Word Embeddings

Bonan Min

bonanmin@gmail.com

Some slides are based on class materials from Thien Huu Nguyen, Richard Socher

Words

The primary elements of natural languages

Each word carries some unit meaning depending on its context

The unit meanings of the words are composed/combined to produce new and more complicated meanings/concepts (e.g., sentences, documents)

Word Meanings

The fundamental of NLP is to be able to allow computers to understand meanings of text

$$
\begin{aligned}
Meaning(\text{“I have a cat”}) = f(\, & Meaning(\text{“I”}), \\
& Meaning(\text{“have”}), \\
& Meaning(\text{“a”}), \\
& Meaning(\text{“cat”})\,)
\end{aligned}
$$

How do we capture/approximate the composition function f and the $Meaning$ function for words?

We will discuss word meanings in this lecture

What Are Meanings?

Definition (Webster dictionary)

The idea that is represented by a word, phrase, etc.

The idea that a person wants to express by using words, signs, etc.

The idea that is expressed in a word of writing, art, etc.

How To Represent The Meanings Of A Word In Computers?

Common solution: Use the sets of synonyms and hypernyms of the word by querying some thesaurus (e.g., WordNet)

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
            ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g. hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Problems With Resources Like WordNet

Great as a resource but missing nuance

- e.g., “proficient” is listed as a synonym for “good”, but this is only true in some contexts.

Missing new meanings of words

- e.g., wicked, badass, nifty, wizard, genius, ninja, bombast
- very challenging to keep up-to-date.

Subjective

Require human labor to create and adapt

Difficult to compute word similarity

Representing Words As Discrete Symbols

In traditional NLP, words are considered as discrete symbols

Mathematically, words are represented by one-hot vectors, where:

- The dimension of the vector = the number of words in some given vocabulary (e.g., 500,000)
- Only the bit corresponding to the word is set to 1 (i.e, 0 otherwise)

hotel = [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]

motel = [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]

- This is call the localist representation (to be distinguished with distributed representation in cognitive science later)

Problems With Words As Discrete Symbols

The size of the vectors is large

The vectors for any pair of words are orthogonal (i.e., cosine similarity = 0), but for similar words like “hotel” and “motel”, we expect their vectors to exhibit some level of similarity (i.e., the cosine similarity should be non-zero).

- e.g., in web search, a search for “Seattle hotel” should return documents with “Seattle model” as well.

Solution for this?

- Can we use the idea of synonyms and hyponyms for such one-hot vectors?
  - Not working well in practice (e.g., incompleteness)
- Learn to explicitly encode similarity in the word vectors themselves, reduce the size of the vectors, go from binary vectors to continuous vectors

Representing Words By Their Contexts

Distributional semantics: a word's meaning is given by the words that frequently appear close-by

- "*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

When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).

Use the many contexts of *w* to build up a representation of *w*

…government debt problems turning into	*banking*	*crises as happened in 2009…*
…saying that Europe needs unified	*banking*	*regulation to replace the hodgepodge…*
…India has just given its	*banking*	*system a shot in the arm…*

Word Vectors

We will introduce a dense vector for each word, chosen so that it is similar to vectors of words appearing in similar contexts.

Word vectors are also called word embeddings or word representations. This is a distributed representation, e.g.,

$$\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Localist Representation Vs. Distributed Representation

In cognitive science, distributed representation has the following property (Hilton et al., 1986; Plate, 2012):

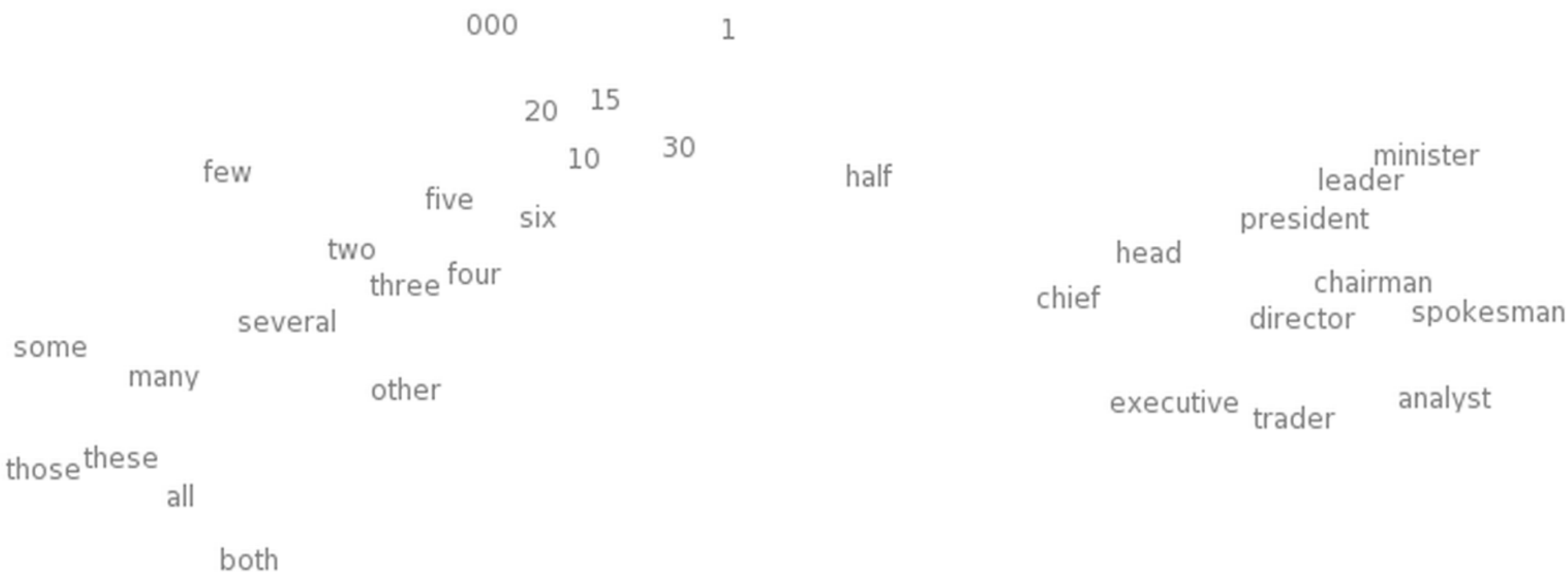
- A concept is represented by a pattern of activity over a collection of neurons (i.e., more than one neuron is required to represent a concept.)
- Each neuron participates in the representation of more than one concept.

