Natural Language Processing 1

Lecture 5: Generalisation and word embeddings

Katia Shutova

ILLC
University of Amsterdam

Outline.

Distributional word clustering

Semantics with dense vectors

Clustering

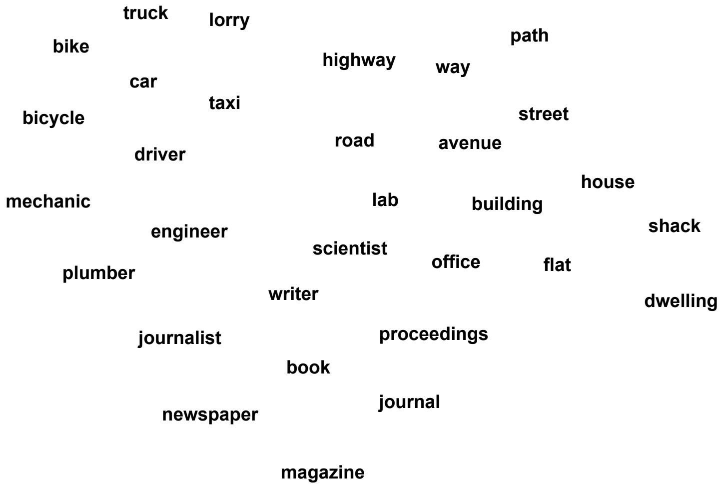
- clustering techniques group objects into clusters
- similar objects in the same cluster, dissimilar objects in different clusters
- allows us to obtain generalisations over the data
- widely used in various NLP tasks:
  - semantics (e.g. word clustering);
  - summarization (e.g. sentence clustering);
  - text mining (e.g. document clustering).

Distributional word clustering

We will:

- cluster words based on the contexts in which they occur
- assumption: words with similar meanings occur in similar contexts, i.e. are distributionally similar
- we will consider noun clustering as an example
- cluster 2000 nouns – most frequent in the British National Corpus
- into 200 clusters

Clustering nouns



Clustering nouns

Feature vectors

- can use different kinds of context as features for clustering
  - window based context
  - parsed or unparsed
  - syntactic dependencies
- different types of context yield different results
- Example experiment: use verbs that take the noun as a direct object or a subject as features for clustering
- Feature vectors: verb lemmas, indexed by dependency type, e.g. subject or direct object
- Feature values: corpus frequencies, normalised

Feature vectors: Examples

tree
131 grow_v_Subj
85 plant_v_Dobj
82 climb_v_Dobj
49 plant_v_Subj
48 see_v_Dobj
46 cut_v_Dobj
40 stand_v_Subj
27 fall_v_Dobj
26 like_v_Dobj
26 fell_v_Subj
25 look_v_Subj
23 make_v_Subj
23 make_v_Dobj
23 grow_v_Dobj
22 use_v_Dobj
22 surround_v_Subj
22 round_v_Dobj
20 overhang_v_Subj
...

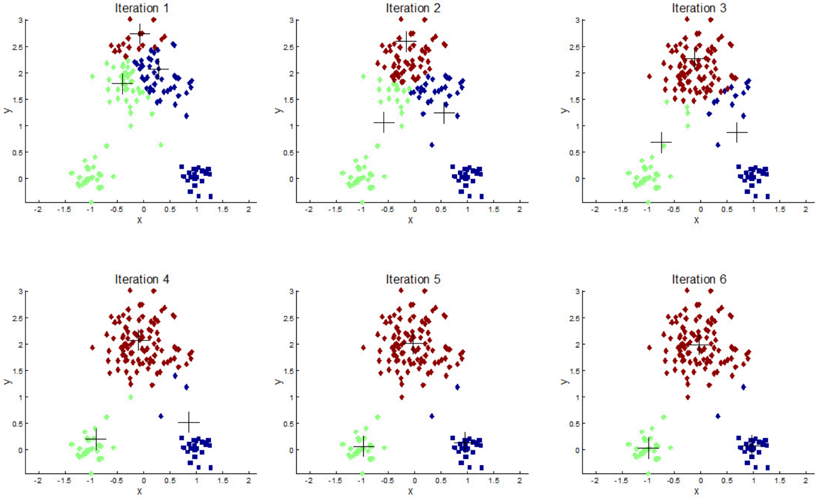
crop
78 grow_v_Subj
76 grow_v_Dobj
44 produce_v_Dobj
23 yield_v_Subj
16 harvest_v_Dobj
12 plant_v_Dobj
10 sow_v_Subj
10 ensure_v_Dobj
10 cut_v_Dobj
9 yield_v_Dobj
9 protect_v_Dobj
9 fail_v_Subj
9 destroy_v_Dobj
8 plant_v_Subj
7 spray_v_Subj
7 spray_v_Dobj
7 lose_v_Dobj
6 feed_v_Subj
...

