CS918: LECTURE 6

Vector Representation and Models for Word Embeddings

Arkaitz Zubiaga, 22nd October, 2018

LECTURE 6: CONTENTS

- Vector space models for language representation.
- Word embeddings.
  - SVD: Singular Value Decomposition.
  - Iteration based models.
  - CBOW and skip-gram models.
  - Word2Vec and Glove.

RECAP: STATISTICAL LANGUAGE MODELS

- Goal: compute the **probability of a sequence of words**:
  $P(W) = P(w_1, w_2, w_3, w_4, w_5, \ldots, w_n)$

- Related task: **probability of an upcoming word**:
- $P(w_5 \mid w_1, w_2, w_3, w_4)$

- Both of the above are **language models**.

VECTOR SPACE MODELS

VECTOR REPRESENTATION

- So far we've extracted **n-gram counts**, etc. from texts.

- For **most NLP tasks**, we need a **vector representation**, which can be fed to:
  - Sentiment classifier.
  - Information retrieval system.
  - Question answering system.
  - Etc.

WORDS AS ATOMIC SYMBOLS

- So far, we have viewed **words as (sequences of) atomic symbols**.
  - We have used **edit distance to compute similarity**.
  - **N-grams & LMs** → what may **follow/precede the word**?

WORDS AS ATOMIC SYMBOLS

- So far, we have viewed **words as (sequences of) atomic symbols**.

- This **doesn’t tell us anything about semantic similarity**, e.g.:
  - Is “Chinese” closer to “Asian” or to “English”?
  - Are “king” & “queen” more related than “doctor” & “mountain”?

WORDS AS ATOMIC SYMBOLS

WARWICK

- We may identify significant similarity based on word overlap between:
  - “Facebook to fight 'fake news' by asking users to rank trust in media outlets”
  - “Facebook's latest fix for fake news: ask users what they trust”

→ Using stemmer/lemmatiser

WORDS AS ATOMIC SYMBOLS

WARWICK

- We may identify significant similarity based on word overlap between:
  - "Facebook to fight 'fake news' by asking users to rank trust in media outlets"
  - "Facebook's latest fix for fake news: ask users what they trust"

→ Using stemmer/lemmatiser

- But we'll fail when there isn't an overlap:
  - "Zuckerberg announces new feature that crowdsources trustworthiness of news organisations"

NO OVERLAP

WORDS AS ATOMIC SYMBOLS

WARWICK

- Likewise for text classification, e.g.:
  - If classifier learns that:

    "Leicester will welcome back Jamie Vardy for their Premier League clash with Watford"

    belongs to the topic "sport"

    NO OVERLAP

  - We'll fail to classify the following also as "sport:"

    "Blind Cricket World Cup: India beat Pakistan by two wickets in thrilling final to retain title"

WORD VECTORS: ONE-HOT OR BINARY MODEL

- Word represented as: $\{0,1\}^{|V|\times 1}$ vector, |V| = vocabulary size

  e.g. V = [hotel, motel, cat, dog], |V| = 4

  hotel = [1, 0, 0, 0]
  motel = [0, 1, 0, 0]
  cat = [0, 0, 1, 0]
  dog = [0, 0, 0, 1]

WORD VECTORS: ONE-HOT OR BINARY MODEL

- Word represented as: $\{0,1\}^{|V|\times 1}$ vector, |V| = vocabulary size
- Still no notion of similarity, e.g.:

$$(w^{hotel})^T w^{motel} = (w^{hotel})^T w^{cat} = 0$$

BAG-OF-WORDS MODEL

- **Bag-of-words:** $\vec{v} = \{|w_1|, |w_2|, \ldots, |w_n|\}$

- Toy example: hello world hello world hello I like chocolate
  $\vec{v} = \{2, 3, 1, 1, 1\}$

- **Widely used**, but largely **being replaced by word embeddings**.
- **Con:** inefficient for large vocabularies.
- **Con:** doesn't capture semantics (each word is an unrelated token)

WORD VECTORS: ONE-HOT OR BINARY MODEL

- **Solution:** why not reduce dimensionality of vector space?

$$\mathbb{R}^{N\times 1} \text{ or (in matrix format) } \mathbb{R}^{N\times|V|}$$

from:

hotel	= [1, 0, 0, 0]
motel	= [0, 1, 0, 0]
cat	= [0, 0, 1, 0]
dog	= [0, 0, 0, 1]

to something like:

hotel	= [1, 0]
motel	= [1, 0]
cat	= [0, 1]
dog	= [0, 1]

1st dimension = 1 if word is a building

2nd dimension = 1 if word is an animal

now we can relate words!

