# CS 2731
# Introduction to Natural Language Processing

## Session 10: Vector semantics, word2vec

Michael Miller Yoder

September 29, 2025

University of Pittsburgh | School of Computing and Information

# Course logistics

- Quiz in class **this Wed Oct 1.** Readings to review:
  - Session 9: J+M 4.5-4.8, 4.13, 4.16
  - Session 10: J+M 5-5.2, 5.5-5.8, 5.10
  - You will have 12 minutes to complete the quiz
- Homework 2 has been released. Is **due next Thu Oct 9**
  - Michael will post the Kaggle competition soon (probably tomorrow)
- Next project deliverable: project proposal due **Oct 16**
  - Michael will post the requirements soon
  - For now, focus on finding related literature and datasets
  - Finding out what evaluation metric to use may require looking at other chapters of the textbook
  - Feel free to email or book office hours with Michael to discuss

# Start running notebook: examine word2vec embeddings

- Click on this nbgitpuller link
  - Or find the link on the course website
- Open **session10_word2vec.ipynb**
- Run the first 2 cells while we go through slides, as they take awhile

# Overview: vector semantics, static word embeddings

- Vector semantics
- Distributional semantics
- Types of word vectors
- Word2vec
- Bias in word vectors
- Coding activity: explore word vectors

# Vector semantics

# Semantics: the study of meaning

Word representations in NLP draw on 2 areas of semantics

a. Vector semantics

b. Distributional semantics

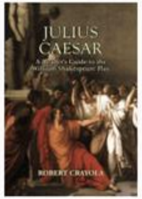
# Vector semantics

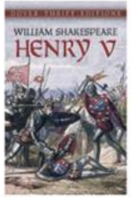
Modeling semantics as points in vector space

- 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 |
|---|---|---|---|---|
| *battle* | 1 | 1 | 8 | 15 |
| *soldier* | 2 | 2 | 12 | 36 |
| *fool* | 37 | 58 | 1 | 5 |
| *clown* | 6 | 117 | 0 | 0 |



*Slide adapted from David Mortensen, Jurafsky & Martin*

# 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



Slide adapted from Jurafsky & Martin

# 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]



*Slide adapted from Jurafsky & Martin*

# 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?



*Slide credit: David Mortensen*

# 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**.



*Slide credit: David Mortensen*

# Sample Contexts of $\pm 7$ Words

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

| | aardvark | computer | data | pinch | result | sugar ... |
|---|---|---|---|---|---|---|
| ⋮ | | | | | | |
| *apricot* | 0 | 0 | 0 | 1 | 0 | 1 |
| *pineapple* | 0 | 0 | 0 | 1 | 0 | 1 |
| *digital* | 0 | 2 | 1 | 0 | 1 | 0 |
| *information* | 0 | 1 | 6 | 0 | 4 | 0 |
| ⋮ | | | | | | |