Clustering algorithms, K-means

- many clustering algorithms are available
- example algorithm: K-means clustering
  - given a set of N data points $\{x_1, x_2, ..., x_N\}$
  - partition the data points into K clusters $C = \{C_1, C_2, ..., C_K\}$
  - minimize the sum of the squares of the distances of each data point to the cluster mean vector μ_i:

$$\arg\min_C \sum_{i=1}^{K} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 \tag{1}$$

K-means clustering



Noun clusters

tree crop flower plant root leaf seed rose wood grain stem forest garden
consent permission concession injunction licence approval
lifetime quarter period century succession stage generation decade phase interval future
subsidy compensation damages allowance payment pension grant
carriage bike vehicle train truck lorry coach taxi
official officer inspector journalist detective constable police policeman reporter
girl other woman child person people
length past mile metre distance inch yard
tide breeze flood wind rain storm weather wave current heat
sister daughter parent relative lover cousin friend wife mother husband brother father

Different senses of ***run***

The children **ran** to the store
If you see this man, **run**!
Service **runs** all the way to Cranbury
She is **running** a relief operation in Sudan
the story or argument **runs** as follows
Does this old car still **run** well?
Interest rates **run** from 5 to 10 percent
Who's **running** for treasurer this year?
They **ran** the tapes over and over again
These dresses **run** small

Subject arguments of ***run***

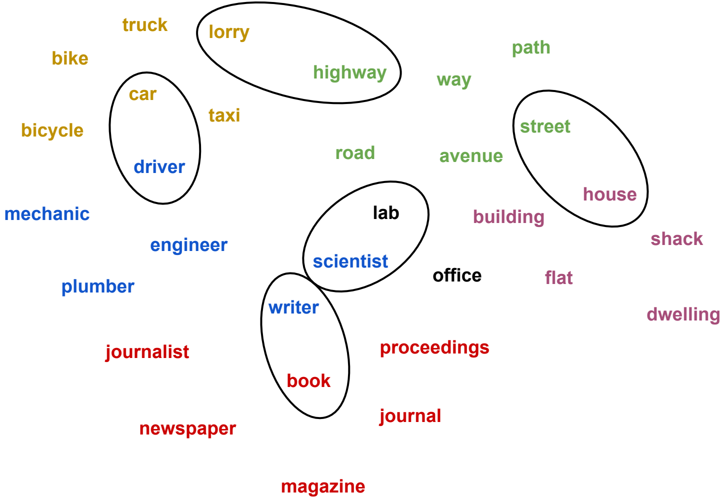
0.2125 drop tear sweat paint blood **water** juice
0.1665 **technology** architecture program system product version interface software tool computer network processor chip package
0.1657 tunnel road **path** trail lane route track street bridge
0.1166 carriage bike **vehicle** train truck lorry coach taxi
0.0919 tide breeze flood wind rain storm weather wave current heat
0.0865 tube lock tank circuit joint filter battery engine device disk furniture machine mine seal **equipment** machinery wheel motor slide disc instrument
0.0792 ocean canal stream bath **river** waters pond pool lake
0.0497 **rope** hook cable wire thread ring knot belt chain string
0.0469 arrangement policy measure reform proposal **project** programme scheme plan course
0.0352 week month year
0.0351 couple minute night morning hour **time** evening afternoon

Subject arguments of ***run*** (continued)

0.0341 criticism appeal charge application allegation claim objection suggestion case complaint
0.0253 championship open tournament league final round race match competition game contest
0.0218 desire hostility anxiety passion doubt fear curiosity enthusiasm impulse instinct emotion feeling suspicion
0.0183 expenditure cost risk expense emission budget spending
0.0136 competitor rival team club champion star winner squad county player liverpool partner leeds
0.0102 being species sheep **animal** creature horse baby human fish male lamb bird rabbit female insect cattle mouse monster
...

