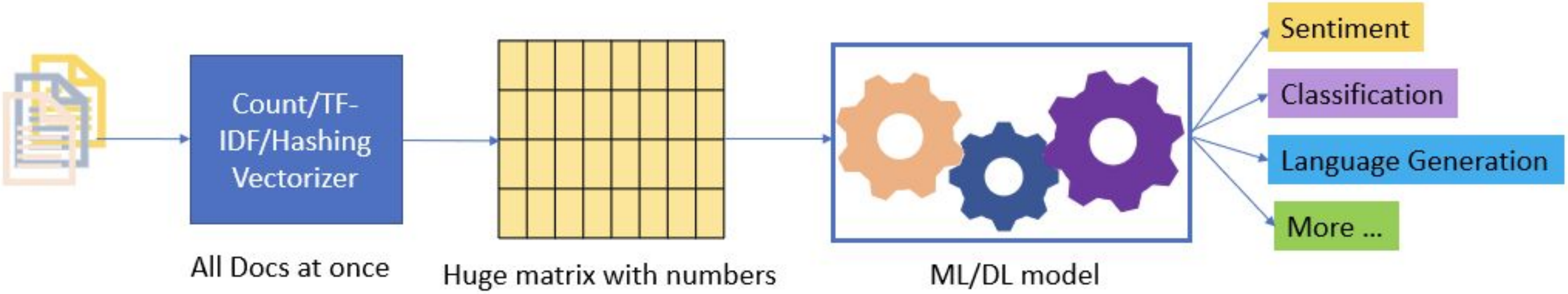
Word Embedding

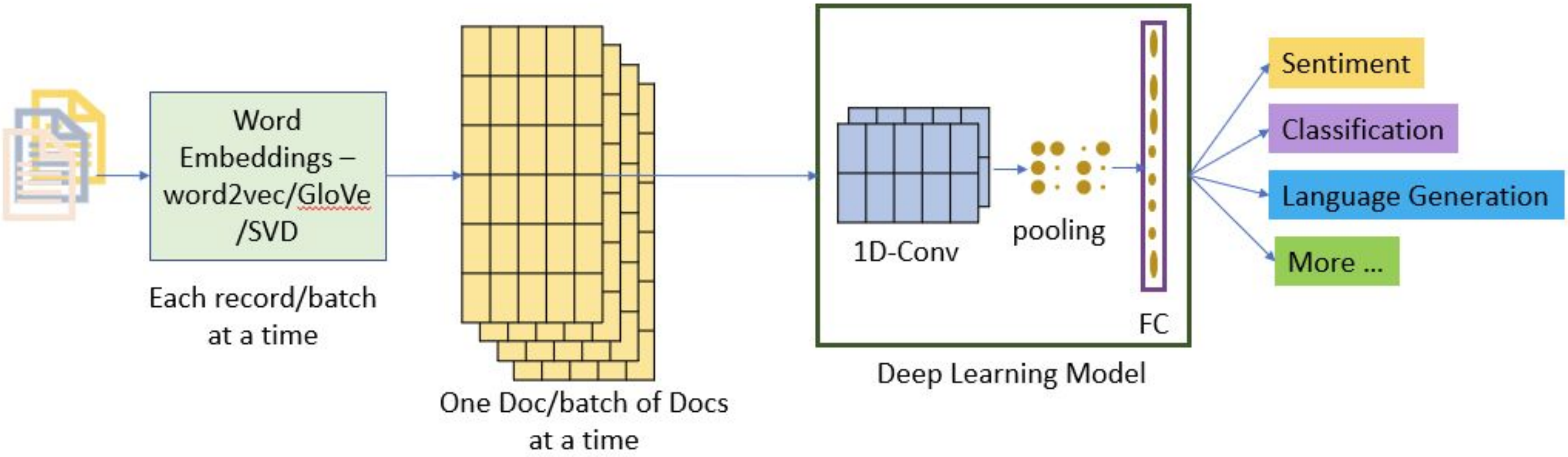
Transforming text into measurable data

Quick recap of previous lecture

Natural Language Processing Traditional Modules



ML model for TEXT – With Deep Learning



Building the Vector Spaces

- **Data Sources & Preprocessing**
  - Corpora, tokenization, normalization, lemmatization, stopword removal
- **Weighting Schemes** [*give more weight to surprising events and less weight to expected events*]
  - TF-IDF, PMI (Pointwise Mutual Information)
- **Matrix Smoothing & Dimensionality Reduction**
  - Motivation: noise reduction, uncover latent structure
  - SVD, PCA
- **Similarity Measures**
  - Cosine similarity, Euclidean distance, correlation

