Word Embeddings Tutorial

HILA GONEN
PHD STUDENT AT YOAV GOLDBERG'S LAB
BAR ILAN UNIVERSITY
5/3/18

Outline

- NLP Intro
- Word representations and word embeddings
- Word2vec models
- Visualizing word embeddings
- Word2vec in Hebrew
- Similarity
- Analogies
- Evaluation
- A simple classification example

NLP - Natural Language Processing

NLP is the field that includes **understanding** / **processing** / **analyzing** / **generating** natural languages

We aim to create applicative models that perform as similar as possible to humans

NLP

Applications in NLP:

- Translation
- Information Extraction
- Summarization
- Parsing
- Question Answering
- Sentiment Analysis
- Text Classification

And many more…

NLP challenges

This field encounters numerous challenges:

- Polysemy
- Syntactic ambiguity
- Variability
- Co-reference resolution
- Lack of data / huge amounts of data

NLP challenges - Polysemy

Book

Verb: **Book** a flight

Noun: He says it’s a very good **book**

Bank

The edge of a river: He was strolling near the river **bank**

A financial institution: He works at the **bank**

Solution

An answer to a problem: Work out the **solution** in your head

From Chemistry: Heat the **solution** to 75° Celsius

NLP challenges – Polysemy

Kids make nutritious snacks

- Kids, when cooked well, can make nutritious snacks

Kids make nutritious snacks

- Kids know how to prepare nutritious snacks

NLP challenges – Syntactic Ambiguity

12 on their way to cruise among dead in plane crash

12 on their way to cruise among dead in plane crash

same words – different meanings

NLP challenges – Syntactic Ambiguity

The cotton clothing is usually made of grows in Mississippi

The cotton clothing is usually made of grows in Mississippi

same words – different meanings

NLP challenges – Syntactic Ambiguity

Fat people eat accumulates

Fat people eat accumulates

same words – different meanings

NLP challenges – Variability

They allowed him to…

They let him …

He was allowed to…

He was permitted to…

Different words – same meaning

NLP challenges – Co-Reference Resolution

Rachel had to wait for Dan because he said he wanted her advice.

This is a simple case…

There are more complex ones.

Dan called Bob to tell him about his surprising experience last week: "you won't believe it, I myself could not believe it".

NLP challenges – Data-related issues

A lot of data

In some cases, we deal with huge amounts of data

Need to come up with models that can process a lot of data efficiently

Lack of data

Many problems in NLP suffer from lack of data:

- Non-standard platforms (code-switching)
- Expensive annotation (word-sense disambiguation, named-entity recognition)

Need to use methods to overcome this challenge (semi-supervised learning, multi-task learning…)

Representation

We can represent objects in different hierarchy levels:

- Documents
- Sentences
- Phrases
- Words

We want the representation to be interpretable and easy-to-use

Vector representation meets those requirements

We will focus on word representation

The Distributional Hypothesis

The Distributional Hypothesis:

- words that occur in the same contexts tend to have similar meanings
  (Harris, 1954)
- "You shall know a word by the company it keeps" (Firth, 1957)

Examples:

- Cucumber, sauce, pizza, ketchup
- Soundtrack, lyrics, sang, duet

tomato

song

Vector Representation

We can define a word by a vector of counts over contexts, For Example:

	song	cucumber	meal	black
tomato	**0**	**6**	**5**	**0**
book	2	0	2	3
pizza	**0**	**2**	**4**	**1**

- Each word is associated with a vector of dimension |V| (the size of the vocabulary)
- We expect similar words to have similar vectors
- Given the vectors of two words, we can determine their similarity (more about that later)

We can use different granularities of contexts: documents, sentences, phrases, n-grams

Vector Representation

Raw counts are problematic:

- frequent words will characterize most words -> not informative

Except from raw counts, we can use other functions:

- TF-IDF (for term (t) – document (d)):

$$TF-IDF(t,d) = \frac{count(t,d)}{|d|} \cdot \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

D – set of all documents

- Pointwise Mutual Information:

$$PMI = \log \frac{p(x,y)}{p(x)p(y)}$$

From Sparse to Dense

These vectors are:

- huge – each of dimension |V| (the size of the vocabulary ~ $100K +$)
- sparse – most entries will be 0

We want our vectors to be small and dense, two options:

1. Use a reduction algorithm such as SVD over a matrix of sparse vectors
2. Learn low-dimensional word vectors directly -

   usually referred as “word embeddings”

We will focus on the second option

Word Embeddings

Each word in the vocabulary is represented by a low dimensional vector ($\sim 300d$)

All words are embedded into the same space

Similar words have similar vectors

(= their vectors are close to each other in the vector space)

Word embeddings are successfully used for various NLP applications

Uses of word embeddings

Word embeddings are successfully used for various NLP applications (usually simply for initialization)

- Semantic similarity
- Word sense Disambiguation
- Semantic Role Labeling
- Named entity Recognition
- Summarization
- Question Answering
- Textual Entailment
- Coreference Resolution
- Sentiment analysis
- etc.

Word2Vec

Models for efficiently creating word embeddings

Remember: our assumption is that similar words appear with similar context

Intuition: two words that share similar contexts are associated with vectors that are close to each other in the vector space

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, 2013. *Efficient estimation of word representations in vector space*. arXiv preprint arXiv:1301.3781.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean, 2013. *Distributed representations of words and phrases and their compositionality*. In Advances in neural information processing systems.