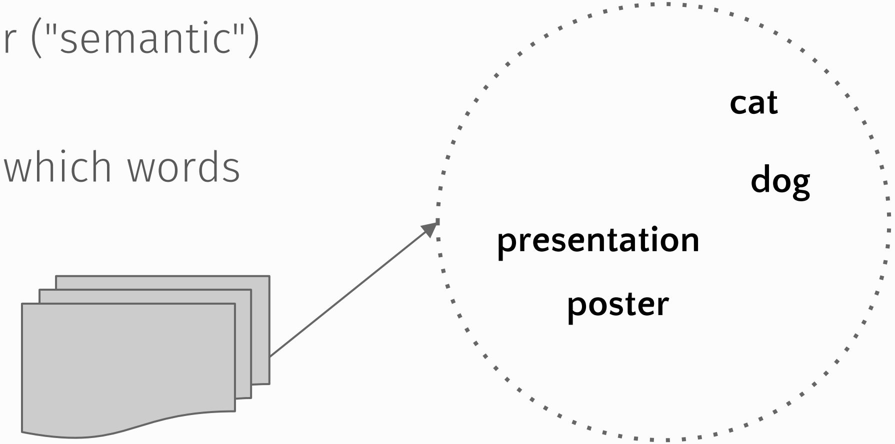
Slide credit: David Mortensen

# 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




*Slide adapted from David Mortensen*

# 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)



*Slide adapted from David Mortensen, Jurafksy & Martin*

# Dense Vectors Have Three Advantages over Sparse Vectors

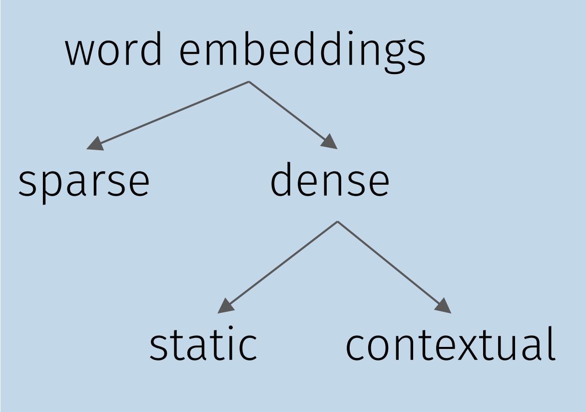
1. Short vectors may be **easier to use as features** in machine learning (less weights to tune).
2. Dense vectors may **generalize better** than storing explicit counts.
3. They may do **better at capturing synonymy**:
    - *car* and *automobile* are synonyms
    - But, in sparse vectors, they are 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



*Slide credit: David Mortensen*

# 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 $w_2$?



*Slide adapted from Jurafsky & Martin*

# 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*?



Slide adapted from Jurafsky & Martin

# 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



Slide adapted from Jurafsky & Martin

# 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



*Slide adapted from Jurafsky & Martin*

# 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

*Slide adapted from Jurafsky & Martin*

# 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



*Slide adapted from Jurafsky & Martin*

# 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.

| A | soothsayer | bids | you | beware | the | Ides | of | March | . |
|---|---|---|---|---|---|---|---|---|---|
| | | | | $c_1$ | $c_2$ | $t$ | $c_3$ | $c_4$ | |

| 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 |



*Slide credit: David Mortensen*

# 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.



*Slide adapted from Jurafsky & Martin*

# 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)



*Slide adapted from Jurafsky & Martin*

# Skip-gram classifier: calculating probabilities

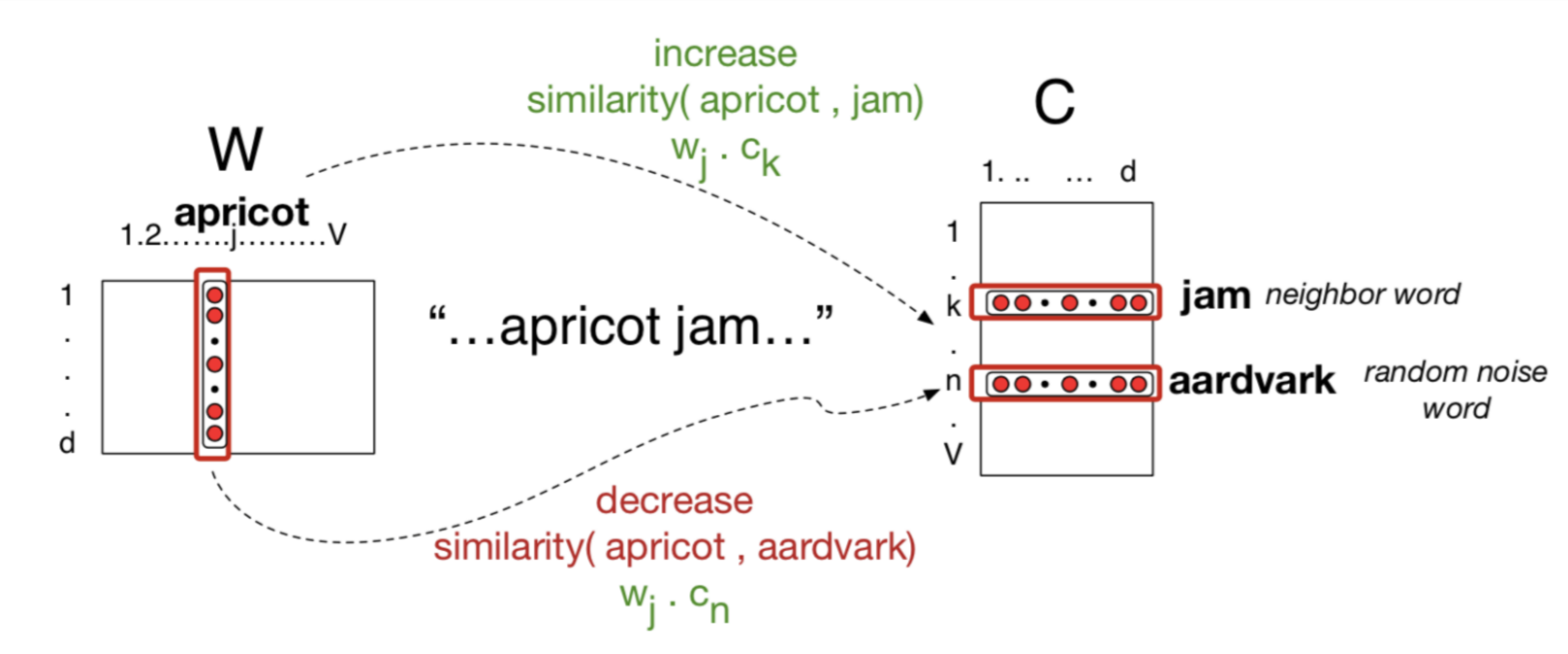
- From input vectors, need to compare for similarity
- Start with dot product: sim(**w**,**c**) ≈ **w** · **c**
- To turn this into a probability, use the sigmoid function from logistic regression:

