

CS 2731

Introduction to Natural Language Processing

Session 7: Vector semantics, word2vec

Michael Miller Yoder

September 18, 2024

Course logistics

- [Homework 1](#) **due tomorrow, Thu Sep 19**
 - Feel free to ask questions in the Canvas discussion forum, email Michael, schedule office hours
- [Project idea submission form](#) **due this Fri Sep 20**
 - All students must submit one idea for credit
 - You can submit one of the example project ideas from the [website](#)
 - Ideas do not need to be fully developed
- Discussion forum **due Mon Sep 23 at 1pm (before class)**
 - Is posted on Canvas
- HW2 will be released by tomorrow

Lecture overview: vector semantics, static word embeddings

- Group activity: clickbait classification with Naïve Bayes
- Vector semantics
- Distributional semantics
- Types of word vectors
- Word2vec
- Bias in word vectors

Group activity: text classification

Clickbait classification code walk-through

- What steps are needed to go from labeled text data to a classifier?
 - Load in data, matched with labels
 - Preprocessing (tokenize, remove punctuation? lowercase?)
 - Extract features (bag of words, etc)
 - Train classifier
 - Test classifier
 - Interpret classifier
- Colab notebook here: <https://colab.research.google.com/drive/1J-FcHTFYXTsNgcEB19kQcqJnE8CsbSe7?usp=sharing>

A brief history of NLP



Sentences in Russian are punched into standard cards for feeding into the electronic data processing machine for translation into English

*The Georgetown-IBM Experiment.
Credit: John Hutchins*

- 1950s: **foundations**
 - Turing Test ("Computing Machinery and Intelligence" paper)
 - Georgetown-IBM Experiment translating Russian to English
- 1960s-1980s: **symbolic reasoning**
 - ELIZA, rule-based parsing, hand-built conceptual ontologies
- 1990s-2010s: **statistical NLP**
 - Learn patterns from large corpora (feature-based machine learning)
- 2000s-today: **neural NLP**
 - SOTA on many tasks from "deep" layers of neural networks

Vector semantics

Semantics: the study of meaning

Word representations in NLP draw on 2 areas of semantics

- a. Vector semantics
- b. Distributional semantics

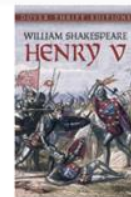
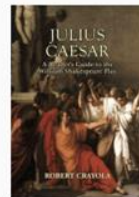
Vector semantics

Modeling semantics as points in vector space

- Words or other text segments are represented by vectors
- Multiple dimensions
- Nearer = more similar words

Term-document matrix: word vectors

Two words are similar if their vectors are similar.



	As You Like It	Twelfth Night	Julius Caesar	Henry V
<i>battle</i>	1	1	8	15
<i>soldier</i>	2	2	12	36
<i>fool</i>	37	58	1	5
<i>clown</i>	6	117	0	0

Pairs of similar words?

Similarity and relatedness

- Synonyms: big/large, couch/sofa, automobile/car
- Similar: sharing some element of meaning
 - coffee/tea, car/bicycle, cow/horse
- Related: by a semantic field
 - coffee/cup, scalpel/surgeon



Distributional semantics

Distributional semantics

"The meaning of a word is its use in the language" [Wittgenstein 1953]



"You shall know a word by the company it keeps" [Firth 1957]



"If A and B have almost identical environments we say that they are synonyms" [Harris 1954]



Distributional semantics

Define the meaning of a word by its **distribution in language use**: its neighboring words or grammatical environments.

You Learn Words by Using Distributional Similarity



Consider

- A bottle of **pocarisweat** is on the table.
- Everybody likes **pocarisweat**.
- **Pocarisweat** makes you feel refreshed.
- They make **pocarisweat** out of ginger.

What does *pocarisweat* mean?

You Know Pocarisweat by the Company It Keeps



From context words humans can guess *pocarisweat* means a beverage like **coke**.

How do you know?

- Other words can occur in the same context
- Those other words are often for beverages (that you drink cold)
- You assume that *pocarisweat* is probably similar

So the intuition is that **two words are similar if they have similar word contexts**.