Clustering nouns

Clustering nouns



Uses of word clustering in NLP

Widely used in NLP as a source of lexical information:

- Word sense induction and disambiguation
- Modelling predicate-argument structure (e.g. semantic roles)
- Identifying figurative language and idioms
- Paraphrasing and paraphrase detection
- Used in applications directly, e.g. machine translation, information retrieval etc.

Outline.

Distributional word clustering

Semantics with dense vectors

Distributional semantic models

1. Count-based models:
   - Explicit vectors: dimensions are elements in the context
   - **long sparse** vectors with **interpretable** dimensions
2. Prediction-based models:
   - Train a model to predict plausible contexts for a word
   - learn word representations in the process
   - **short dense** vectors with **latent** dimensions

Sparse vs. dense vectors

Why dense vectors?

- easier to use as features in machine learning (less weights to tune)
- may generalize better than storing explicit counts
- may do better at capturing synonymy:
  - e.g. *car* and *automobile* are distinct dimensions in count-based models
  - will not capture similarity between a word with *car* as a neighbour and a word with *automobile* as a neighbour

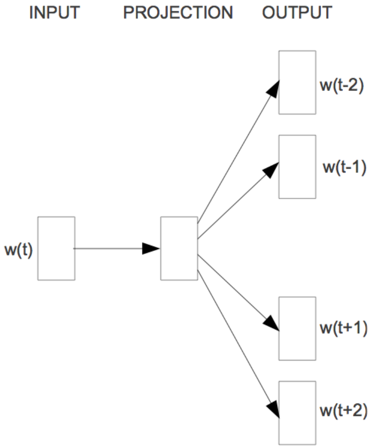
Prediction-based distributional models

Mikolov et. al. 2013. *Efficient Estimation of Word Representations in Vector Space*.

word2vec: **Skip-gram** model

- inspired by work on neural language models
- train a neural network to predict neighboring words
- learn dense embeddings for the words in the training corpus in the process

Skip-gram



Slide credit: Tomas Mikolov

Skip-gram

Intuition: words with similar meanings often occur near each other in texts

Given a word $w(t)$:

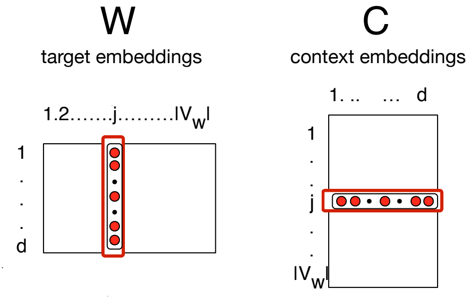
- Predict each neighbouring word
  - in a context window of $2L$ words
  - from the current word.
- For $L = 2$, we predict its 4 neighbouring words:

$$[w(t-2), w(t-1), w(t+1), w(t+2)]$$

Skip-gram: Parameter matrices

Learn 2 embeddings for each word $w_j \in V_w$:

- **word embedding** v, in word matrix W
- **context embedding** c, in context matrix C



Skip-gram: Setup

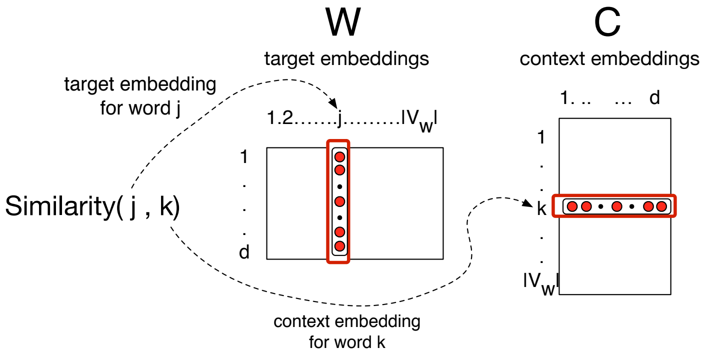
- Walk through the corpus pointing at word $w(t)$, whose index in the vocabulary is j — we will call it w_j
- our goal is to predict $w(t+1)$, whose index in the vocabulary is k — we will call it w_k
- to do this, we need to compute

$$p(w_k|w_j)$$

- Intuition behind skip-gram: to compute this probability we need to compute similarity between w_j and w_k

Skip-gram: Computing similarity

Similarity as dot-product between the target vector and context vector



Slide credit: Dan Jurafsky

Skip-gram: Similarity as dot product

- Remember cosine similarity?

$$cos(v1, v2) = \frac{\sum v1_k * v2_k}{\sqrt{\sum v1_k^2} * \sqrt{\sum v2_k^2}} = \frac{v1 \cdot v2}{||v1||||v2||}$$

It's just a normalised dot product.