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



*Slide adapted from Jurafsky & Martin*

# Training for Embeddings






Slide adapted from David Mortensen, Jurafsky & Martin

# 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 $\eta$
- Higher learning rate means move *w* faster



*Slide adapted from Jurafsky & Martin*

# Loss function for one $w$ with $c_{pos}$ , $c_{neg1}$ ...$c_{negk}$

Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the $k$ negative sampled non-neighbor words.

$$
\begin{aligned}
L_{\mathrm{CE}} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log\sigma(c_{pos}\cdot w)+\sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$



*Slide adapted from Jurafsky & Martin*

# Word2vec training process

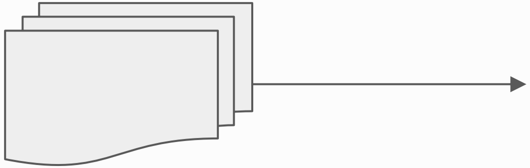
Updates on $C$ and $W$

$$\underbrace{c_{pos}^{t+1}}_{\text{new context weights}} = \underbrace{c_{pos}^{t}}_{\text{old context weights}} - \underbrace{\eta}_{\text{learning rate}} \underbrace{[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]}_{\text{derivative of loss wrt } c_{pos}}$$

$$\underbrace{w^{t+1}}_{\text{new target word weights}} = \underbrace{w^{t}}_{\text{old context weights}} - \underbrace{\eta}_{\text{learning rate}} \underbrace{\left[ [\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + [\sigma(c_{neg_i} \cdot w^{t})]c_{neg_i} \right]}_{\text{derivative of loss wrt } w}$$

*Slide adapted from Jurafsky & Martin*

# Summary: How to learn word2vec embeddings



| | |
|---|---|
| cat | [0.31, 0.24, 0.07, 0.65 … ] |
| dog | [0.37, 0.29, 0.06, 0.63 … ] |
| presentation | [0.65, 0.93, 0.16, 0.78 … ] |
| poster | [0.57, 0.82, 0.21, 0.73 … ] |

# Summary: How to learn word2vec embeddings

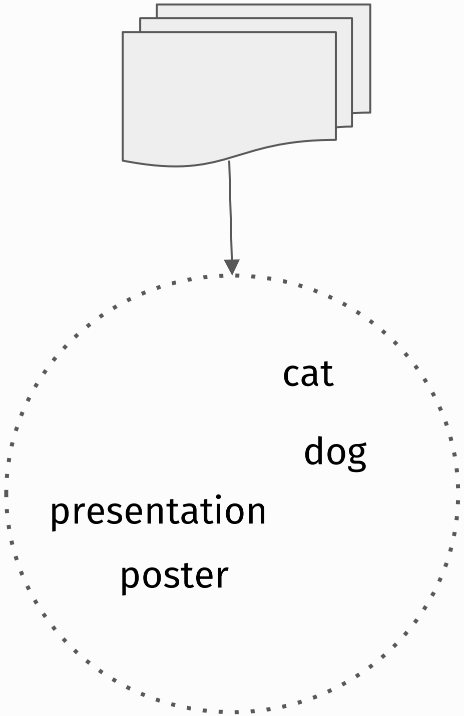
1. Start with randomly initialized word embeddings
2. From a corpus, extract pairs of words that co-occur (positive)
3. Extract pairs of words that don't co-occur (negative)
4. Train a classifier to distinguish between positive and negative examples by slowly adjusting all the embeddings to improve the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



*Slide adapted from Jurafsky & Martin*

# 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






Slide adapted from Jurafsky & Martin

# 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

*Slide credit: David Mortensen*

# 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



*Slide adapted from Jurafsky & Martin*

# Conclusion: vector semantics, static word embeddings

- NLP typically represents words as vectors in spaces where distance ≈ 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

# Coding activity

# Notebook: examine word2vec embeddings

- Click on this nbgitpuller link
  - Or find the link on the course website
- Open **session10_word2vec.ipynb**