Sample Contexts of ± 7 Words

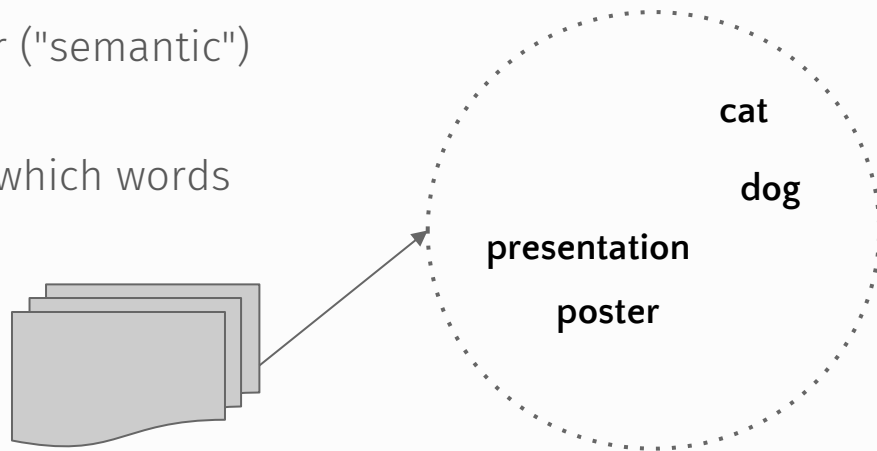
sugar, a sliced lemon, a tablespoonful of their enjoyment. Cautiously she sampled her first well suited to programming on the digital for the purpose of gathering data and **apricot** **pineapple** **computer.** **information** preserve or jam, a pinch each of, and another fruit whose taste she likened In finding the optimal R-stage policy from necessary for the study authorized in the

	aardvark	computer	data	pinch	result	sugar ...
\vdots						
<i>apricot</i>	0	0	0	1	0	1
<i>pineapple</i>	0	0	0	1	0	1
<i>digital</i>	0	2	1	0	1	0
<i>information</i>	0	1	6	0	4	0
\vdots						

Types of word vectors

Shared Intuition: Words are Vectors of Numbers Representing Meaning

- Model the meaning of a word by “**embedding**” it in a vector space.
- The meaning of a word is a vector of numbers:
 - Vector models are also called **embeddings**
 - Often, the word *embedding* is reserved for *dense* vector representations
- In contrast, word meaning is represented in many (early) NLP applications by a vocabulary index (“word number 545”; compare to **one-hot representations**)
- Similar words are nearby in vector (“semantic”) space
- Build “semantic space” by seeing which words are nearby in text



There are Two Kinds of Vector Models

- **Sparse embeddings** (vectors from term-document matrix)
 - long (length of 20,000 to 50,000)
 - sparse: most elements are 0
- **Dense embeddings** (Word2vec)
 - short (length of 50-1000)
 - dense (most elements are non-zero)

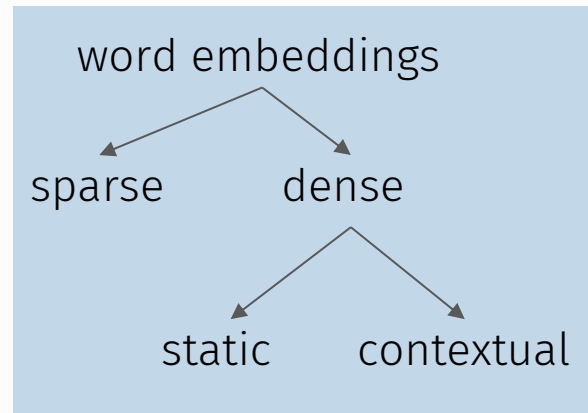


Dense Vectors Have Three Advantages over Sparse Vectors

1. Short vectors may be **easier to use as features** in machine learning (less weights to tune).
2. Dense vectors may **generalize better** than storing explicit counts.
3. They may do **better at capturing synonymy**:
 - *car* and *automobile* are synonyms
 - But, in sparse vectors, they are represented as distinct dimensions
 - This fails to capture similarity between a word with *car* as a **neighbor** and a word with *automobile* as a **neighbor**

Methods for learning short, dense word embeddings

- Static, neural embeddings
 - Fixed embeddings for word types
 - Word2Vec, GloVe
- Contextual embeddings
 - Embeddings for words vary by context
 - ELMo, BERT, LLMs



Word2vec

Word2vec [Mikolov et al. 2013]


- Instead of counting words, train a classifier on a binary prediction task
 - Is w_1 likely to show up near w_2 ?

Word2vec [Mikolov et al. 2013]


- Instead of counting words, train a classifier on a binary prediction task
 - Is w_1 likely to show up near *apricot*?



Word2vec [Mikolov et al. 2013]

- Instead of counting words, train a classifier on a binary prediction task
 - Is w_1 likely to show up near *apricot*? 
- Take the learned classifier weights as the word embeddings

Word2vec [Mikolov et al. 2013]

- Instead of counting words, train a classifier on a binary prediction task
 - Is w_1 likely to show up near *apricot*? 
- Take the learned classifier weights as the word embeddings
- Training techniques: skip-gram and CBOW

Word2vec: training supervision

- **Self-supervision** [Bengio et al. 2003, Collobert et al. 2011]
- Use naturally occurring text as labels
- A word c that occurs near *apricot* in the corpus counts as the gold "correct answer" for supervised learning

Word2vec training overview

1. Positive examples: the target word w and a neighboring context word c_{pos}
2. Negative examples: Randomly sample other words c_{neg} in the lexicon to pair with w
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights (W, C) as the word embeddings

Training for Embeddings

- We do not know what W and C are. So we learn them through an iterative process.
- We use a large corpus as a training data
- We also randomly sample the corpus to find words that are NOT in the context—negative sampling.