Limitations & Extensions

- **Limits of Classical VSM**
  - Ignores word order, syntax, ambiguity resolution
- **From VSM to Word Embeddings**
  - Connection to Word2Vec, GloVe
  - Neural embeddings learn similar vector spaces
- **Beyond Words: Sentence & Document Embeddings**
  - Averaging, Paragraph Vector, transformer encoders (SBERT)

Dense Word Embedding

Sparse (VSM) vs Dense embeddings

- **Matrix Types in VSMs**
  - Term–Document, Word–Context, Pair–Pattern
- **Weighting Schemes** [*give more weight to surprising events and less weight to expected events*]
  - TF-IDF, PMI (Pointwise Mutual Information)
  - Long (|V| > 20K)
  - Sparse (most elements are zero)
- **Dense vectors**
  - Short (50 - 1000) [*did not have a clear interpretation*]
  - Dense (most elements are non-zero)

Why short dense vectors?

- Dense embeddings convert words into shorter vectors, resulting in fewer parameters for models to learn. This leads to faster training and helps prevent overfitting. [**Efficient Feature Representation**]
- Dense vectors capture patterns and similarities in the data, allowing the model to **better generalize** beyond explicit word counts and rare combinations.
- **Synonymy Handling:**
  - Dense representations can place synonyms (like "car" and "automobile") close together in the vector space, even if they rarely share neighbors in explicit count-based models.
  - This allows words that appear near either "car" or "automobile" to be considered semantically similar, capturing real-world language relationships.
- In practice, low-dimensional vectors consistently outperform sparse, count-based features in most NLP tasks. [It is still not 100% clear why, but the weight of evidence is strong]

Dense embeddings methods

- **Word2vec** https://code.google.com/archive/p/word2vec/
- Fasttext http://www.fasttext.cc/
- Glove http://nlp.stanford.edu/projects/glove/