- Skip-gram: Similar vectors have a high dot product

$$Similarity(c_k, v_j) \propto c_k \cdot v_j$$

Skip-gram: Compute probabilities

- Compute similarity as a dot product

$$Similarity(c_k, v_j) \propto c_k \cdot v_j$$

- Normalise to turn this into a probability
- by passing through a softmax function:

$$p(w_k|w_j) = \frac{e^{c_k \cdot v_j}}{\sum_{i \in V} e^{c_i \cdot v_j}}$$

Skip-gram: Learning

- Start with some initial embeddings (usually random)
- At training time, walk through the corpus
- iteratively make the embeddings for each word
  - more like the embeddings of its neighbors
  - less like the embeddings of other words.

Skip-gram: Objective

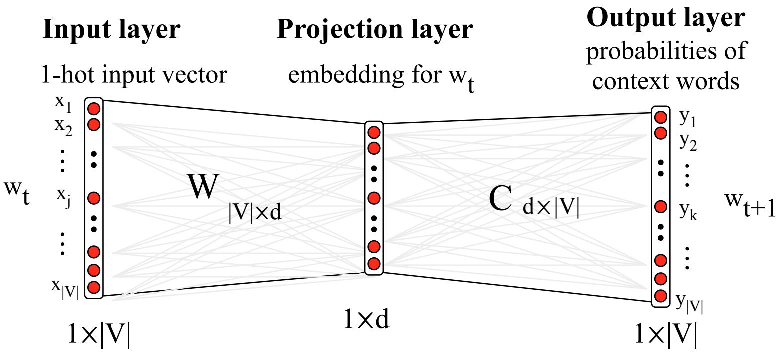
Learn parameters C and W that maximize the overall corpus probability:

$$\arg\max \prod_{(w_j, w_k) \in D} p(w_k | w_j)$$

$$p(w_k | w_j) = \frac{e^{c_k \cdot v_j}}{\sum_{i \in V} e^{c_i \cdot v_j}}$$

$$\arg\max \prod_{(w_j, w_k) \in D} p(w_k | w_j) = \prod_{(w_j, w_k) \in D} \frac{e^{c_k \cdot v_j}}{\sum_{i \in V} e^{c_i \cdot v_j}}$$

Visualising skip-gram as a network



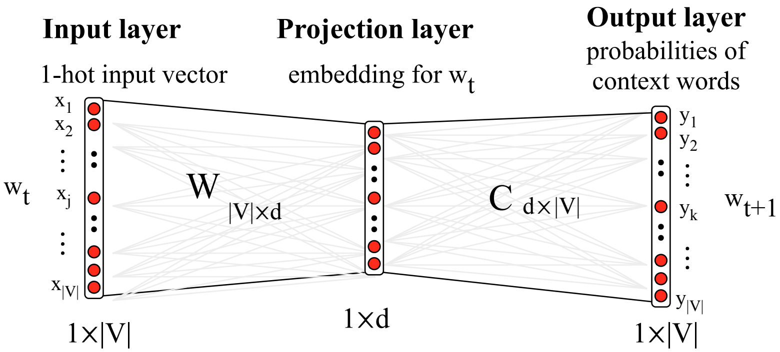
Slide credit: Dan Jurafsky

One hot vectors

- A vector of length |V|
- 1 for the target word and 0 for other words
- So if “bear” is vocabulary word 5
- The one-hot vector is [0,0,0,0,1,0,0,0,0........0]

$$\begin{array}{cccccccccccccccccccc}
w_0 & w_1 & & & & & & & & w_j & & & & & & & & & & w_{|V|} \\
0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 0
\end{array}$$