By contrast, in localist representation, each neuron represents a single concept on a stand-alone basis. The critical distinction is that localist units have "meaning and interpretation" whereas units in distributed representation don't.

- "These representations are distributed, which typically has the consequence that interpretable information cannot be obtained by examining activity of single hidden units." – Elman, 1995.

Roy, Asim. "A theory of the brain: localist representation is used widely in the brain." Frontiers in psychology vol. 3 551

Word Meaning As A Neural Word Vector



How Do We Obtain Such Word Vectors?

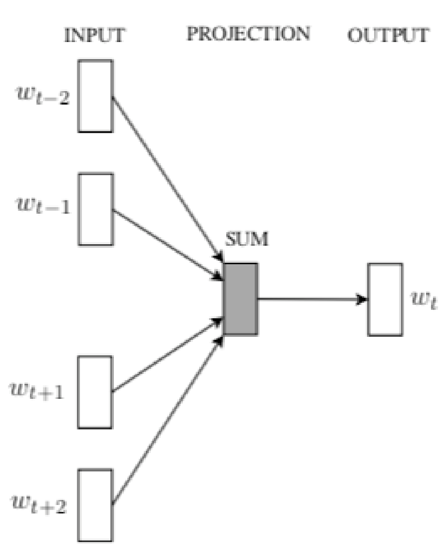
Word2vec (Mikolove et al. 2013) is a popular framework to learn word vectors (although many other efforts have been made before it)

Idea:

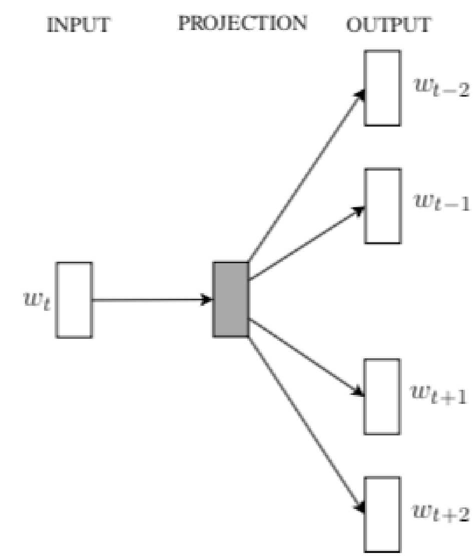
- We start with a large corpus of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context words o (surrounding words)
- Use the similarity of the word vectors for c and o to compute the probability of c given o ($P(c|o)$) (or vice versa)
- Keep updating the word vectors to maximize this probability

Two Variants Of Word2vec

Context words: windows of size 2 before and after the center word



Continuous Bag of Words (CBOW):
predicting the center words using
the context words ($P(w_t|w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2})$)



Skip-grams (SG):
predicting the context words using
the center word ($P(w_{t+i}|w_t), i \in \{-2, -1, 1, 2\}$)

We will only discuss Skip-grams due to its popularity

Wovd2vec: SG Objective Function

For each position $i = 1, \dots, N$, predict the context words within a window of fixed size m, given the the center word w_i:

$$\text{Likelihood} = L(\theta) = \prod_{i=1}^{N} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{i+j} | w_i; \theta)$$

The objective/loss function is the (average) negative log likelihood:

$$\text{loss} = J(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P(w_{i+j} | w_i; \theta)$$

θ is the parameter used to define $P(w_{i+j} | w_i; \theta)$. It is the model parameters

Minimizing the loss function amounts to maximizing the predictive accuracy

Wovd2vec: SG Objective Function

How do we compute $P(w_{i+j}|w_i;\theta)$?