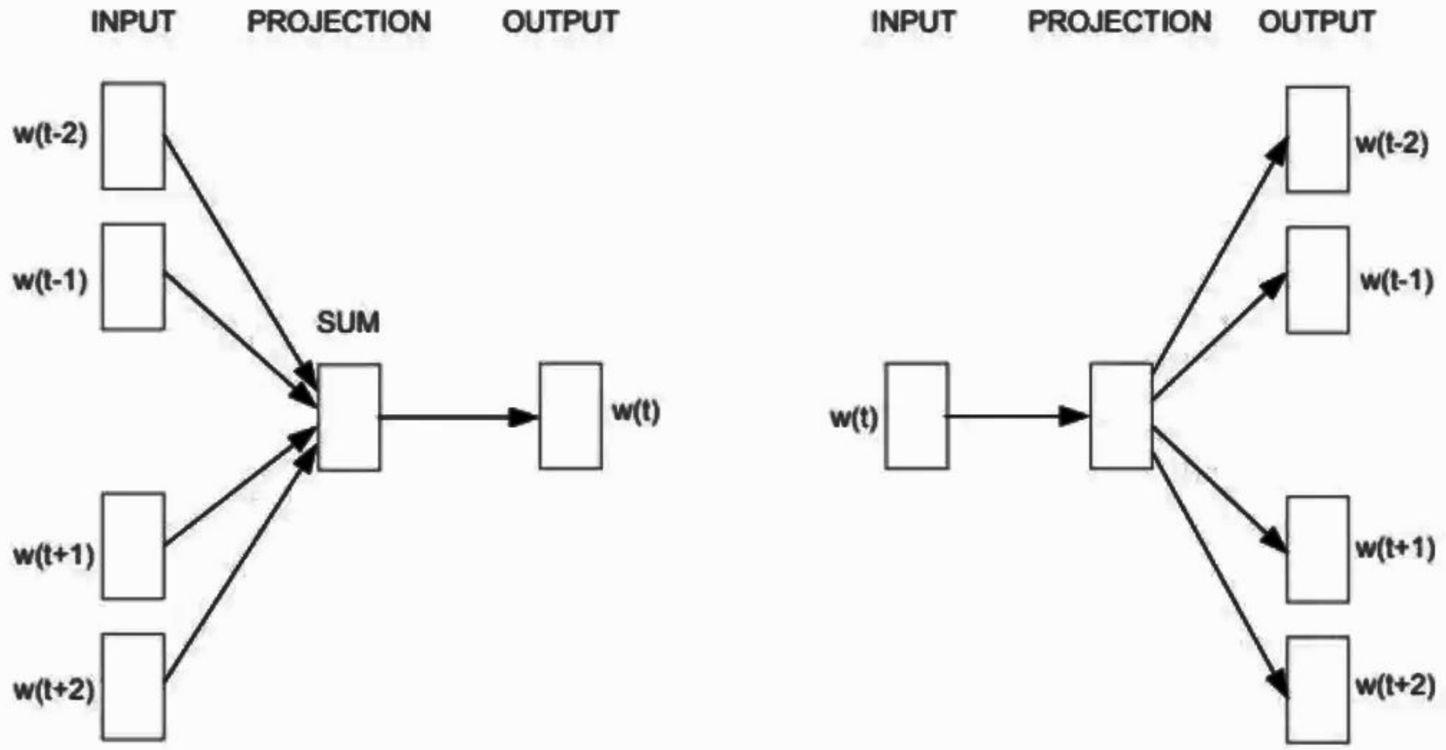
Word2Vec

- Popular embedding methods (Skip-gram, CBOW)
- Very fast to train
- Key idea: **predict rather than count**
- Code available on the web
    - Pytorch: https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html
    - Gensim: https://radimrehurek.com/gensim/models/word2vec.html

Paper:

- Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems* 26 (2013).
- Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* (2013).

Word2Vec



CBOW

Skip-gram

Cite: Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* (2013).

Word2Vec

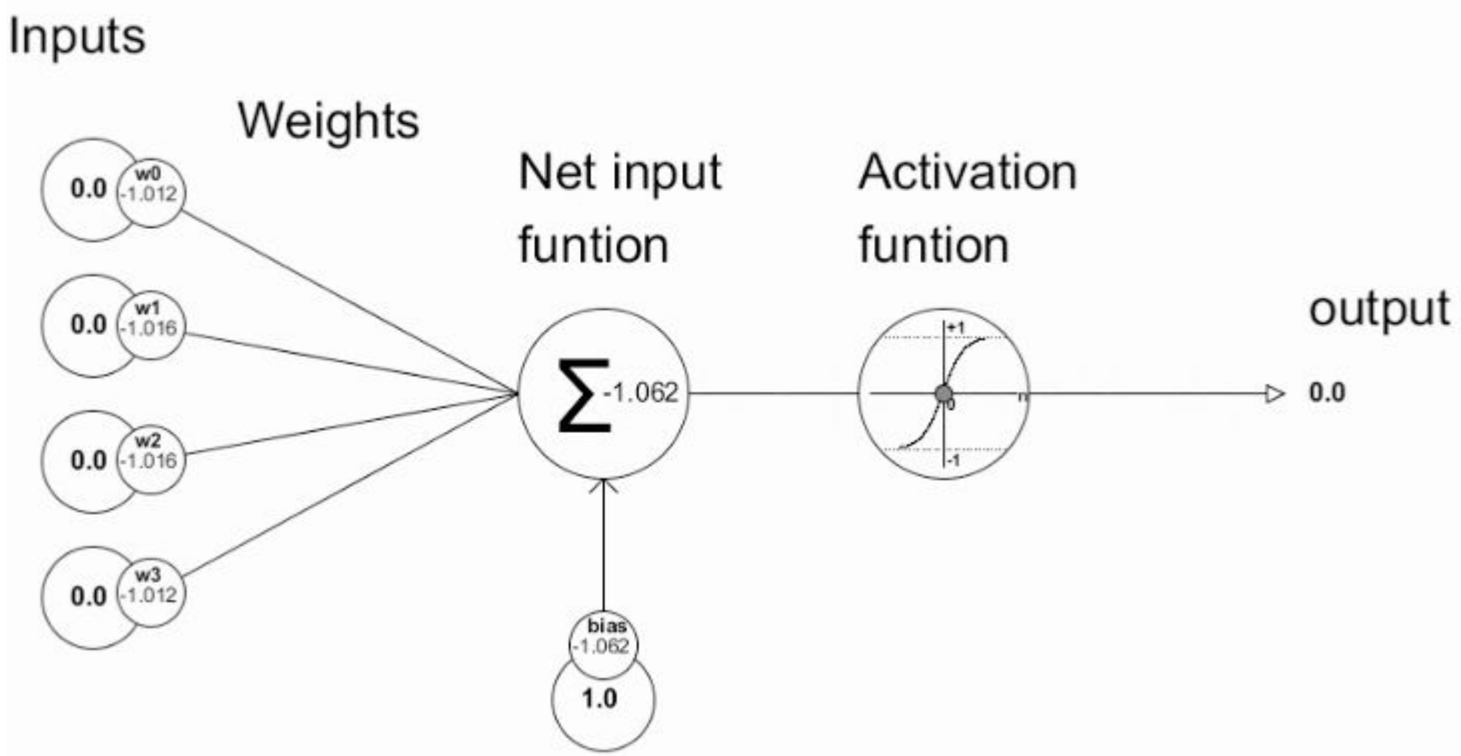
- Word2Vec generates **static embeddings**: each word has a single, fixed vector.
- Unlike **contextual models** (e.g., BERT), Word2Vec's embeddings do not change with context.
- Uses **self-supervision:**
  - Trains a binary classifier to predict if word A appears near word B.
  - Requires no labeled data—works on any unlabeled corpus.
- The focus is on the **learned embeddings**, not the classifier itself.