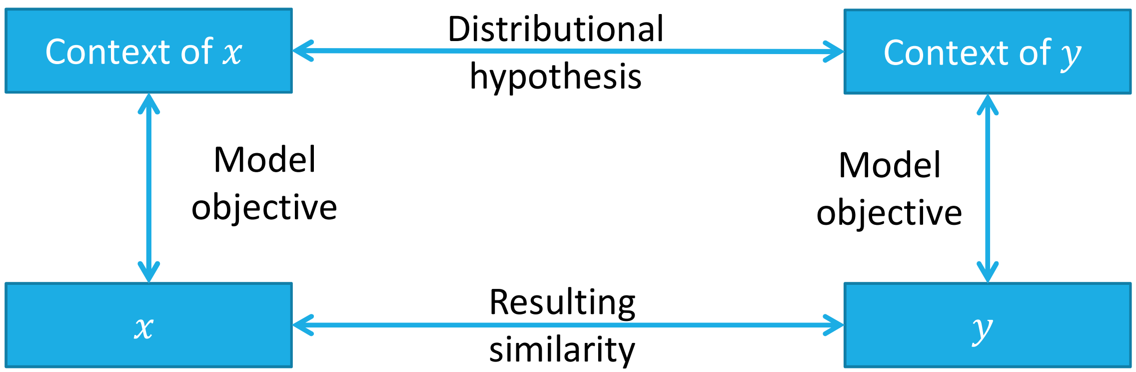
Word2Vec

Models for efficiently creating word embeddings

Remember: our assumption is that similar words appear with similar context

Intuition: two words that share similar contexts are associated with vectors that are close to each other in the vector space

Let x and y be similar words



Word2Vec

The input: one-hot vectors

- bananas: $(1,0,0,0)$
- monkey: $(0,1,0,0)$
- likes: $(0,0,1,0)$
- every: $(0,0,0,1)$

vocabulary size
$|V| = 4$

We are going to look at pairs of neighboring words:

Every monkey likes bananas $\longrightarrow$

$(every, monkey)$

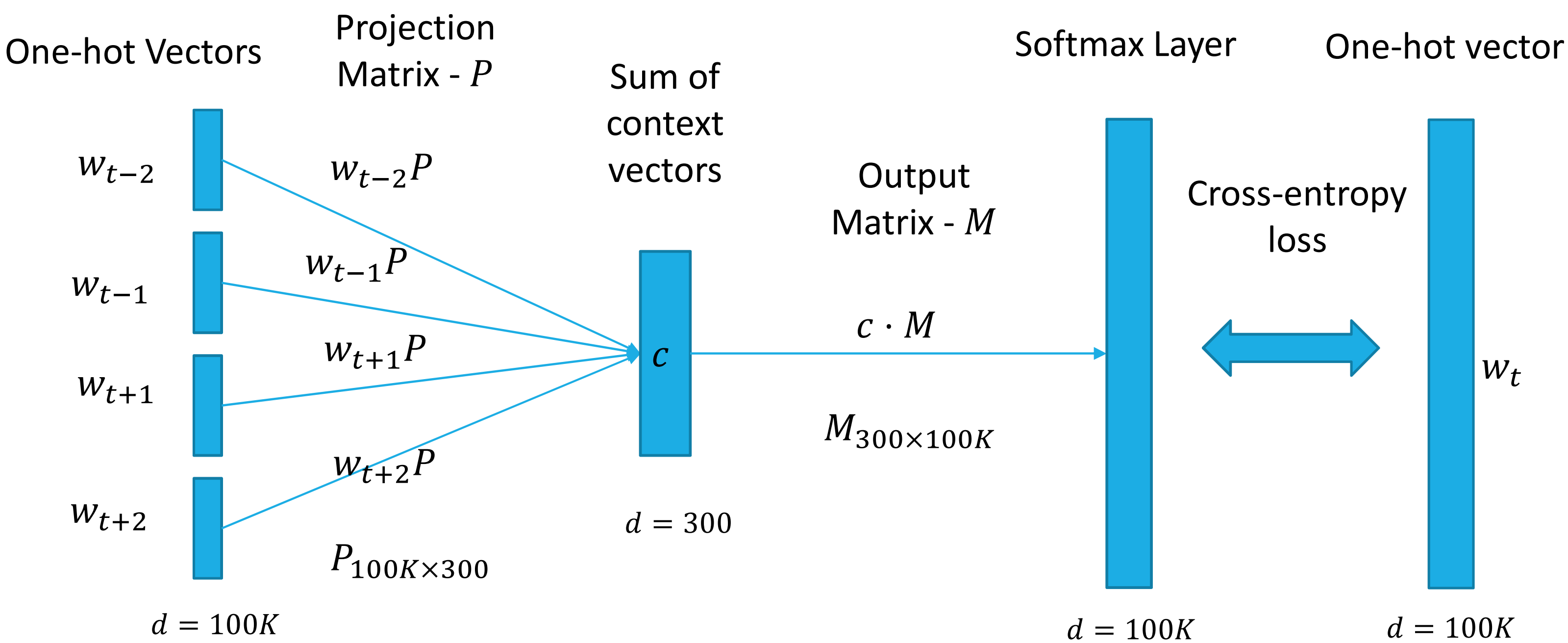
$(likes, monkey)$

$(bananas, monkey)$

CBOW – high level

The resulting projection matrix P is the embedding matrix

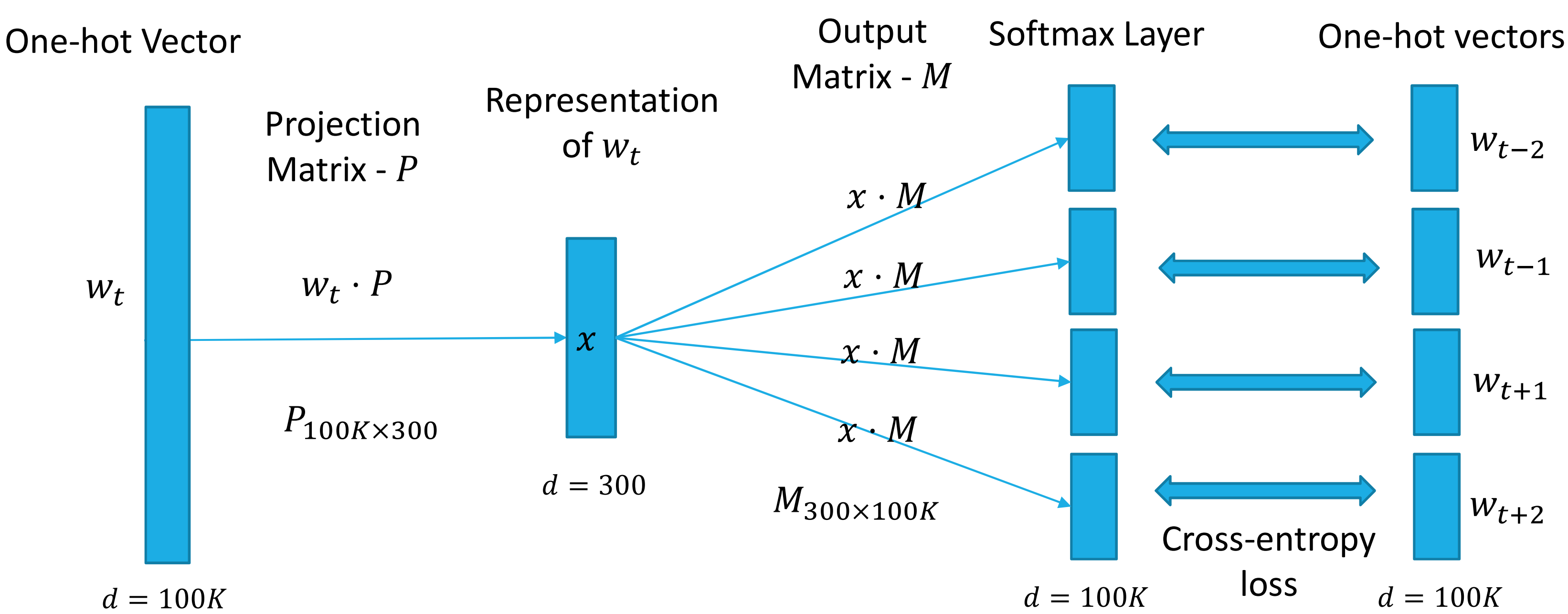
Goal: Predict the middle word given the words of the context



Skip-gram – high level

The resulting projection matrix P is the embedding matrix

Goal: Predict the context words given the middle word



Skip-gram – details

Vector representations will be useful for predicting the surrounding words.

Formally:

Given a sequence of training words $w_1, w_2, ...w_T$, the objective of the Skip-gram model is to maximize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

The basic Skip-gram formulation defines $p(w_{t+j}|w_t)$ using the softmax function:

$$p(w_{t+j}|w_t) = \frac{\exp(v'_{w_{t+j}} v_{w_t})}{\sum_{i=1}^{T}\exp(v'_{w_i} v_{w_t})}$$

v - input vector representations

v' - output vector representations

Negative Sampling

Recall that for Skip-gram we want to maximize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

Which is equivalent to minimizing the **cross-entropy loss**:

$$L = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log\frac{\exp(v'_{w_{t+j}}v_{w_t})}{\sum_{i=1}^{T}\exp(v'_{w_i}v_{w_t})}$$

This is extremely computational-expensive, as we need to update all the parameters of the model for each training example…

Negative Sampling

When looking at the loss obtained from a single training example, we get:

$$-\log p(w_{t+j}|w_t) = -\log \frac{\exp(v'_{w_{t+j}} v_{w_t})}{\sum_{i=1}^{T} \exp(v'_{w_i} v_{w_t})} = -v'_{w_{t+j}} v_{w_t} + \log \sum_{i=1}^{T} \exp(v'_{w_i} v_{w_t})$$

"positive" pair "negative" pair

When using negative sampling, instead of going through all the words in the vocabulary for negative pairs, we sample a modest amount of k words (around 5-20). The exact objective used:

$$\log \sigma(v'_{w_{t+j}} v_{w_t}) + \sum_{1=1}^{k} \log \sigma(-v'_{w_i} v_{w_t})$$

Replaces the term: $\log p(w_{t+j}|w_t)$ for each word in the training

$$\sigma(x) = \frac{1}{1+\exp(-x)}$$

Context Sampling

We want to give more weight to words closer to our target word

For a given window size C, we sample R in [1, C] and try to predict only R words before and after our target word

For each word in the training we need to perform 2*R word classifications

(R is not fixed)

Subsampling of Frequent Words

In order to eliminate the negative effect of very frequent words such as “in”, “the” etc. (that are usually not informative), a simple subsampling approach is used:

Each word w_i in the training set is discarded with probability: $P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$

This way frequent words are discarded more often

This method improves the training speed and makes the word representations significantly more accurate

Phrases

These models are unable to represent phrases that are not compositions of the individual words

"New York" != "New" + "York"

"Boston Globe" != "Boston" + "Globe"