WORD EMBEDDINGS:

SINGULAR VALUE DECOMPOSITION (SVD)

- Assumptions:
  - We can represent **words as vectors** of some dimension.
  - Each **dimension has some semantic meaning**, unknown a priori, but could be e.g.:
    - Whether it is an object/concept/person.
    - Gender of person.
    - …

INTUITION OF WORD EMBEDDINGS

- Words with the same context will have similar meaning:

buy a car
buy chocolate
don’t **buy**
will you **buy** it?

purchase a car
purchase chocolate
don’t **purchase**
will you **purchase** it?

get a car
get chocolate
don’t **get**
will you **get** it?

buy, purchase and get occur in equal or very similar contexts
they must have **similar meanings**!

BUILDING A CO-OCCURRENCE MATRIX

- Given as input:
  - A **text/corpus**.
  - An **offset Δ** (e.g. 5 words)

- In a co-occurrence matrix with |V| rows, |V| columns:
  - The (i, j)th value indicates the **number of times words i and j co-occur within the given offset Δ.**

BUILDING A CO-OCCURRENCE MATRIX

WARWICK

- Examples (Δ = 2 words):
  - We need to tackle fake news to keep society informed.
  - How can we build a classifier to deal with fake news?
  - Fake co-occurs with: to(2), news(2), deal(1), tackle(1), with(1)
  - Deal (with) and tackle are different tokens for us.
    - **Frequent occurrence in similar contexts will indicate similarity.**

WORD EMBEDDINGS: WORD-WORD MATRIX

WARWICK

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

- The table will be **huge (and sparse) for large |V|** (vocabularies).
- We need to **reduce the dimensionality**.

WORD EMBEDDINGS: SVD METHODS

WARWICK

- SVD: **Singular Value Decomposition**
- We **build co-occurrence matrix** (|V|x|V|) with offset Δ.
- We use **SVD to decompose X** as $X = USV^T$, where:
  - U(|V| x r) and V(|V| x r) are unitary matrices, and
  - S(r x r) is a diagonal matrix.
- The **columns of U** (the left singular vectors) are then the **word embeddings of the vocabulary**.

WORD EMBEDDINGS: SVD METHODS

$$
\begin{array}{c} \\ |V| \end{array}
\begin{array}{c} |V| \\ \begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix} \end{array}
=
\begin{array}{c} \\ |V| \end{array}
\begin{array}{c} |V| \\ \begin{bmatrix} | & | & \\ u_1 & u_2 & \cdots \\ | & | & \end{bmatrix} \end{array}
\begin{array}{c} \\ |V| \end{array}
\begin{array}{c} |V| \\ \begin{bmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \end{array}
\begin{array}{c} \\ |V| \end{array}
\begin{array}{c} |V| \\ \begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \end{bmatrix} \end{array}
$$

Reducing dimensionality by selecting first k singular vectors:

$$
\begin{array}{c} \\ |V| \end{array}
\begin{array}{c} |V| \\ \begin{bmatrix} & & \\ & \hat{X} & \\ & & \end{bmatrix} \end{array}
=
\begin{array}{c} \\ |V| \end{array}
\begin{array}{c} k \\ \begin{bmatrix} | & | & \\ u_1 & u_2 & \cdots \\ | & | & \end{bmatrix} \end{array}
\begin{array}{c} \\ k \end{array}
\begin{array}{c} k \\ \begin{bmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \end{array}
\begin{array}{c} \\ k \end{array}
\begin{array}{c} |V| \\ \begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \end{bmatrix} \end{array}
$$

We get |*V*| vectors of *k* dimensions each: word embeddings

e.g. word embedding of word w:

WE(w) = {v_1, v_2, …, v_k}

We've **reduced *w*'s dimensionality from |*V*| to *k***.