Training Word2Vec model

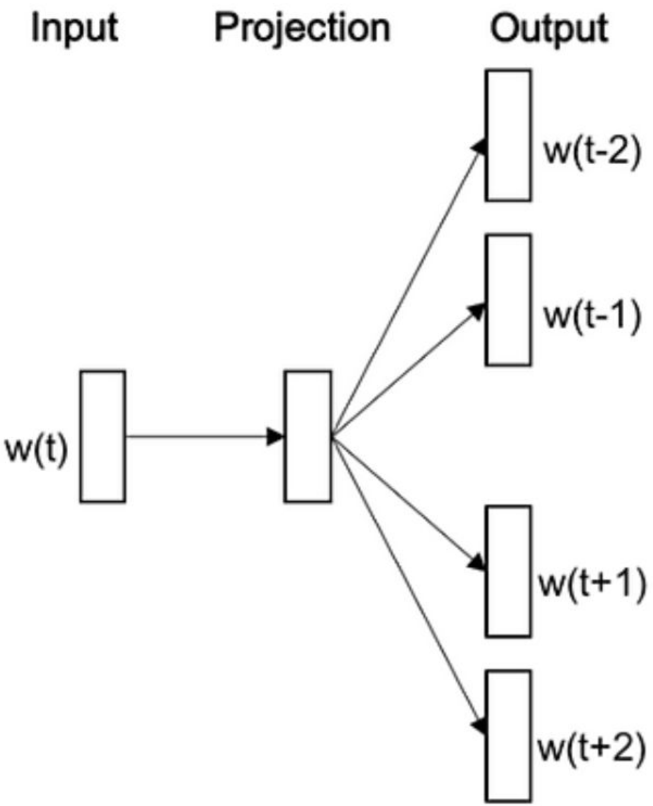
- Running text provides implicitly supervised training data!