The extension is simple:

- Find words that appear frequently together, and infrequently in other contexts
  - phrases are formed based on the unigram and bigram counts

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

  - The bigrams with score above the chosen threshold are then used as phrases
  - "New York Times" will be replaced with a unique token, "this is" will remain unchanged
- Train word2vec as usual

Use word2vec package

Using this package is extremely simple:

- Download the code from Mikolov’s git repository:
- https://github.com/tmikolov/word2vec
- Compile the package
- Download the default corpus (wget http://mattmahoney.net/dc/text8.zip) or another corpus of your choice
- Train the model using the desired parameters

Jupyter: code for downloading, compiling, and training

Importance of Parameters – window size

Word: **walk**

Window size = 3

Word	Cosine distance
go	0.488083
snipe	0.464912
shoot	0.456677
fly	0.449722
sit	0.449678
pass	0.442459
climbs	0.440931
walked	0.436502
ride	0.434034
stumble	0.426750
bounce	0.425577
travelling	0.419419
walking	0.412107
walks	0.410949
trot	0.410418
leaping	0.406744
sneak	0.401918
climb	0.399793
move	0.396715
wait	0.394463
going	0.391639
shouted	0.388382
roam	0.388073
thrown	0.384087
get	0.383894

Window size = 30

Word	Cosine distance
walking	0.486317
walked	0.430764
walks	0.406772
stairs	0.401518
go	0.399274
sidewalk	0.385786
stand	0.380480
cortege	0.371033
wheelchair	0.362877
strapped	0.360179
hollywood	0.356544
carousel	0.356187
grabs	0.356007
swim	0.355027
breathe	0.354314
tripped	0.352899
cheer	0.352477
moving	0.350943
inductees	0.347791
walkway	0.347164
shout	0.346229
pounding	0.340554
blvd	0.339121
crowd	0.338731
levada	0.334899

Importance of Parameters – iterations

Word: **walk**

No. of iterations = 1

Word	Cosine distance
walking	0.851438
walks	0.846485
bat	0.843796
ride	0.830734
crowd	0.821692
quiet	0.812538
spot	0.802777
steal	0.787917
door	0.787571
doors	0.786485
bed	0.773686
dinner	0.772160
shadow	0.769573
luck	0.768221
baby	0.767862
shoot	0.765968
walked	0.765739
sitting	0.765394
shirt	0.759116
rides	0.759047
watching	0.755140
watch	0.750808
gehrig	0.741494
shoots	0.740971
looking	0.740904

No. of iterations = 100

Word	Cosine distance
walked	0.483473
ride	0.470925
walks	0.470889
stand	0.449993
walking	0.449071
go	0.430172
shoot	0.421110
get	0.404258
move	0.403757
live	0.403347
fly	0.400929
climbs	0.396346
throw	0.391768
climb	0.384038
wiggle	0.380892
thrown	0.380426
pull	0.375478
goes	0.375406
moving	0.374447
pass	0.372463
conversing	0.364413
sit	0.362765
crowd	0.361651
kiss	0.359883
stay	0.357015

Importance of Parameters – dimensions

Word: **walk**

No. of dimensions = 5

Word	Cosine distance
catcher	0.998074
shirt	0.996589
lechuck	0.995313
bullseye	0.994644
bowler	0.994381
punter	0.993154
lovell	0.992815
heels	0.992255
whip	0.992085
outfit	0.992047
tore	0.991924
steals	0.991524
guybrush	0.991166
gigs	0.990291
hanging	0.990201
burns	0.990043
backing	0.989966
orser	0.989960
torch	0.989747
beat	0.989435
showdown	0.989381
feat	0.989242
cheers	0.988951
clad	0.988646
lunch	0.988326

No. of dimensions = 1000

Word	Cosine distance
walks	0.304954
walked	0.303322
snipe	0.287221
walking	0.272690
ride	0.266770
canter	0.251025
bandleaders	0.246454
climbs	0.233725
catapulted	0.230075
climb	0.229263
trot	0.228362
shouted	0.227306
stand	0.223288
seagulls	0.221745
fly	0.216602
fences	0.216366
lifts	0.215402
pray	0.214977
paws	0.214865
bounces	0.214449
shoot	0.213457
grabs	0.212018
walkway	0.211136
swim	0.209120
tumble	0.207765

How does the file usually look like

Word embedding files in readable format usually have a row for each word in the vocabulary