SVD EXAMPLE IN PYTHON

WARWICK

Corpus:
I like deep learning. I like NLP. I enjoy flying.

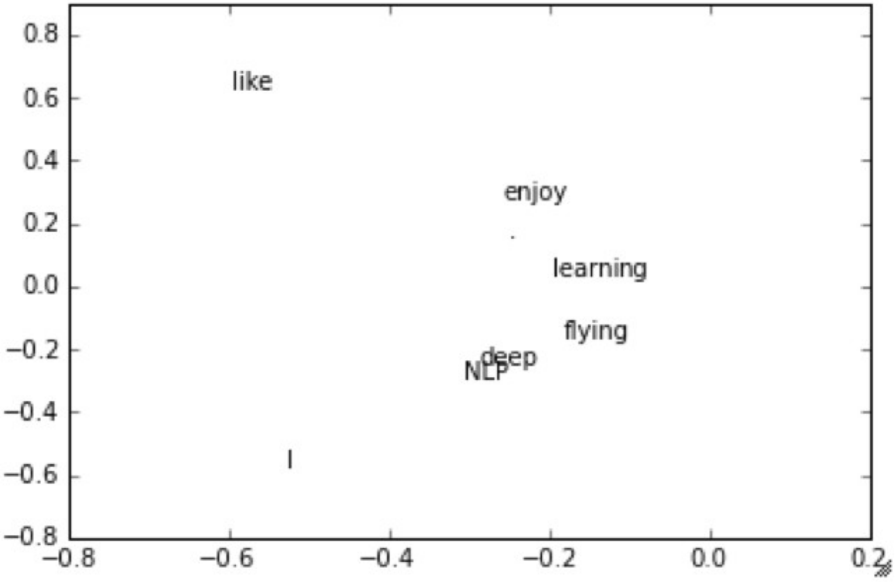
```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learning","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```

$\Delta = 1$

like & I co-occur twice

PLOTTING SVD EXAMPLE IN PYTHON

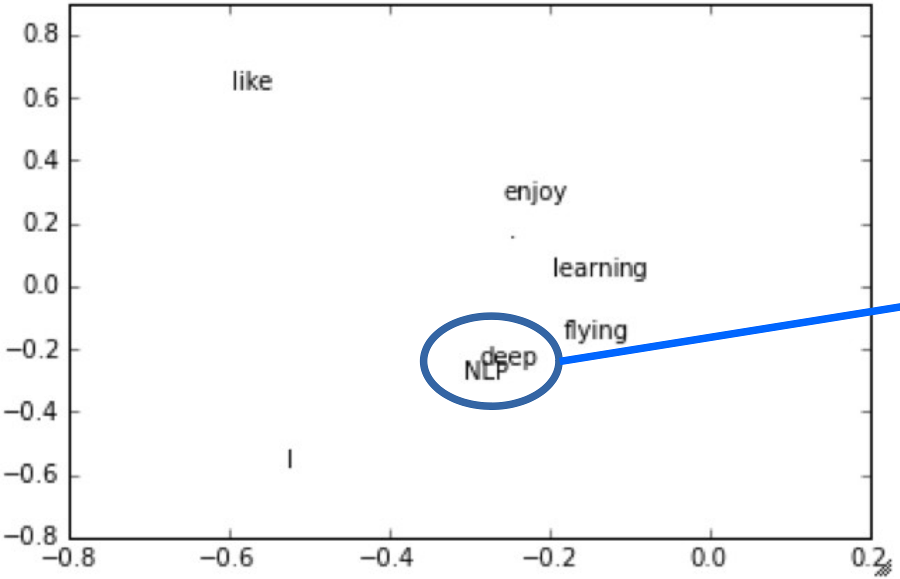
- **Corpus:** I like NLP. I like deep learning. I enjoy flying.




```
for i in xrange(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```

PLOTTING SVD EXAMPLE IN PYTHON

- **Corpus:** I like NLP. I like deep learning. I enjoy flying.