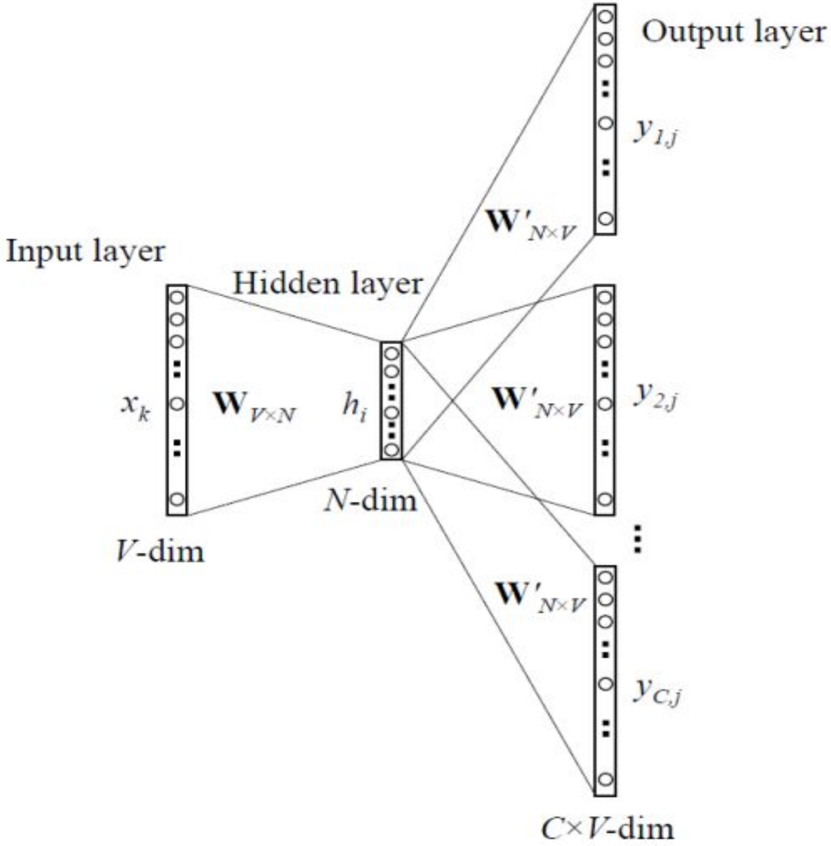
```
... lemon,   a [tablespoon of apricot jam,       a] pinch ...
               c1          c2     w      c3       c4
```

- A word context c near apricot acts as gold ‘correct answer’ to the question
  - “Is word c likely to show up near apricot?”
- No need for hand-labeled supervision
- The idea comes from neural language modeling
  - Bengio et al. (2003) ["A Neural Probabilistic Language Model"]
  - Collobert et al. (2011) ["Natural Language Processing (Almost) from Scratch"]


Inputs
Weights
Net input
funtion
Activation
funtion
output
0.0
w0
-1.012
0.0
w1
-1.016
0.0
w2
-1.016
0.0
w3
-1.012
-1.062
bias
-1.062
1.0
+1
0
-1
0.0

Word2Vec (Skip-gram model)





One hot encoding

- V = [a, aardvark, enjoy, home, I, lectures, love, mangos, NLP, zerbra]
- Index vocabulary with UNK for Out Of Vocabulary (OOV)
  - [a, aardvark, enjoy, home, I, lectures, love, mangos, NLP, zerbra, UNK]
    0 1 2 3 4 5 6 7 8 9 10

- |V| = 11

One hot encoding

- Vocabulary, V =
  [a, aardvark, enjoy, home, I, lectures, love, mangos, NLP, zerbra, UNK]
  0 1 2 3 4 5 6 7 8 9 10

- Sentence = I love NLP yeah

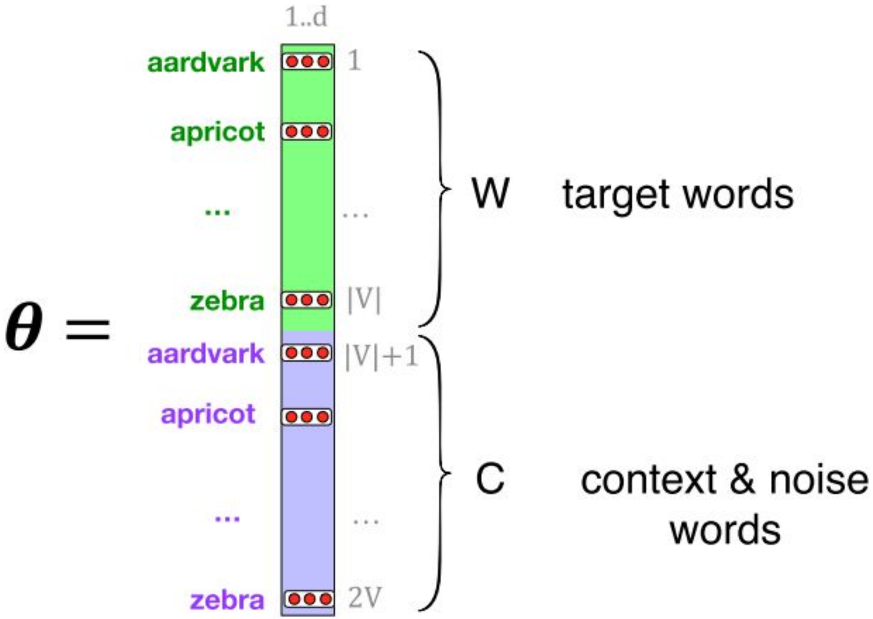
One-hot encoding for token == I
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0] == I