We will use two vectors per word w:

- v_w when w is a center word
- u_w when w is a context word
- Using two vectors makes the later optimization easier, average both at the end to obtain final word vectors
- Although using one vector per word is possible too

Dot product measures similarity of two vectors

$$u^T v = u.v = \sum u_i v_i$$

Then:

$$P\left(w_{i+j}\middle|w_i;\theta\right) = \frac{\exp(u_{w_{i+j}}^T v_{w_i})}{\sum_{w\in V}\exp(u_w^T v_{w_i})}$$

What is θ in this case?

Wovd2vec: SG Objective Function

We compute the probability using the softmax function

$$P\left(w_{i+j}\middle|w_i;\theta\right) = \frac{\exp(u_{w_{i+j}}^T v_{w_i})}{\sum_{w\in V}\exp(u_w^T v_{w_i})}$$

As for training, we have a loss function $J(\theta)$ with the word vectors as the parameters.

We want to find the parameters (word vectors) that can minimize this loss function.

- Can be solved by stochastic gradient descent.

Negative Sampling

$$P\left(w_{i+j} \middle| w_i; \theta\right) = \frac{\exp(u_{w_{i+j}}^T v_{w_i})}{\sum_{w \in V} \exp(u_w^T v_{w_i})}$$

The normalization factor needs to enumerate over all the words in the vocabulary that can be very large!

We can instead obtain only a sample of the vocabulary to estimate the normalization factor. This is called Negative Sampling as every word other than w_{i+j} is considered as negative in this case.

Negative Sampling in the Original Paper

Paper: “Distributed Representations of Words and Phrases and their Compositionality” (Mikolov et al., 2013).

Train binary logistic regression for a true pair (a center word and a word in its context window) versus several noise pairs (the center word paired with a random word)

Overall objection function to maximize:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta)$$

$$J_t(\theta) = \log \sigma\left(u_o^T v_c\right) + \sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)} \left[\log \sigma\left(-u_j^T v_c\right)\right]$$

The sigmoid function : $\sigma(x) = \frac{1}{1+e^{-x}}$

In the loss function, we basically maximize the probability of two words co-occurring in the first log

The Skip-gram Model With Negative Sampling (Implementation)

This is to minimize

$$J_{neg-sample}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))$$

Negative sampling: take K negative samples (using word probabilities)

- $P(w) = U(w)^{3/4}/Z$: the unigram distribution $U(w)$ raised to the power of ¾
- The power increase the probability for less frequent words and decrease the probability for more frequent words

is: 0.9^(3/4)/1.11 = 0.92/1.11 =0.83
Constitution: 0.09^(3/4) /1.11 = 0.16/1.11=0.14
bombastic: 0.01^(3/4) /1.11 = 0.032/1.11=0.03

Objective

- Maximize the probability that a real word appearing in context
- Minimize the probability that random words appear around the center word

Co-occurrence Counts

Word2Vec capture the co-occurrence of words via the prediction tasks.

A simpler approach to capture word co-occurrence is via the direct co-occurrence counts between words and X

Two options for X: words in windows and full documents

- Window: Counts are done between pairs of words. Similar to Word2Vec, use window around each word -> capturing both syntactic (POS) and semantic information

- Document: The co-occurrence counts are done between words and documents, encoding the general topics and leading to "Latent Semantic Analysis"

https://en.wikipedia.org/wiki/Latent_semantic_analysis

Example: Window Based Co-occurrence Matrix

Window length 1 (although 5-10 are more common)

Symmetric (don’t distinguish left or right context)

Example corpus:

- I like deep learning.
- I like NLP.
- I enjoy flying.

Example: Window Based Co-occurrence Matrix

Example corpus:

- I like deep learning.
- I like NLP.
- I enjoy flying.

counts	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

These are the word vectors!

Problems With Simple Co-occurrence Vectors

Increase in size with vocabulary

Very high dimensional: need a lot of storage

Subsequent classification models have sparsity issues

Thus, models are less robust

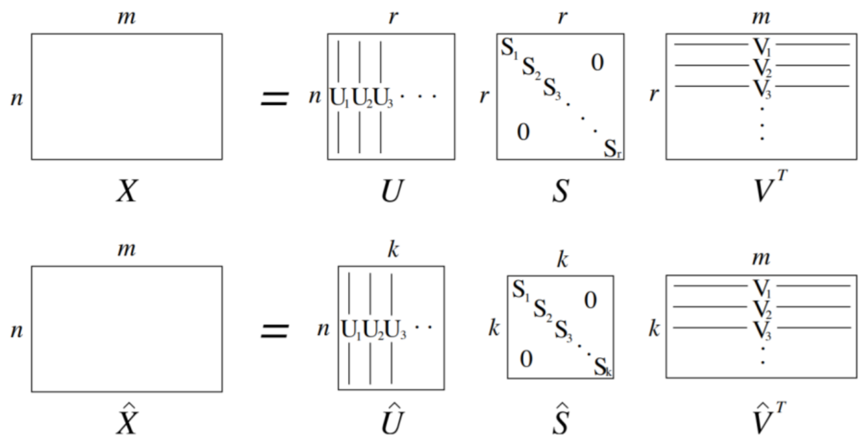
Solution: Low dimensional vectors

- Idea: store most of the important information in a fixed, small number of dimensions: a dense vector
- Usually 25-1000 dimensions (like Word2Vec)
- Main question: How to reduce the dimensionality?