Visualising skip-gram as a network



Slide credit: Dan Jurafsky

Skip-gram with negative sampling

Problem with softmax: expensive to compute the denominator for the whole vocabulary

$$p(w_k|w_j) = \frac{e^{c_k \cdot v_j}}{\sum_{i \in V} e^{c_i \cdot v_j}}$$

Approximate the denominator: **negative sampling**

- At training time, walk through the corpus
- for each target word and positive context
- sample k noise samples or negative samples, i.e. other words

Skip-gram with negative sampling

- Objective in training:
  - Make the word like the context words

    lemon, a [tablespoon of apricot preserves or] jam.

    c_1 c_2 w c_3 c_4
  - And not like the *k* negative examples

[cement idle dear coaxial apricot attendant whence forever puddle]

n_1 n_2 n_3 n_4 w n_5 n_6 n_7 n_8

Skip-gram with negative sampling: Training examples

Convert the dataset into word pairs:

- **Positive (+)**

  (apricot, tablespoon)
  (apricot, of)
  (apricot, jam)
  (apricot, or)

- **Negative (-)**

  (apricot, cement)
  (apricot, idle)
  (apricot, attendant)
  (apricot, dear)
  ...

Skip-gram with negative sampling

- instead of treating it as a **multi-class problem** (and returning a probability distribution over the whole vocabulary)
- **return a probability** that word w_k is a valid context for word w_j

$$P(+|w_j, w_k)$$

$$P(-|w_j, w_k) = 1 - P(+|w_j, w_k)$$

Skip-gram with negative sampling

- model similarity as dot product

$$Similarity(c_k, v_j) \propto c_k \cdot v_j$$

- turn this into a probability using the **sigmoid function**:

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

$$P(+|w_j, w_k) = \frac{1}{1 + e^{-c_k \cdot v_j}}$$

$$P(-|w_j, w_k) = 1 - P(+|w_j, w_k) = 1 - \frac{1}{1 + e^{-c_k \cdot v_j}} = \frac{1}{1 + e^{c_k \cdot v_j}}$$

Skip-gram with negative sampling

- model similarity as dot product

$$Similarity(c_k, v_j) \propto c_k \cdot v_j$$

- turn this into a probability using the **sigmoid function**:

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

$$P(+|w_j, w_k) = \frac{1}{1 + e^{-c_k \cdot v_j}}$$

$$P(-|w_j, w_k) = 1 - P(+|w_j, w_k) = 1 - \frac{1}{1 + e^{-c_k \cdot v_j}} = \frac{1}{1 + e^{c_k \cdot v_j}}$$

Skip-gram with negative sampling: Objective

- make the word like the context words
- and not like the negative examples

$$\arg\max \prod_{(w_j,w_k)\in D_+} p(+|w_k, w_j) \prod_{(w_j,w_k)\in D_-} p(-|w_k, w_j)$$

$$\arg\max \sum_{(w_j,w_k)\in D_+} \log p(+|w_k, w_j) + \sum_{(w_j,w_k)\in D_-} \log p(-|w_k, w_j) =$$

$$\arg\max \sum_{(w_j,w_k)\in D_+} \log \frac{1}{1+e^{-c_k \cdot v_j}} + \sum_{(w_j,w_k)\in D_-} \log \frac{1}{1+e^{c_k \cdot v_j}}$$

Skip-gram with negative sampling: Objective

- make the word like the context words
- and not like the negative examples

$$\arg\max \prod_{(w_j,w_k)\in D_+} p(+|w_k, w_j) \prod_{(w_j,w_k)\in D_-} p(-|w_k, w_j)$$

$$\arg\max \sum_{(w_j,w_k)\in D_+} \log p(+|w_k, w_j) + \sum_{(w_j,w_k)\in D_-} \log p(-|w_k, w_j) =$$

$$\arg\max \sum_{(w_j,w_k)\in D_+} \log \frac{1}{1+e^{-c_k \cdot v_j}} + \sum_{(w_j,w_k)\in D_-} \log \frac{1}{1+e^{c_k \cdot v_j}}$$

Skip-gram with negative sampling: Objective

- make the word like the context words
- and not like the negative examples

$$\arg\max \prod_{(w_j,w_k)\in D_+} p(+|w_k,w_j) \prod_{(w_j,w_k)\in D_-} p(-|w_k,w_j)$$

$$\arg\max \sum_{(w_j,w_k)\in D_+} \log p(+|w_k,w_j) + \sum_{(w_j,w_k)\in D_-} \log p(-|w_k,w_j) =$$

$$\arg\max \sum_{(w_j,w_k)\in D_+} \log \frac{1}{1+e^{-c_k \cdot v_j}} + \sum_{(w_j,w_k)\in D_-} \log \frac{1}{1+e^{c_k \cdot v_j}}$$