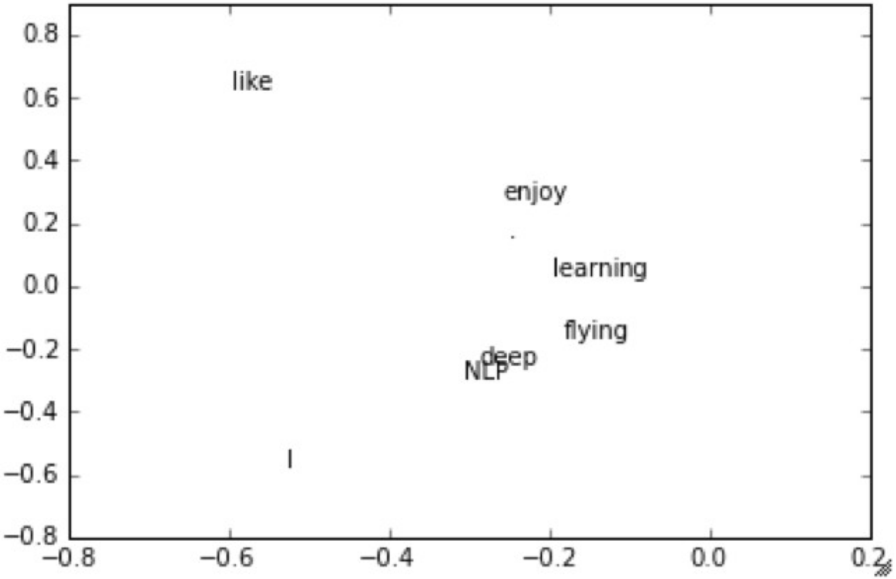



```
for i in xrange(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```

NLP and deep aren't directly connected in the corpus (Δ is 1), but have common context (like)

COMPUTING WORD SIMILARITY

- **Corpus:** I like NLP. I like deep learning. I enjoy flying.




```
for i in xrange(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```

We can compute similarity between w_i and w_j by comparing: U[i, 0:k] and U[j, 0:k]

COMPUTING WORD SIMILARITY

WARWICK

- Given 2 words w_1 and w_2, similarity is computed as:
  - Dot/inner product, which equates:
    $\mathbf{|w_1| * |w_2| * \cos(\theta)}$

Cosine of angle between vectors

Length of w_2

Length of w_1

COMPUTING WORD SIMILARITY

WARWICK

- Given 2 words w_1 and w_2, similarity is computed as:
  - Dot/inner product, which equates:
    $|\mathbf{w_1}| * |\mathbf{w_2}| * \cos(\theta)$
  - **High similarity** for:
    near-parallel vectors with high values in same dimensions.
  - **Low similarity** for:
    orthogonal vectors, low value vectors.

PROS AND CONS OF SVD

- **Pro:** has shown to perform well in a number of tasks.
  - Useful e.g. for topic models, Latent Dirichlet Allocation (LDA).

- **Con:** dimensions need to change as new words are added to the corpus, costly.
- **Con:** resulting vectors can still be high dimensional and sparse.
- **Con:** Quadratic cost to perform SVD.

WORD EMBEDDINGS:

STATE-OF-THE-ART ALTERNATIVES TO SVD

ITERATION BASED METHODS: WORD2VEC

WARWICK

- **Main intuition:** Instead of computing co-occurrences from entire corpus, **predict surrounding words** in a window of length c of every word.
  - Allows **easier updates**, faster to incorporate new words in model.
  - Leads to **low dimensional, dense vectors**.
  - This is the idea behind **word2vec** (Mikolov 2013)

ALTERNATIVE: ITERATION BASED METHODS

WARWICK

- **Intuition: predict surrounding words.**

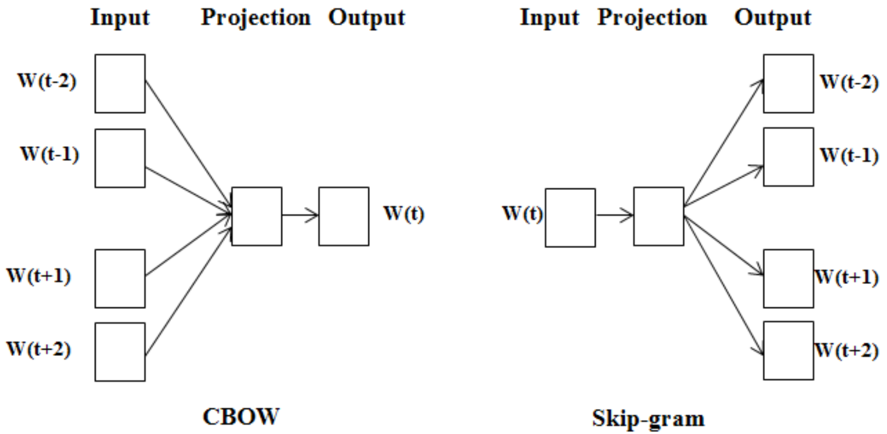
  e.g. will you **X** it? → try to predict X, use a neural network to predict and refine predictions.