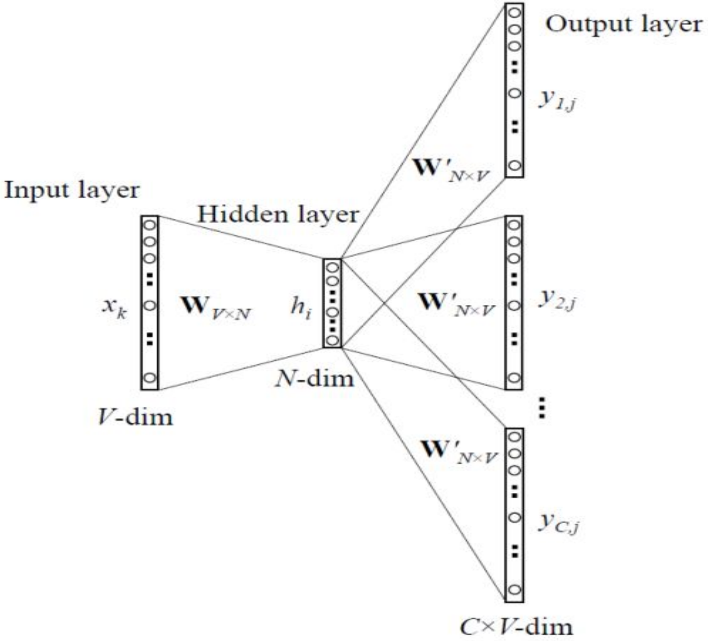
One-hot encoding for other tokens = love NLP yeah
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0] == love
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0] == NLP
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1] == yeah == UNK

Encoded sequence for "I love NLP yeah"
[[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]

Word2Vec (Skip-gram model)

- Skip-gram model stores a target embedding matrix W for target words and a context embedding matrix C for context and noise words
- Embedding matrices have a dimension d whose size is found empirically





Skip-gram Approach

- Input is a target word **w** and a window of **L** words to make some context words **c**

```
... lemon,   a [tablespoon of apricot jam,      a] pinch ...
               c1          c2     w      c3         c4
```

- Instead of counting how often each word c occurs near "apricot“
- Train a classifier on a binary prediction task:

    Is c likely to show up near "apricot"?

- P(+|w,c) = probability c is a context word for w
- P(-|w,c) = probability c is not a context word for w

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

Note: we don’t actually care about this task!

- But we'll take the learned classifier weights as the word embeddings

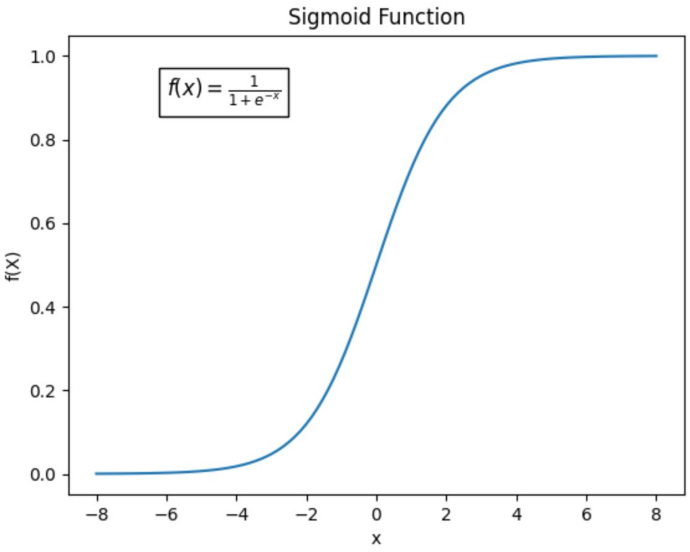
Word2Vec

- Probability is based on embedding vector similarity of w and c
    - Dot product + logistic function σ to make it into a probability

$$Similarity(w,c) \approx \mathbf{c} \cdot \mathbf{w} \qquad \sigma(x) = \frac{1}{1+\exp(-x)}$$