Method 1: Dimensionality Reduction

Singular Value Decomposition (SVD) of the co-occurrence matrix X

Factorize X into USV^T where U and V are orthonormal ($U^T \cdot U = I$ and $V^T \cdot V = I$)



Retain only k singular values, in order to generalize.

$\hat{X}$ is the best rank k approximation to X, in terms of least squares.

Classic linear algebra result. Very expensive to compute for large matrices.

Some Tricks For Dimensionality Reduction

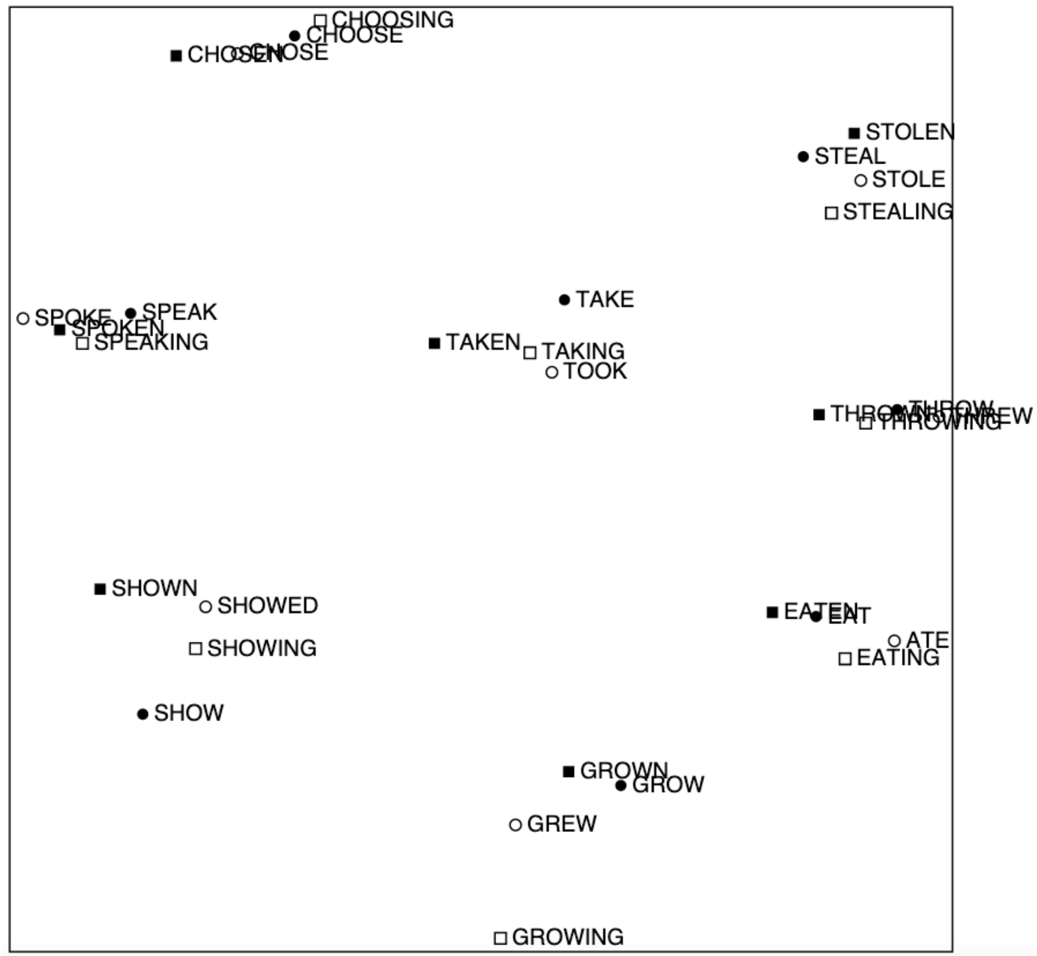
Scaling the counts in the cells of A can help a lot

- Problem: function words (*the*, *he*, *has*) are too frequent, so syntax has too much impact. Some fixes:
  - $Min(A, t)$ with $t \approx 100$
  - Ignore them all