  will you **buy** it? → high score
  will you **purchase** it? → high score

  will you **beer** it? → low score
  will you **university** it? → low score

WORD2VEC: CBOW AND SKIPGRAM MODELS

- **Continuous bag of words model (CBOW):** having the context, predict a word.
- **Skip gram model:** having the word, predict its context.



WORD2VEC: WHY IS IT COOL?

- They are very good for encoding 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 (Semeval 2012 task 2)

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

WORD2VEC: WHY IS IT COOL?

- They are **very good for inferring word relations:**
  - v('Paris') - v('France') + v('Italy') = v('Rome')
  - v('king') - v('man') + v('woman') = v('queen')

PROS AND CONS: ITERATION BASED METHODS

- **Pro:** Do not need to operate on entire corpus which involves very sparse matrices.
- **Pro:** Can capture semantic properties of words as linear relationships between word vectors.
- **Pro:** Fast and can be easily updated with new sentences.
- **Con:** Can't take into account the vast amount of repetition in the data.

ANOTHER ALTERNATIVE: GLOVE

- **Glove (Pennington et al. 2014)**, is similar to word2vec:
  - Count-based method instead of prediction-based.
  - Does matrix factorisation for dimensionality reduction.

- Can leverage repetitions in the corpus as using the entire word co-occurrence matrix.

- **How?** Train only on non-zero entries of the co-occurrence matrix.

GLOVE

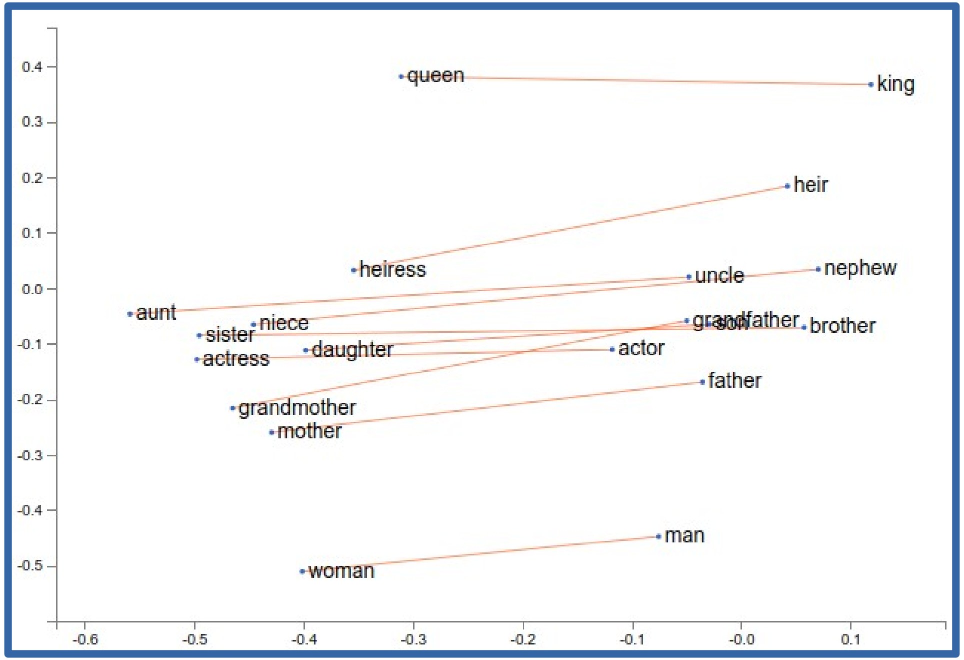
WARWICK

- **Computationally expensive for building matrix, then much faster** as non-zero entries are not so many.