Properties of embeddings

They capture similarity

FRANCE 454	JESUS 1973	XBOX 6909	REDDISH 11724	SCRATCHED 29869	MEGABITS 87025
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Slide credit: Ronan Collobert

Properties of embeddings

They capture analogy

Analogy task: ***a*** *is to* ***b*** *as* ***c*** *is to* ***d***
The system is given words a, b, c, and it needs to find d.

"apple" is to "apples" as "car"" is to ?
"man" is to "woman" as "king" is to ?

Solution: capture analogy via vector offsets

$$a - b \approx c - d$$

$$man - woman \approx king - queen$$

$$d_w = \underset{d'_w \in V}{\operatorname{argmax}}\, cos(a - b, c - d')$$

Properties of embeddings

They capture analogy

Analogy task: ***a*** *is to* ***b*** *as* ***c*** *is to* ***d***
The system is given words a, b, c, and it needs to find d.

"apple" is to "apples" as "car"" is to ?
"man" is to "woman" as "king" is to ?

Solution: capture analogy via vector offsets

$$a - b \approx c - d$$

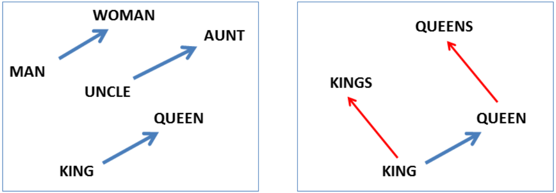
$$man - woman \approx king - queen$$

$$d_w = \underset{d'_w \in V}{\text{argmax}}\, cos(a - b, c - d')$$

Properties of embeddings

Capture analogy via vector offsets

$$man - woman \approx king - queen$$



Mikolov et al. 2013. *Linguistic Regularities in Continuous Space Word Representations*

Properties of embeddings

They capture a range of semantic relations

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Mikolov et al. 2013. *Efficient Estimation of Word Representations in Vector Space*

Word embeddings in practice

Word2vec is often used for pretraining in other tasks.

- It will help your models start from an **informed** position
- Requires only **plain text** - which we have a lot of
- Is very **fast** and easy to use
- Already **pretrained** vectors also available (trained on 100B words)

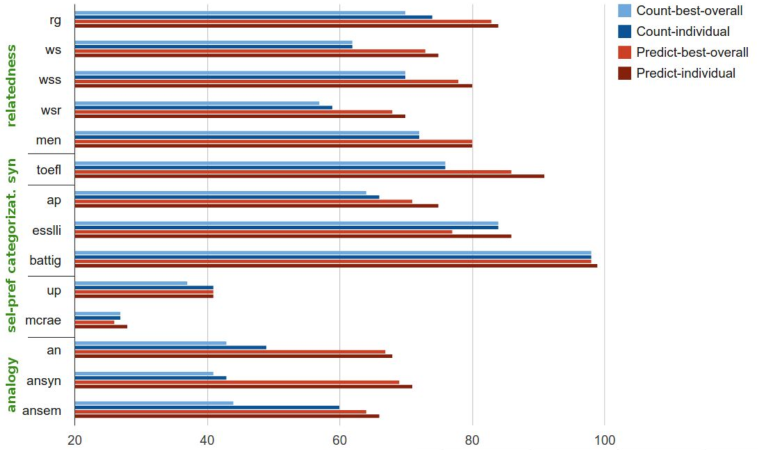
However, for best performance it is important to continue training, fine-tuning the embeddings for a specific task.

Count-based models vs. skip-gram word embeddings

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

- Comparison of count-based and neural word vectors on 5 types of tasks and 14 different datasets:
  1. Semantic relatedness
  2. Synonym detection
  3. Concept categorization
  4. Selectional preferences
  5. Analogy recovery

Count-based models vs. skip-gram word embeddings



Some of these findings were later disputed by Levy et. al. 2015. *Improving Distributional Similarity with Lessons Learned from Word Embeddings*

Acknowledgement

Some slides were adapted from Dan Jurafsky and Marek Rei