In each row, the specific word is followed by the values of the respected vector

Possibly some additional information in the first rows

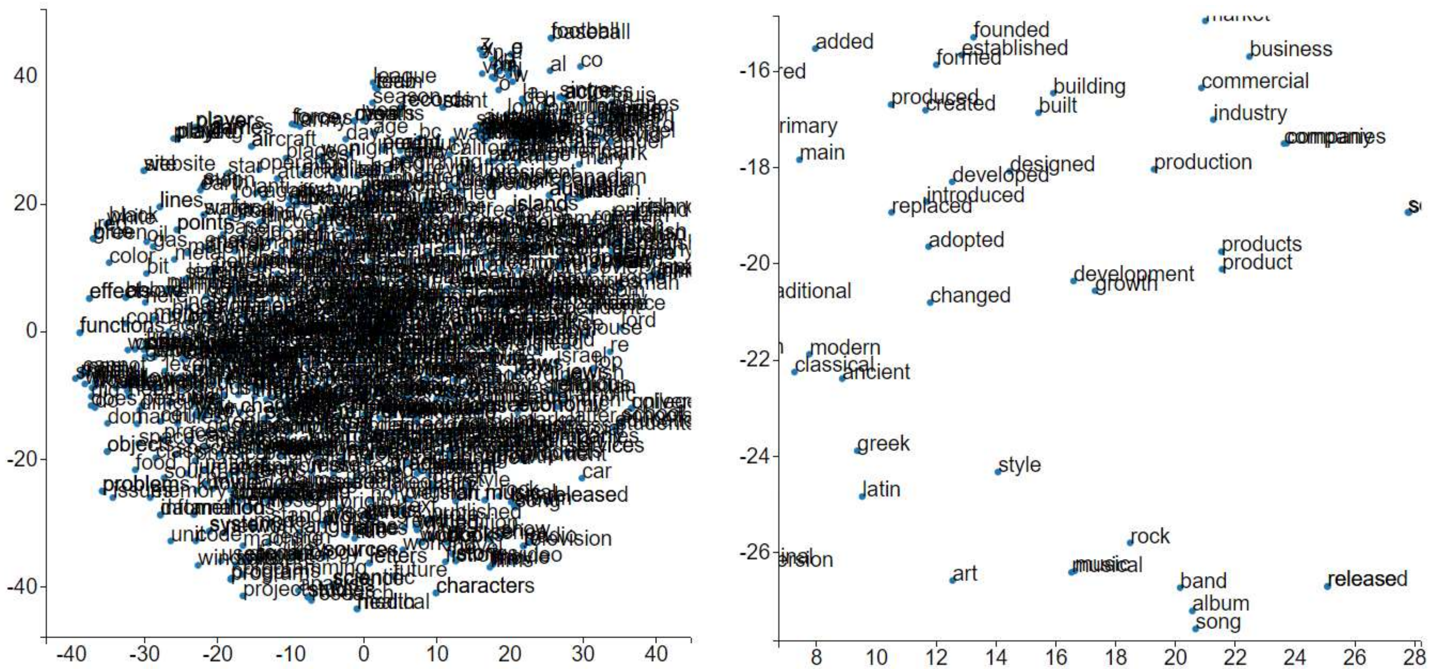
Visualizing word embeddings

Using tSNE

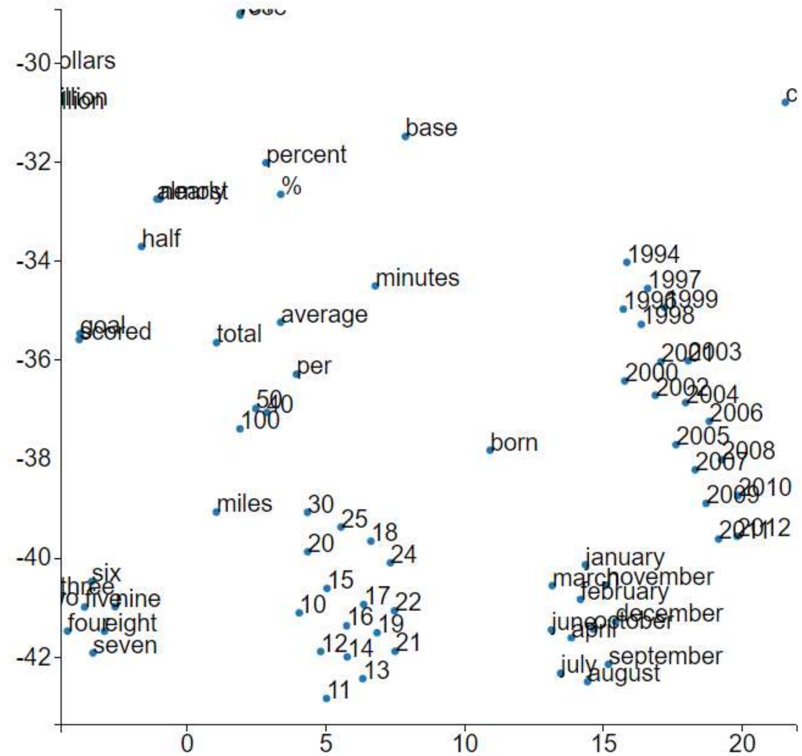
- *Visualizing Data using t-SNE*, Maaten and Hinton, 2008