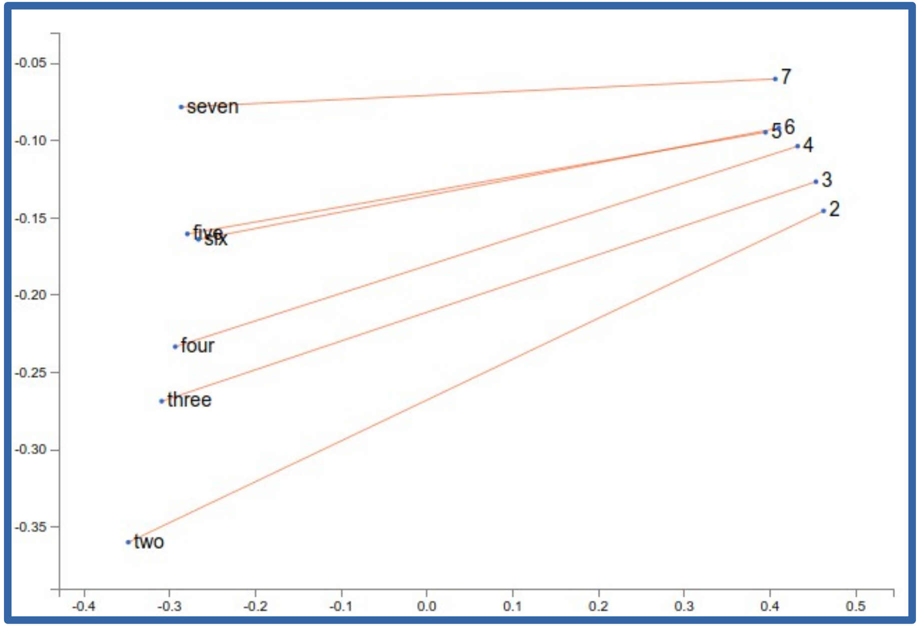
- **Intuition:** relationships between words should be explored in terms of their co-occurrence probabilities with some selected words k.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