Positive Examples		Negative Examples			
t	c	t	c	t	c
ides	beware	ides	aardvark	ides	twelve
ides	of	ides	puddle	ides	hello
ides	March	ides	where	ides	dear
ides	the	ides	coaxial	ides	forever

Word2vec: learning embeddings

- Start with randomly initialized context C and target word W matrices
- Go through the positive and negative training pairs, adjusting word vectors such that we:
 - Maximize the similarity of the target word, context word pairs (w, c_{pos}) drawn from the positive data
 - Minimize the similarity of the (w, c_{neg}) pairs drawn from the negative data.

Skip-gram classifier

Classifier input pairs:

(target word w , context word c)

Classifier output: probabilities that w occurs with c

$$P(+|w, c)$$

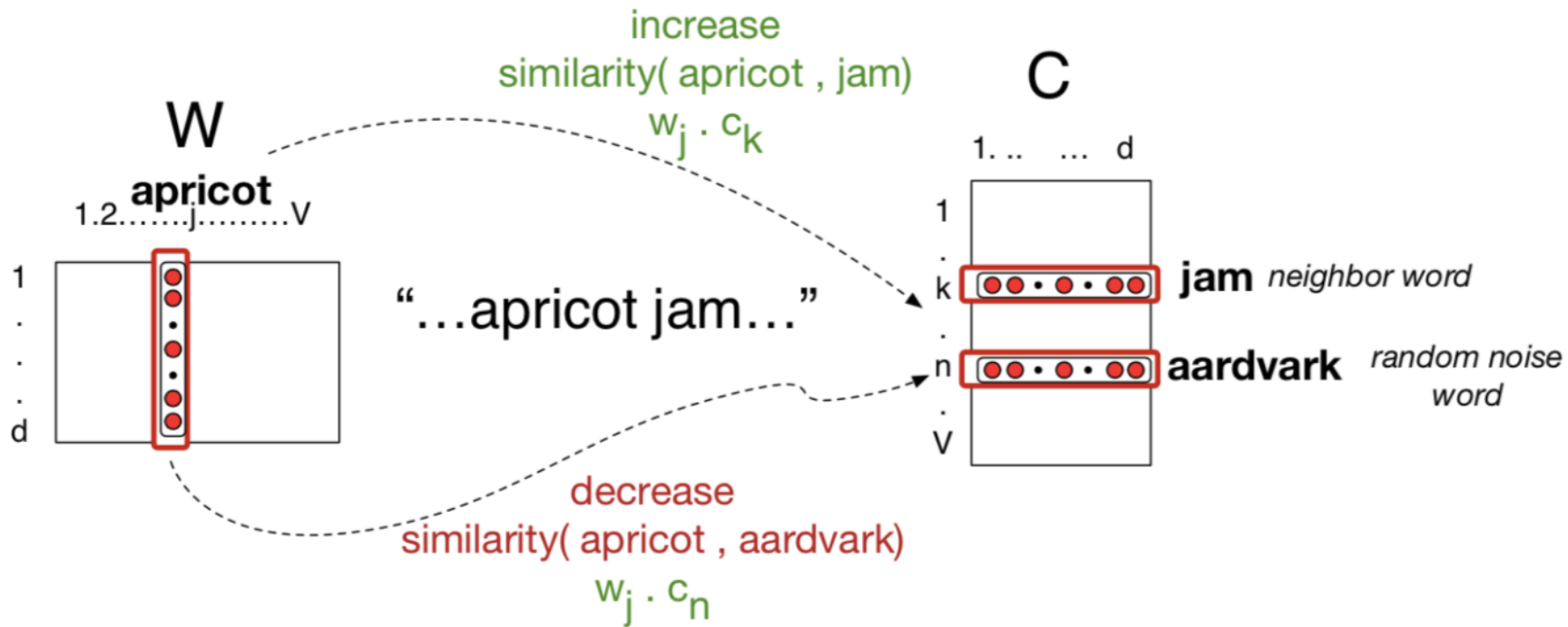
$$P(-|w, c) = 1 - P(+|w, c)$$

Skip-gram classifier: calculating probabilities

- From input vectors, need to compare for similarity
- Start with dot product: $\text{sim}(\mathbf{w}, \mathbf{c}) \approx \mathbf{w} \cdot \mathbf{c}$
- To turn this into a probability, use the sigmoid function from logistic regression:

$$P(+|\mathbf{w}, \mathbf{c}) = \sigma(\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{c} \cdot \mathbf{w})}$$