Jupyter: Loading and Visualizing word vectors

Visualizing word embeddings– Word2Vec



Visualizing word embeddings - Glove



Wevi: word embedding visual inspector

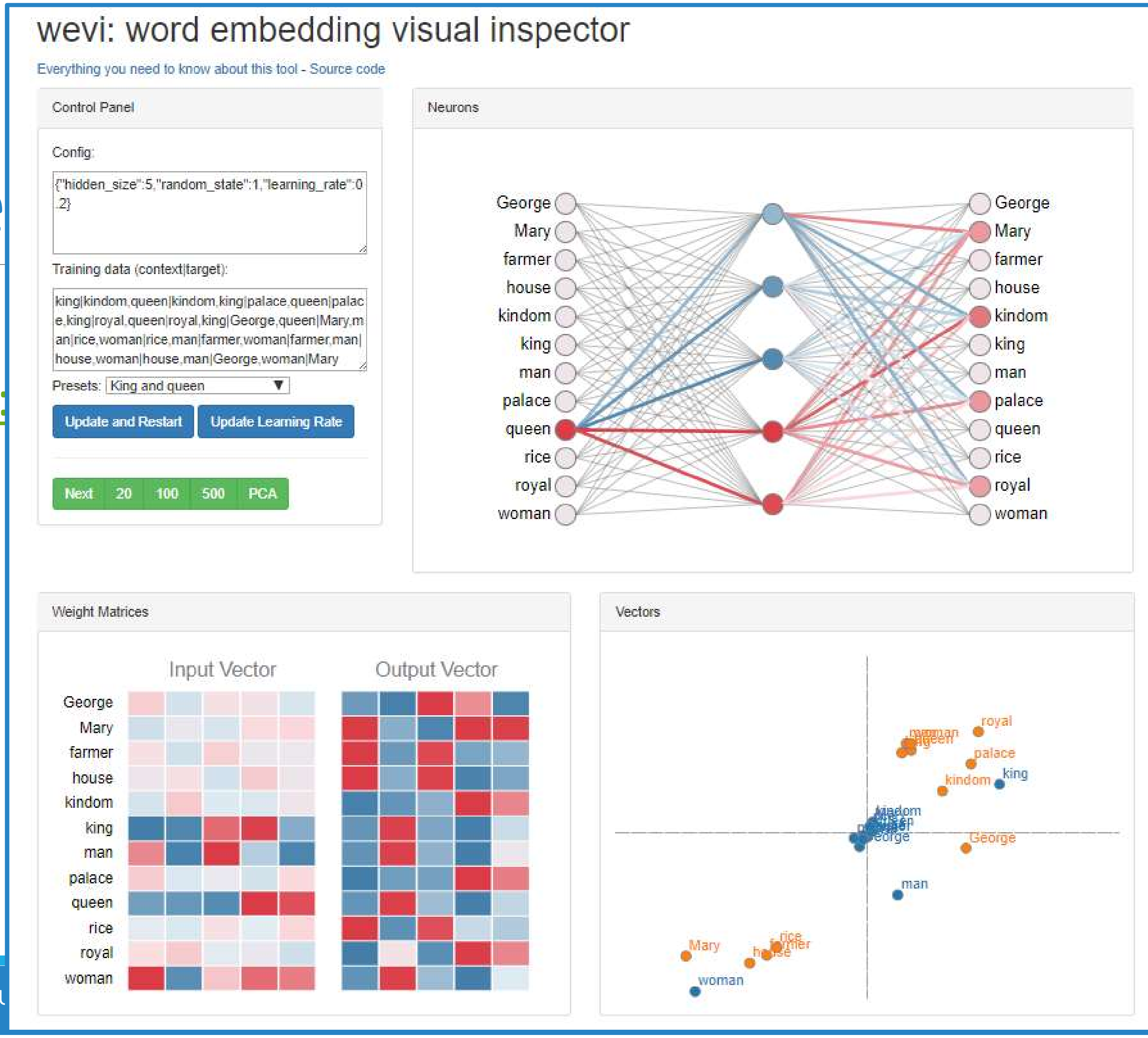
A tool that visualizes the basic working mechanism of word2vec

https://ronxin.github.io/wevi/

We... ...ctor

A too... ...2vec

https:...



Word2Vec - Hebrew

Link: http://u.cs.biu.ac.il/~yogo/tw2v/similar/ (By Ron Shemesh and Yoav Goldberg)

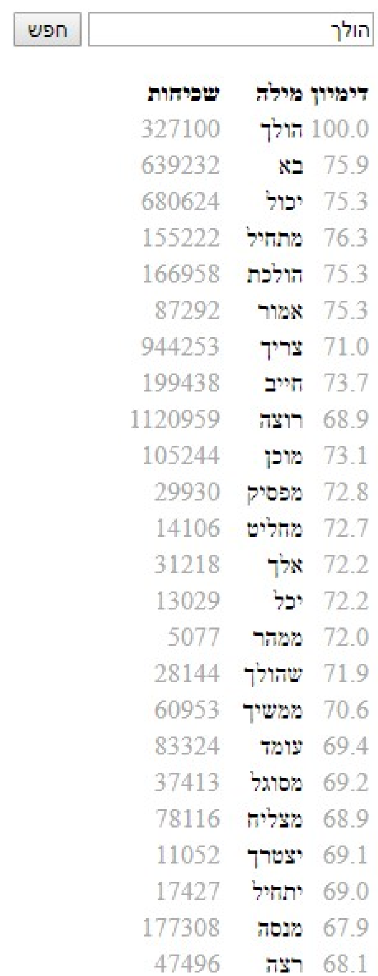
Based on tweets in Hebrew

Wor … eb

Link: http … /tw2v … nesh

Based on

דימיון	מילה	שכיחות
100.0		16065
94.7		14738
93.5	☺	20020
92.7		16020
90.2		62507
87.1		68331
86.6		5438
85.8		3556
82.7		4577
81.5		44223
80.8		30838
79.9	☺	19796
79.6		24873
78.8		2327
76.5		13376
76.7		1236
76.6		2197
76.0		3758
75.4		749
75.3	^_^	2258
74.5		13669



הולך | חפש

דימיון	מילה	שכיחות
100.0	הולך	327100
75.9	בא	639232
75.3	יכול	680624
76.3	מתחיל	155222
75.3	הולכת	166958
75.3	אמור	87292
71.0	צריך	944253
73.7	חייב	199438
68.9	רוצה	1120959
73.1	מוכן	105244
72.8	מפסיק	29930
72.7	מחליט	14106
72.2	אלך	31218
72.2	יכל	13029
72.0	ממהר	5077
71.9	שהולך	28144
70.6	ממשיך	60953
69.4	עומד	83324
69.2	מסוגל	37413
68.9	מצליח	78116
69.1	יצטרך	11052
69.0	יתחיל	17427
67.9	מנסה	177308
68.1	רצה	47496