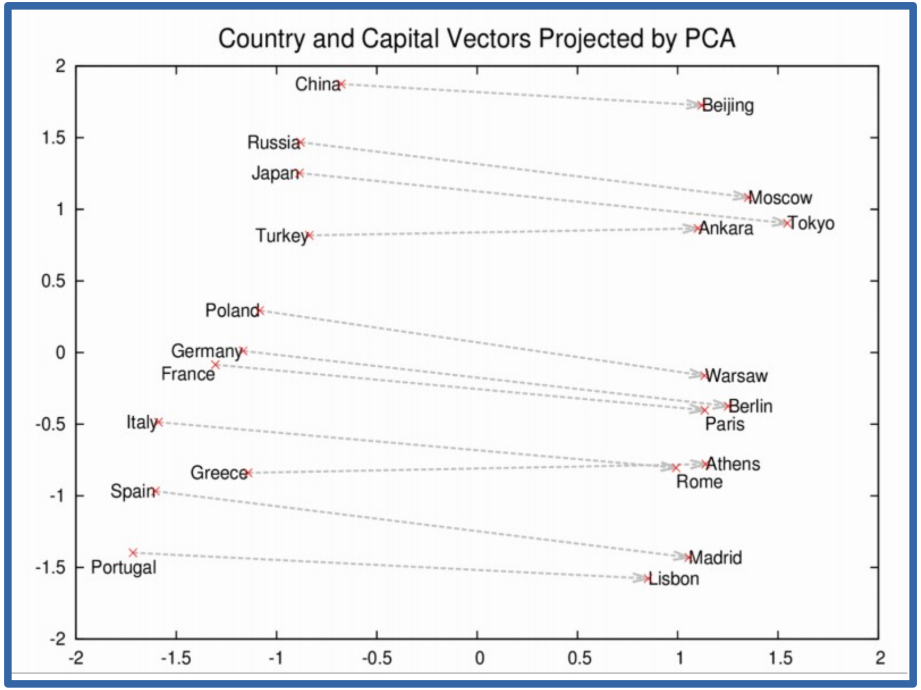
GLOVE: VISUALISATION



GLOVE: VISUALISATION



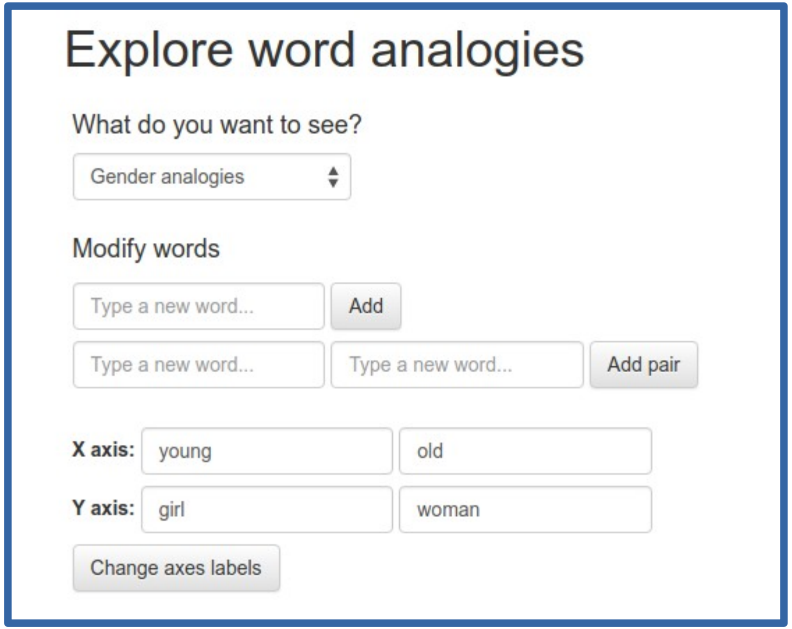
GLOVE: VISUALISATION



GLOVE: VISUALISATION

- Want to play around?

  https://lamyiowce.github.io/word2viz/



EVALUATION OF WORD EMBEDDINGS

- **Extrinsic evaluation:** test your model in a text classification, sentiment analysis, machine translation,… task!
  - Does it **outperform other methods** (e.g. bag-of-words)?
  - **Compare two models** A and B: which one's better?

EVALUATION OF WORD EMBEDDINGS

WARWICK

- **Intrinsic evaluation**:
  - Use datasets labelled with word similarities:
    - e.g. TOEFL dataset: “pose” is closest in meaning to:
      a) claim, b) model, c) assume, d) present

      do we get it right with embeddings?
  - Common sense:
    - Paris + UK – France = London?

PYTHON: USING WORD2VEC

- Preparing the input:
  Word2Vec takes lists of lists of words (lists of sentences) as input.

  e.g.:

```
sentences = [[‘this’, ‘is’, ‘my’, first', 'sentence'],
             [‘a’, ‘short’, ‘sentence’],
             [‘another’, ‘sentence’],
             ['and’, ‘this’, ‘is’, ‘the’, ‘last’, ‘one’]]
```

PYTHON: USING WORD2VEC

- Training the model:

```
model = Word2Vec(sentences, min_count=10, size=300, workers=4)
```

We will only train vectors for words occurring 10+ times in the corpus

We want to parallellise the task running 4 processes

We want to produce word vectors of 300 dimensions

PYTHON: USING WORD2VEC

WARWICK

- It’s memory intensive!

  It stores matrices: **#vocabulary** (dependent on min_count), **#size** (size parameter) of **floats** (single precision aka 4 bytes).

  Three such matrices are held in RAM. If you have:
  100,000 unique words, size=200, the model will require approx.:

  100,000*200*4*3 bytes = ~229MB.

PYTHON: USING WORD2VEC

- Storing a model:

```
model = Word2Vec.load_word2vec_format('mymodel.txt', binary=False)
```

- or

```
model = Word2Vec.load_word2vec_format('mymodel.bin.gz', binary=True)
```

- Resuming training:

```
model = gensim.models.Word2Vec.load('mymodel.bin.gz')
model.train(more_sentences)
```

WARWICK

Using the model

Word2vec supports several word similarity tasks out of the box:

```
model.most_similar(positive=['woman', 'king'], negative=['man'], topn=1)
[('queen', 0.50882536)]
model.doesnt_match("breakfast cereal dinner lunch";.split())
'cereal'
model.similarity('woman', 'man')
0.73723527
```