Use Pearson correlations instead of counts, then set negative values to 0

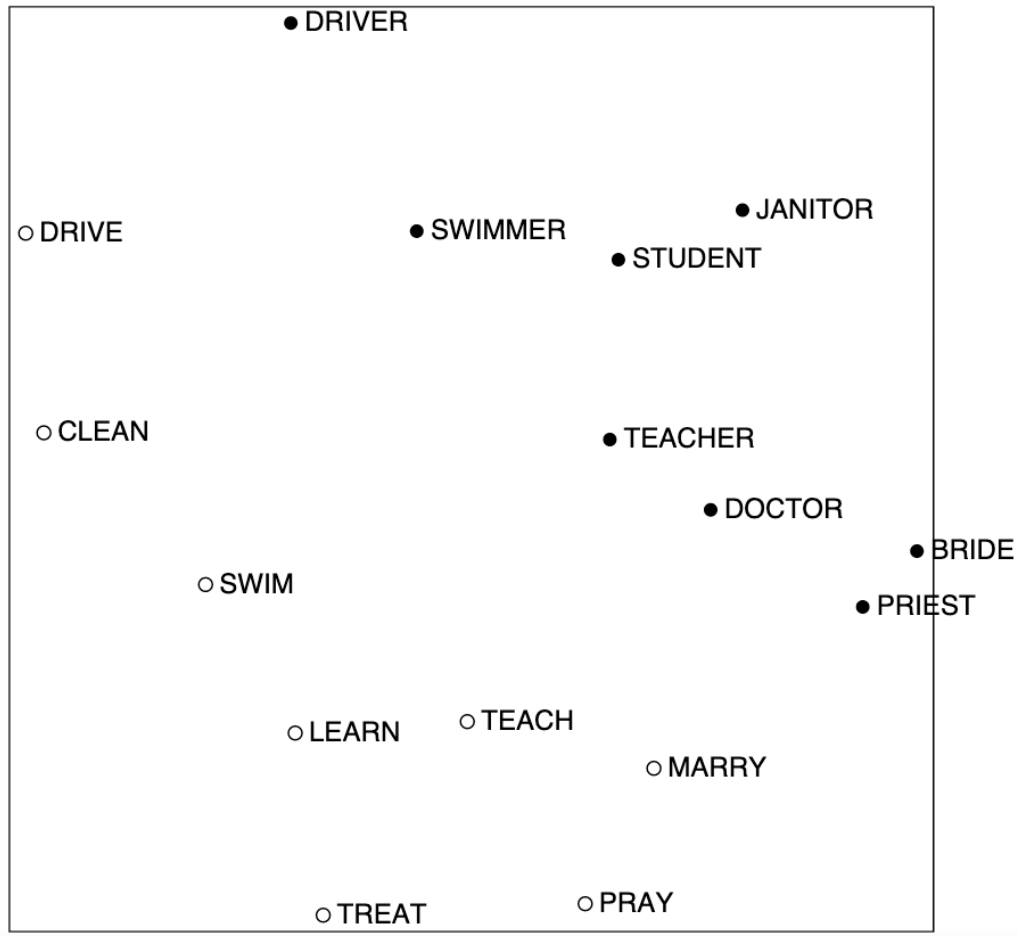
…

Interesting Syntactic Patterns Emerging In Word Vectors



COALS model from: An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence (Rohde et al., 2005)

Interesting Semantic Patterns Emerging In Word Vectors



COALS model from: An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence (Rohde et al., 2005)

Count Based Vs. Direct Prediction

- LSA, HAL (Lund & Burgess),
- COALS, Hellinger-PCA (Rohde et al, Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate influence given to large counts

- Skip-gram/CBOW (Mikolov et al)
- NNLM, HLBL, RNN (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton)

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

Method 2: GloVe (Pennington et al., EMNLP 2014) Encoding Meaning In Vector Differences

Crucial insight: Ratios of co-occurrence probabilities can encode meaning components (i.e., relationships of words)

Probe words

	x = solid	x = gas	x = water	x = random
$P(x \mid \text{ice})$	large	small	large	small
$P(x \mid \text{steam})$	small	large	large	small
$\dfrac{P(x \mid \text{ice})}{P(x \mid \text{steam})}$	large	small	~1	~1

Method 2: GloVe (Pennington et al., EMNLP 2014) Encoding Meaning In Vector Differences

Crucial insight: Ratios of co-occurrence probabilities can encode meaning components (i.e., relationships of words)

Probe words

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

Large

small

~1

Encoding Meaning In Vector Differences (GloVe)

Question: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?

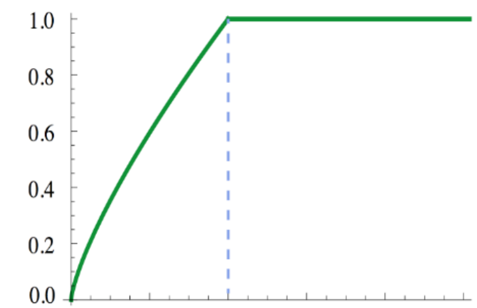
The loss function:

The weighting function

Number of times word j occur in the context of word i

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

$f \sim$



Advantages

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus and small vectors

Word2Vec and GloVe are very popular in NLP now. Which one is better depends on their specific applications.

GloVe Results

Nearest words to frog:
- frogs
- toad
- litoria
- leptodactylidae
- rana
- lizard
- eleutherodactylus

litoria

leptodactylidae

rana

eleutherodactylus

How To Evaluate Word Vectors?

Related to general evaluation in NLP: Intrinsic vs extrinsic

Intrinsic:

- Evaluation on a specific/intermediate subtask
- Fast to compute
- Helps to understand that system
- Unclear if really helpful unless correlation to real tasks is established

Extrinsic:

- Evaluation on a real task (things that we will study in this class)
- Can take a long time to evaluate the accuracy
- If a problem occurs, unclear if it is due to the word vectors, the system for the real task, or their interactions
- If replacing exactly one system for the real task with another improves accuracy -> great!