שפה | חפש

דימיון	מילה	שכיחות
100.0	שפה	20864
78.1	הגיה	125
74.3	הגייה	255
72.9	אסכולה	288
71.7	אינטואיציה	292
71.2	דיסציפלינה	44
70.2	לוגיקה	2287
70.6	משפה	261
68.7	תיאוריה	3938
69.3	תפישה	496
69.2	מיתולוגיה	391
68.9	גנטיקה	1099
69.1	ורסיה	187
69.0	כשפה	450
68.9	מתודולוגיה	179
67.6	פלטפורמה	3684
68.2	התחכמות	589
66.2	הגדרה	9034
65.3	שיטה	11916
68.1	סכמה	53
67.7	פרקטיקה	1579
67.9	טרמינולוגיה	582
66.9	תפיסה	4492
67.9	עגה	50

Word2Vec - Hebrew

Ok… Nice.

But:

What about יעד vs. מטרה which are synonyms?

Noun genders dramatically affect results –

We do not want that, or at least not for arbitrary gender

Word2Ve

Ok… Nice.

But:

What about יעד vs. … onyı

Noun genders drar … ; –

We do not want th … arbi

יעד חפש

דימיון	מילה	שכיחות
100.0	יעד	8136
70.6	כיעד	535
65.2	קריטריון	1476
63.6	נכס	6366
61.6	מודל	10784
61.6	ויעד	158
60.5	תרחיש	3626
60.0	היעד	11644
59.2	הישג	17600
59.4	כסיכון	46
58.7	סטנדרט	2528
57.1	פתרון	38259
56.8	תמריץ	1470
51.3	מקום	278883
56.2	קרטריון	37
56.0	ובינלאומי	95
56.0	ואקספקלוסיבי	13
55.8	ייתרון	237
55.3	שטח	23446
55.1	רף	5116
54.6	יתרון	12953
54.1	מהלך	24049
54.2	פוטנציאל	11696
53.9	כיוון	31019

מטרה חפש

דימיון	מילה	שכיחות
100.0	מטרה	19358
74.5	שאיפה	2658
71.9	משימה	9828
69.1	הזדמנות	61495
66.2	סיבה	105895
70.5	אלטרנטיבה	8805
70.7	נוסחה	1613
70.0	שיטה	11916
70.5	נוסחא	237
70.0	אסטרטגיה	4714
67.9	אג'נדה	7000
67.4	יישות	681
67.4	תפישה	496
67.0	פלטפורמה	3684
65.1	נקודה	39317
65.6	גישה	19288
66.4	דוקטרינה	196
66.0	אסטרטגייה	44
64.9	בחירה	21334
65.6	תפיסה	4492
65.6	חלופה	3168
65.3	הגדרה	9034
62.9	אפשרות	39166
64.6	ישות	1900

Word2Vec - Hebrew

Another example:

בצרה

בצל

Prefixes and suffixes are not always handled correctly

Also, not always clear what the wanted behavior is

Word2V… ew

Another examp…

בצרה

בצל

Prefixes and su… ys handlec

Also, not alway… anted beh

בצל חפש

דימיון	מילה	שכיחות
100.0	בצל	23556
79.8	ובצל	1574
63.3	וסלרי	103
56.8	מרק	27207
61.7	ובזיליקום	314
61.5	ופטרוזיליה	286
61.4	מגורד	183
59.8	ברוטב	5301
60.7	וזוקיני	45
60.2	בחרדל	127
59.9	וכוסברה	335
60.0	וזעתר	185
59.1	סלק	3523
58.2	רוטב	7020
59.1	ולימון	1723
58.9	קונפי	298
58.9	וברוקולי	203
58.6	וסלמון	196
58.6	ואורגנו	77
58.6	ואספרגוס	88
58.3	ופרמזן	147
58.1	כתוש	270
58.1	וסלק	235
58.0	פורטבלו	135

בצרה חפש

דימיון	מילה	שכיחות
100.0	בצרה	1624
60.6	במצוקה	4622
50.2	בבעיה	8617
44.5	בסביבה	13472
47.2	בצרות	3026
46.2	בהכרה	3730
42.7	בסכנה	9296
46.1	ומרעננה	85
46.0	בקרבתם	181
45.8	באחווה	184
45.5	בקירבה	97
45.5	ונטועה	16
45.5	משמאלם	12
44.8	בחובות	1742
45.1	בטובתם	96
44.1	במיצוט	1852
44.2	בגלות	1610
43.0	מאחוריך	4082
43.1	בבועה	2503
43.7	בבוטניקה	27
43.5	בפרנויה	150
43.2	מאופקים	775
43.5	ובשביה	53
43.3	ובריאותו	61

Word2Vec – Hebrew

Word embeddings include inherent biases, as a result of biases in the corpus

(not only in Hebrew…)

For example, רופא vs. רופאה

Word2Ve… w

Word embeddings … biases, a… s in the corpus

(not only in Hebre…

For example, רופא…

חפש | רופאה

דימיון	מילה	שכיחות
100.0	רופאה	4307
74.4	הרופאה	3064
73.2	מטופלת	2407
72.3	פציינטית	64
72.1	וטרינרית	457
71.9	קוסמטיקאית	1242
71.6	עו"סית	80
71.3	כאחות	453
70.9	פקידה	2470
69.5	תלמידה	3860
69.5	מיילדת	440
69.1	מדריכה	1748
69.1	רוקחת	304
68.6	מדענית	530
68.3	קרדיולוגית	43
67.9	יולדת	2529
67.5	שיננית	1237
67.4	סטודנטית	5575
67.2	גננת	4402
67.0	פיליפינית	749
66.7	הוטרינרית	329
66.0	אחות	25801
66.5	פרופסורית	168
61.3	אישה	182017