Training for Embeddings



Reminder: one step of gradient descent

- Direction: We move in the reverse direction from the gradient of the loss function
- Magnitude: we move the value of this gradient $d/dw L(P(+|w,c) + P(-|w,c))$ weighted by a learning rate η
- Higher learning rate means move w faster

Word2vec training process

Updates on C and W

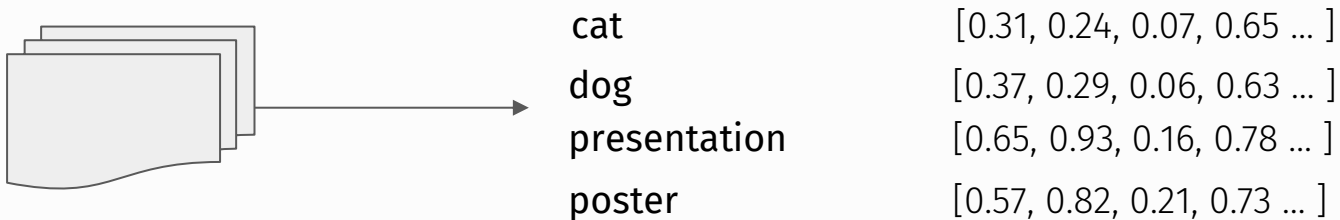
$$c_{pos}^{t+1} = c_{pos}^t - \eta [\sigma(c_{pos}^t \cdot w^t) - 1]$$

new context weights old context weights learning rate derivative of loss wrt c_{pos}

$$w^{t+1} = w^t - \eta \left[[\sigma(c_{pos} \cdot w^t) - 1] c_{pos} + [\sigma(c_{neg_i} \cdot w^t)] c_{neg_i} \right]$$

new target word weights old context weights learning rate derivative of loss wrt w

Summary: How to learn word2vec embeddings

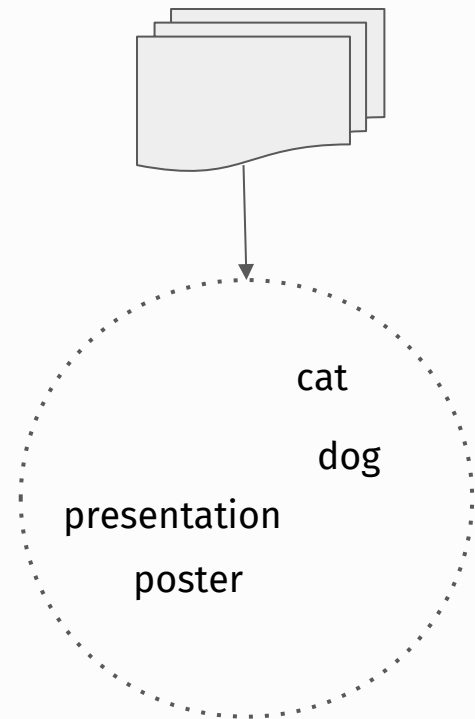


Summary: How to learn word2vec embeddings

1. Start with randomly initialized word embeddings
2. From a corpus, extract pairs of words that co-occur (positive)
3. Extract pairs of words that don't co-occur (negative)
4. Train a classifier to distinguish between positive and negative examples by slowly adjusting all the embeddings to improve the classifier performance
5. Keep the weights as our word embeddings

Final embeddings

- Can add representations for a word in W and in C together for final word vector for W_i
- Can just keep W and throw away C
- Can find "nearest neighbors" of certain words with cosine similarity in embedding space



There are Tools and Resources Available for Training and Using Embeddings

- **Pretrained embeddings**
 - Skip-gram
 - CBOW
 - fastText
 - GloVe
- **Training your own embeddings**
 - You can easily train skip-gram, CBOW, and fastText embeddings with `gensim`
 - Straightforward Python interface

Embeddings reflect cultural biases [Bolukbasi et al. 2016]

- Paris : France :: Tokyo : *Japan*
- Sexist occupational stereotypes
 - father : doctor :: mother : *nurse*
 - man : computer programmer :: woman : *homemaker*
- Would be problematic to use embeddings in hiring searches for programmers

Conclusion: vector semantics, static word embeddings

- NLP typically represents words as vectors in spaces where distance \approx semantic similarity
- Word2vec learns static embeddings (vectors) for words by predicting which words occur together in training data
- These embeddings are effective in downstream NLP tasks, but also reflect social biases of training data text

Questions?