Intrinsic Word Vector Evaluation

Word Vector Analogies

a:b :: c:?

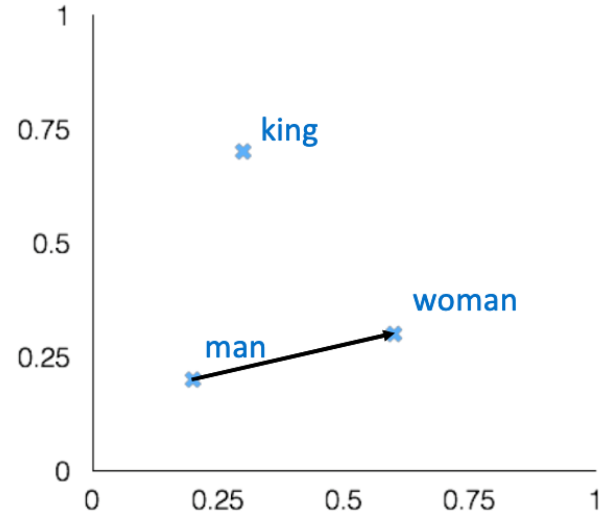
$$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}$$

man:woman :: king:?

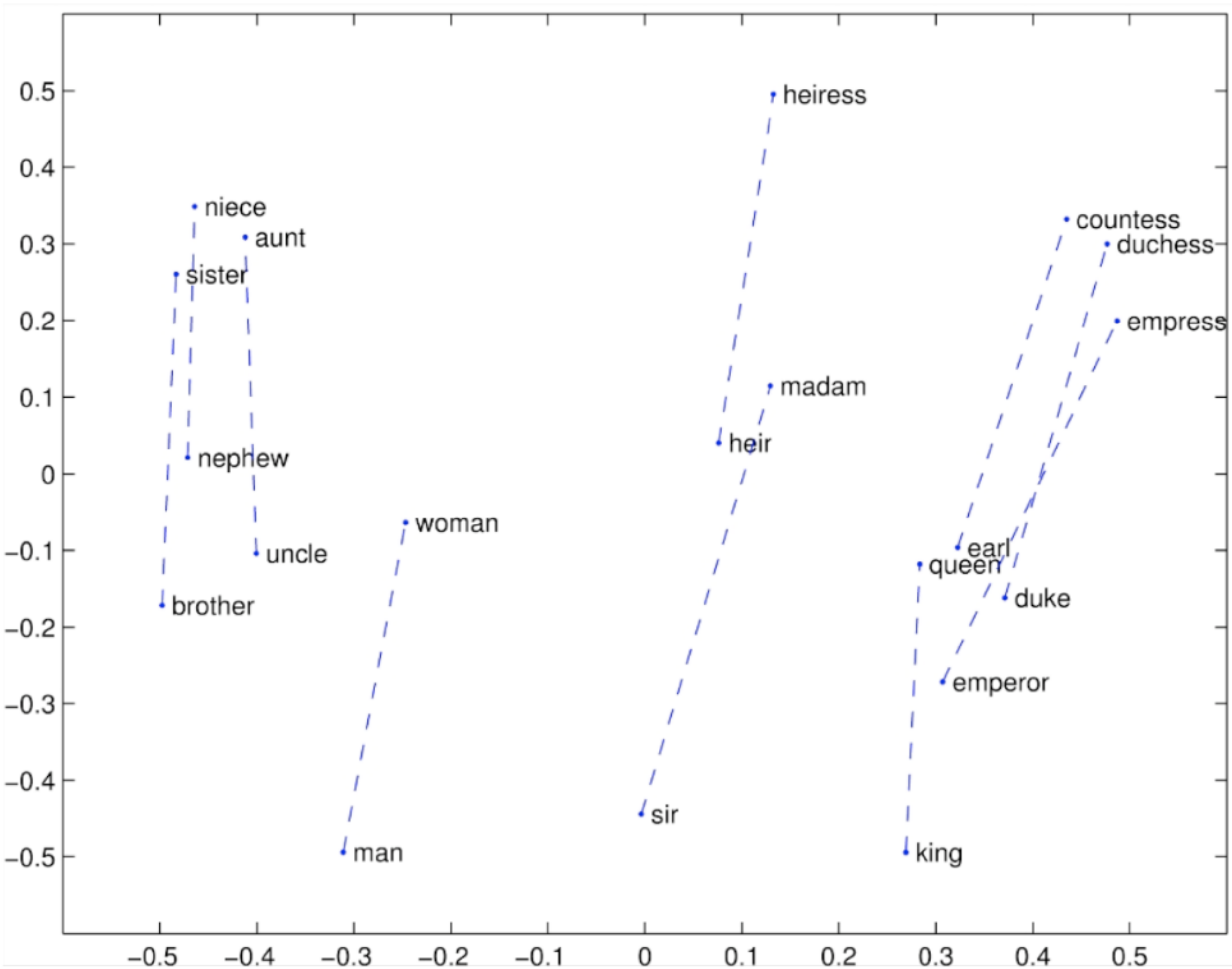
Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions

Discarding the input words from the search! $\arg\max_{x \in V \setminus \{king, man, woman\}} cos(x, king - man + woman)$

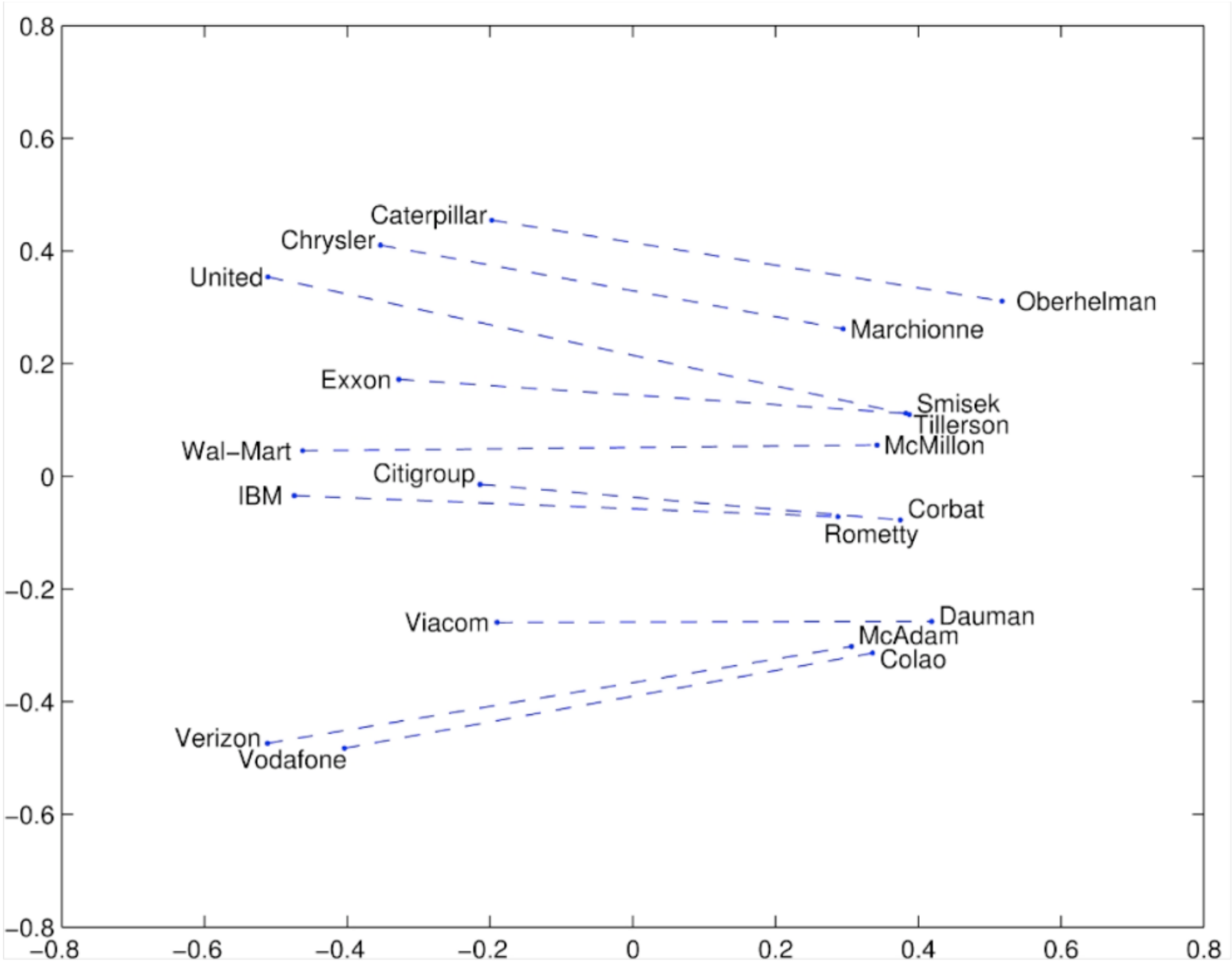
Problem: What if the information is there but not linear?



