from the final layer: compute the gradients for the weights of the final layer.
  - Use these gradients to compute the gradients of previous layers using the chain rule
  - Propagate the error backwards
- Backpropagation essentially is an application of the chain rule for differentiation.



Error: $E = \|y - t\|^2 = (y_1 - t_1)^2 + (y_2 - t_2)^2$

$$\frac{\partial E}{\partial b_{11}} = \frac{\partial E}{\partial s_{y_1}} \frac{\partial s_{y_1}}{\partial b_{11}} = \delta_{y_1} h_1$$

$$\delta_{y_1} = \frac{\partial E}{\partial s_{y_1}} = \frac{\partial E}{\partial y_1} \frac{\partial y_1}{\partial s_{y_1}} = 2(y_1 - t_1) g'(s_{y_1})$$

Notation:

Activation function: g

$s_{y_1} = b_{11} h_1 + b_{12} h_2 \, , y_1 = g(s_{y_1})$

$s_{y_2} = b_{21} h_1 + b_{22} h_2 \, , y_2 = g(s_{y_2})$

$s_{h_1} = a_{11} x_1 + a_{12} x_2 \, , h_1 = g(s_{h_1})$

$s_{h_2} = a_{21} x_1 + a_{22} x_2 \, , h_2 = g(s_{h_2})$

$$\frac{\partial E}{\partial b_{21}} = \delta_{y_2} h_1$$

$$\delta_{y_2} = \frac{\partial E}{\partial s_{y_2}} = 2(y_2 - t_2) g'(s_{y_2})$$

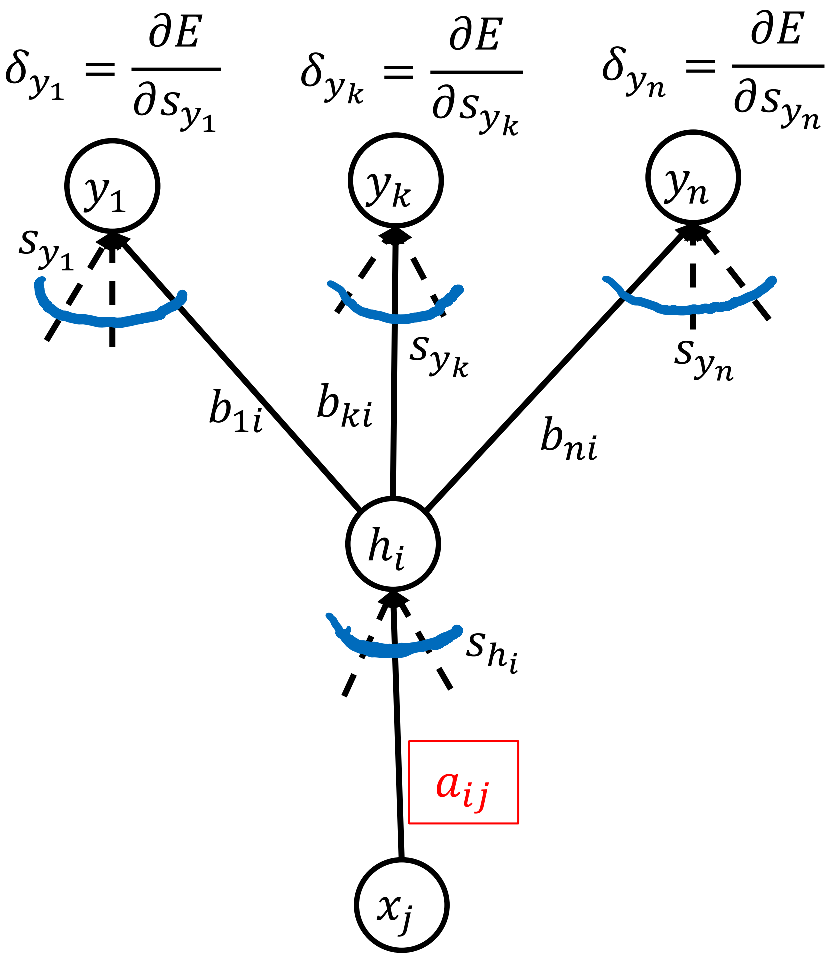
$$\frac{\partial E}{\partial b_{12}} = \delta_{y_1} h_2 \qquad \frac{\partial E}{\partial b_{22}} = \delta_{y_2} h_2$$

$$\frac{\partial E}{\partial a_{11}} = \frac{\partial E}{\partial s_{h_1}} \frac{\partial s_{h_1}}{\partial a_{11}} = \delta_{h_1} x_1 \qquad \frac{\partial E}{\partial a_{22}} = \frac{\partial E}{\partial s_{h_2}} \frac{\partial s_{h_2}}{\partial a_{22}} = \delta_{h_2} x_2 \qquad \frac{\partial E}{\partial a_{21}} = \delta_{h_1} x_2 \qquad \frac{\partial E}{\partial a_{12}} = \delta_{h_2} x_1$$

$$\delta_{h_1} = \frac{\partial E}{\partial s_{h_1}} = \frac{\partial E}{\partial h_1} \frac{\partial h_1}{\partial s_{h_1}} = \left(\frac{\partial E}{\partial s_{y_1}} \frac{\partial s_{y_1}}{\partial h_1} + \frac{\partial E}{\partial s_{y_2}} \frac{\partial s_{y_2}}{\partial h_1} \right) g'(s_{h_1}) = (\delta_{y_1} b_{11} + \delta_{y_2} b_{21}) g'(s_{h_1})$$

$$\delta_{h_2} = (\delta_{y_1} b_{12} + \delta_{y_2} b_{22}) g'(s_{h_2})$$

Backpropagation



$$\frac{\partial E}{\partial a_{ij}} = \sum_{k=1}^{n} \delta_{y_k} b_{ki}\, g'(s_{h_i}) x_j$$

For the sigmoid function:

$$g(x) = \frac{1}{1 + e^{-x}}$$

The derivative is:

$$g'(x) = g(x)(1 - g(x))$$

This makes it easy to compute it. We have:

$$g'(s_{h_i}) = h_i(1 - h_i)$$

Stochastic gradient descent

- Ideally the loss should be the average loss over all training data.
- We would need to compute the loss for all training data every time we update the gradients.
  - However, this is expensive.
- Stochastic gradient descent: Consider one input point at the time. Each point is considered only once.
- Intermediate solution: Use mini-batches of data points.

End of extra slides