If you need the raw output vectors in your application, you can access these either on a word-by-word basis

```
model['computer']  # raw NumPy vector of a word
array([-0.00449447, -0.00310097,  0.02421786, ...], dtype=float32)
```

PYTHON: USING WORD2VEC

```
model['computer']  # raw NumPy vector of a word
array([-0.00449447, -0.00310097,  0.02421786, ...], dtype=float32)
```

- This will give us the vector representation of ‘computer:’

  v(‘computer’) = {-0.00449447, -0.00310097, ...}

- How do we get then the vector representations for sentences, e.g.:

  I have installed Ubuntu on my computer

- **Vector representations for sentences**, e.g.:
  - I have installed Ubuntu on my computer

- Standard practice is either of:
  - **Summing word vectors** (they have the same dimensionality):
    v('I') + v('have') + v('installed') + v('Ubuntu') + …

  - **Getting the average of word vectors:**
    (v('I') + v('have') + v('installed') + …) / 7

PRE-TRAINED WORD VECTORS

WARWICK

- One can train a model from a large corpus (millions, if not billions of sentences). Can be time-consuming, memory-intensive.

- **Pre-trained models are available.**
  - **Remember to choose a suitable pre-trained model.**
  - Don’t use word vectors pre-trained from news articles when you’re working with social media!

PRE-TRAINED WORD VECTORS

- Glove's pre-trained vectors: https://nlp.stanford.edu/projects/glove/

Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose http://www.opendatacommons.org/licenses/pddl/1.0/.
  - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB d
  - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip
  - Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors, 2.03 GB download): glove.840B.300d.zip
  - Twitter (2B tweets, 27B tokens, 1.2M vocab, uncased, 25d, 50d, 100d, & 200d vectors, 1.42 GB download): glo
- Ruby script for preprocessing Twitter data

PRE-TRAINED WORD VECTORS

WARWICK

- Pre-trained word vectors for 30+ languages (from Wikipedia): https://github.com/Kyubyong/wordvectors

Language	ISO 639-1	Vector Size	Corpus Size	Vocabulary Size
Bengali (w) | Bengali (f)	bn	300	147M	10059
Catalan (w) | Catalan (f)	ca	300	967M	50013
Chinese (w) | Chinese (f)	zh	300	1G	50101
Danish (w) | Danish (f)	da	300	295M	30134
Dutch (w) | Dutch (f)	nl	300	1G	50160
Esperanto (w) | Esperanto (f)	eo	300	1G	50597
Finnish (w) | Finnish (f)	fi	300	467M	30029
French (w) | French (f)	fr	300	1G	50130
German (w) | German (f)	de	300	1G	50006
Hindi (w) | Hindi (f)	hi	300	323M	30393
Hungarian (w) | Hungarian (f)	hu	300	692M	40122
Indonesian (w) | Indonesian (f)	id	300	402M	30048

PRE-TRAINED WORD VECTORS

- UK Twitter word embeddings: https://figshare.com/articles/UK_Twitter_word_embeddings_II_/5791650

UK Twitter word embeddings (II)

16.01.2018, 23:08 by Vasileios Lampos

Word embeddings trained on UK Twitter content (II)

The total number of tweets used was approximately 1.1 billion, covering the years 2012 to and including 2016.

Settings: Skip-gram with negative sampling (10 noise words), a window of 9 words, 512 layers (dimensionality) and 10 epochs of training.

PRE-TRAINED WORD VECTORS

- Twitter Word2Vec model: https://www.fredericgodin.com/software/

Twitter Word2vec model

As part of our ACL W-NUT 2015 shared task paper, we release a Twitter word2vec model trained on 400 million tweets, as described in detail in this paper. The model, including Python code to load and access it, can be downloaded here.

REFERENCES

- Gensim (word2vec): https://radimrehurek.com/gensim/
- Word2vec tutorial: https://rare-technologies.com/word2vec-tutorial/
- FastText: https://github.com/facebookresearch/fastText/
- GloVe: Global Vectors for Word Representation: https://nlp.stanford.edu/projects/glove/

ASSOCIATED READING

- Jurafsky, Daniel, and James H. Martin. 2009. Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. 3rd edition. **Chapter 6.3-6.13**.