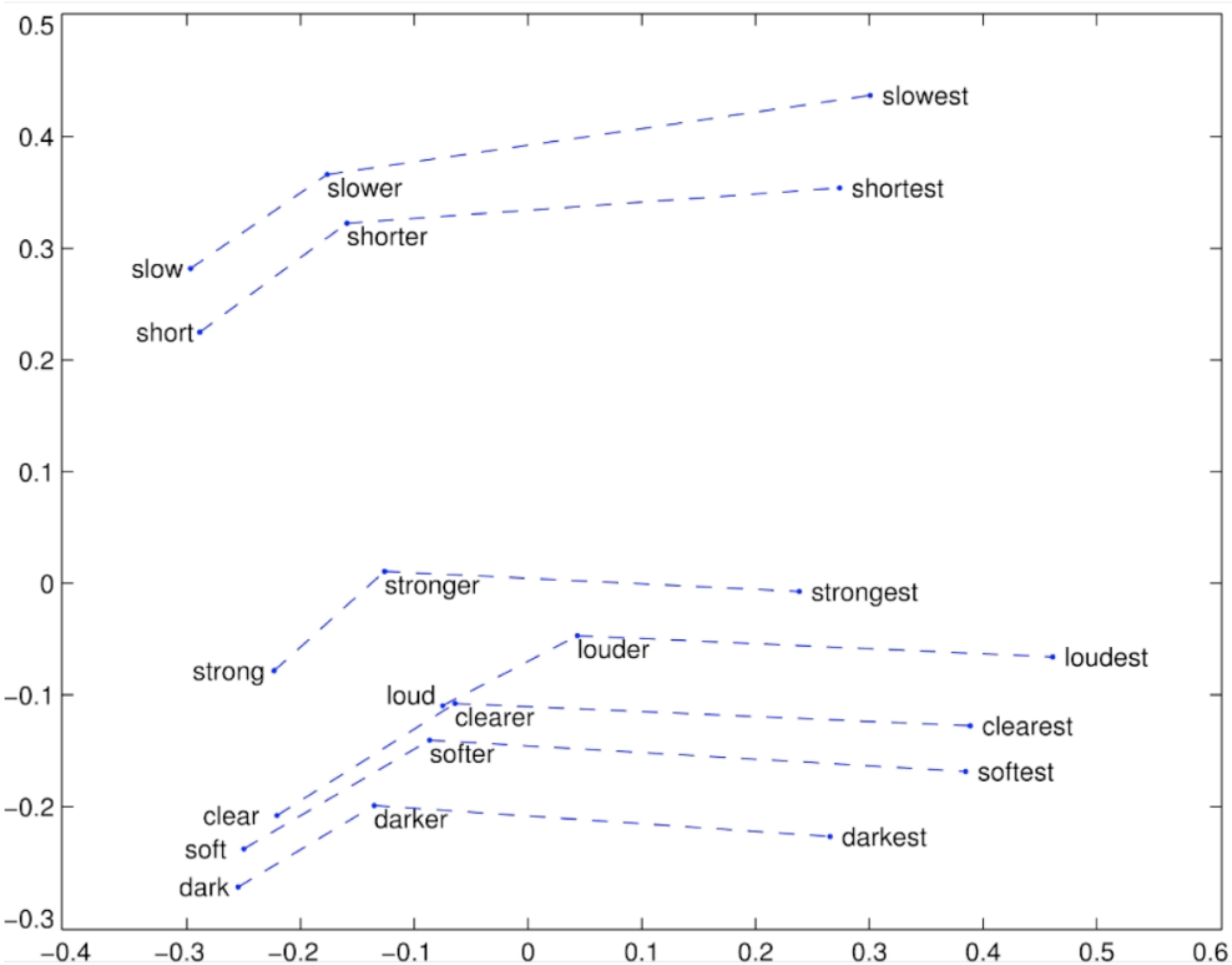
GloVe Visualization



GloVe Visualization: Company - CEO



GloVe Visualization: Comparative & Superlative



Intrinsic Word Vector Evaluation

Word Vector Analogies: Syntactic and Semantic examples from:

https://code.google.com/archive/p/word2vec/source

: gram4-superlative
bad worst big biggest
bad worst bright brightest
bad worst cold coldest
bad worst cool coolest
bad worst dark darkest
bad worst easy easiest
bad worst fast fastest
bad worst good best
bad worst great greatest

: city-in-state
Chicago Illinois Houston Texas
Chicago Illinois Philadelphia Pennsylvania
Chicago Illinois Phoenix Arizona
Chicago Illinois Dallas Texas
Chicago Illinois Jacksonville Florida
Chicago Illinois Indianapolis Indiana
Chicago Illinois Austin Texas
Chicago Illinois Detroit Michigan
Chicago Illinois Memphis Tennessee
Chicago Illinois Boston Massachusetts

Analogy Evaluation And Hyperparameters

Accuracy

Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	67.5	54.3	60.3
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	64.8	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	80.8	61.5	70.3
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	67.4	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	**81.9**	**69.3**	**75.0**

Another Intrinsic Word Vector Evaluation

Word vector distances and their correlation with human judgments

- Humans estimate the *relatedness* of the words in pairs on a scale from 0 (totally unrelated words) to 10 (very much related or identical words).

Example dataset: WordSim353
http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Correlation Evaluation

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW†	6B	57.2	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	**75.9**	**83.6**	**82.9**	**59.6**	**47.8**
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

Extrinsic Word Vector Evaluation

Extrinsic evaluation of word vectors: All subsequent tasks in this class

One example where good word vectors should help directly is Named Entity Recognition (i.e., finding names of persons, organization, or locations in text)

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	**88.7**	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	**93.2**	88.3	**82.9**	**82.2**

Word vectors/representations have been a major breakthrough in NLP in the last few years, enabling a novel approach for NLP based on deep learning, and leading to a new era for NLP with models of better performance, robustness and portability.

We will study a new generation of word vectors in a later class.