חפש | רופא

דימיון	מילה	שכיחות
100.0	רופא	31223
85.1	אורתופד	368
82.0	כירורג	511
79.4	פסיכיאטר	3270
78.0	נוירולוג	207
74.6	הרופא	17732
76.3	וטרינר	1772
76.4	אורולוג	218
73.1	פסיכולוג	5093
72.6	קרדיולוג	298
69.8	אורטופד	454
66.1	לרופא	12820
66.7	מטופל	6271
66.7	כירופרקט	175
66.6	סקסולוג	50
66.3	פנימאי	47
66.0	סטאז'ר	106
65.3	גניקולוג	1140
64.9	פתולוג	309
64.4	א.א.ג	52
64.3	הוטרינר	980
63.7	רופאה	4307
64.0	ורופא	450
63.4	פסיכיאטרי	1630

Other pre-trained word embeddings

Glove (Pennington et al.):

- Based on ratios of co-occurrence probabilities
- https://nlp.stanford.edu/projects/glove/

Fast-text (Bojanowski et al.):

- Each word is represented as a bag of character n-grams. A vector representation is associated to each character n-gram, and words are represented as the sum of these representations
- https://fasttext.cc/

Similarity

In order to evaluate word embeddings on similarity tasks, we first need to define "similarity"

There are many different ways to define "similarity" and "correlation" between words…

- walk – walking, walk – run, walk – stroll
- Germany – Berlin, Germany – England
- dog – cat, dog – Labrador, dog – leash

This is still an open issue…

Similarity measure

The distance between two vectors is not a good measure

- We do not want to take the length of the vector into account

Most popular similarity measure is Cosine similarity:

- The similarity between two vectors v and w is: $\frac{v \cdot w}{||v||||w||}$

Similarity

Jupiter: find topK

Similarity

You can't always get what you want…

walk - Top5 similar words:

walked, walks, walking, climbs, ride

book – top5 similar words:

books, chapter, novel, abridged, autobiography

Analogies

These models are capable of learning linguistic regularities

For example,

vector(“king”) - vector(“man”) + vector(“woman”) ≅ vector(“queen”)

vector(“mice”) - vector(“mouse”) + vector(“door”) ≅ vector(“doors”)

Jupyter: Analogies

Analogies

How does it work?

Given the analogy $a : a^*, b : b^*$, where word b^* is to be found, we try to maximize the following objective:

$$\arg\max_{b^* \in V}(sim(b^*, b - a + a^*)) \longrightarrow \arg\max_{b^* \in V}(cos(b^*, b - a + a^*))$$

When vectors are normalized, this is equivalent to:

$$\arg\max_{b^* \in V}(cos(b^*, b) - cos(b^*, a) + cos(b^*, a^*))$$

We actually search for a word that is similar to *b*, and *a**, but different from *a*

Linguistic Regularities in Sparse and Explicit Word Representations, Levy and Goldberg, 2014

Analogies

This does not always work that well…

Naturally, depends on the corpus and the hyper-parameters

Additionally, in some cases, specific aspect of relations might dominate others:

London : England , Baghdad : ? $\longrightarrow$ Mosul (instead of Iraq)

Here, even though Iraq is more similar to England than Mosul, the similarity of Mosul to Baghdad dominates, making Mosul the best candidate

A possible solution – use multiplication instead of summation (equivalent to using *log* values):

$$\arg\max_{b^* \in V} \frac{cos(b^*, b)cos(b^*, a)}{cos(b^*, a^*) + \epsilon}$$

Linguistic Regularities in Sparse and Explicit Word Representations, Levy and Goldberg, 2014

Evaluation

Intrinsic Evaluation:

1. Syntactic and semantic analogies:
   - Athens : Greece ; Oslo : ?
   - think : thinking ; read : ?
   - mouse : mice ; door : ?

2. Word correlation benchmarks with human scores (wordsim353, simLex999):

word1	word2	Human score
train	car	6.31
drink	ear	1.31
gem	jewel	8.96

Extrinsic evaluation:

- Show Improvement on downstream tasks when using word embeddings

Classification with word embedding

An optional use of word embedding is a simple classification:

Say we have a short list of professions, and we want to elaborate it

We can run a simple classification model with sklearn

- Use the short list as positive examples
- Add random negative example
- Learn a classification model
- Predict True/False for new words from the vocabulary

Jupyter: Classification example

Biases in word embeddings

A very nice tutorial about word embedding biases:

How to make a racist AI without really trying

https://gist.github.com/rspeer/ef750e7e407e04894cb3b78a82d66aed

Conclusion

Word embedding:

- A powerful word representation
- Easy to incorporate into different models

Can capture word similarities and linguistic regularities

Existing models have their limitations

Need to custom training parameters according to the desired properties and similarities

Questions?

Word	Cosine distance
question	0.744367
answers	0.577580
doubts	0.535749
answer	0.521965
concerns	0.492096
issues	0.487333
unanswered	0.457177
discussions	0.454697
answering	0.448383
matters	0.446861
debates	0.439423
statements	0.433208
objections	0.429943
issue	0.424728
debate	0.421695
conclusions	0.421594
inquiries	0.419190
problem	0.419025
answered	0.414844
responses	0.412600
arguments	0.410255
problems	0.406869

Thank you!