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

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

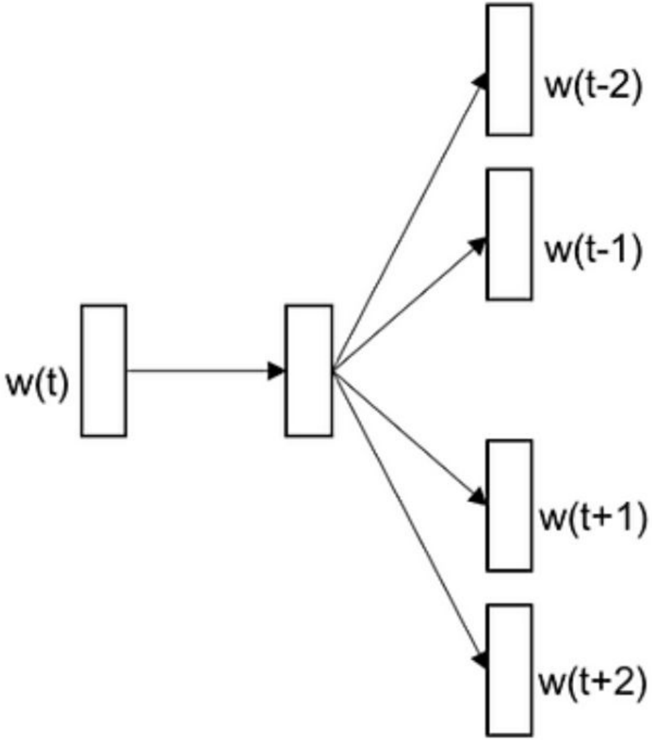


Word2Vec (Skip-gram model)

- Simplifying assumption that all context words are independent allows us to just multiple the probabilities for all of window L

$$P(+|w, c_{1:L}) = \prod_{i=1}^{L} \sigma(\mathbf{c_i} \cdot \mathbf{w})$$

$$\log P(+|w, c_{1:L}) = \sum_{i=1}^{L} \log \sigma(\mathbf{c_i} \cdot \mathbf{w})$$



Skip-Gram Training

- Assume context words are those in +/- 2 word window

```
... lemon,   a [tablespoon of apricot jam,      a] pinch ...
               c1          c2     w      c3       c4
```

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

P(+|t,c)

For each positive example,
we'll create k negative examples.

- Using noise words
- Any random word that isn't *t*

Skip-Gram Training

- Assume context words are those in +/- 2 word window

```
... lemon,  a [tablespoon of apricot jam,     a] pinch ...
               c1          c2     w     c3       c4
```

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

negative examples - k=2

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

P(+|t,c)

P(-|t,c)

Choosing noise words

- To avoid a strong bias towards common words for noise an α weighted unigram sample frequency is used to select noise words

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$$

- α= ¾ works well because it gives rare noise words slightly higher probability
  - imagine two events p(a)=.99 and p(b) = .01

$$P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = 0.97$$

$$P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = 0.03$$

Skip-gram: training set-up

- Each word is assigned a randomly initialized vector (e.g., 300 dimensions).
  - Total parameters: vocabulary size V × embedding size (e.g., 300 × V).
- Training updates vectors so words in similar contexts get similar embeddings.
  - A key part of the setup is selecting an appropriate loss function that guides which word vectors should become closer or further apart.
  - Embeddings are learned by minimizing a loss function using stochastic gradient descent (SGD).
- The Skip-gram objective: maximize the likelihood of actual context words for each target word.

Skip-gram: training objective

- **Motivation**: Over the entire training set, the goal is to learn word vectors that capture meaningful semantic relationships.
  - For each positive target and context word pairs (t,c), we want to maximize the similarity. (high dot product)
  - For each random negative target and context words pairs (t,c), we want to minimize the similarity. (low dot product)

- **Objective**: we want to maximize the probability that positive pairs are correctly labeled, and negative pairs are correctly rejected – using a loss function based on the sigmoid of the dot product.

$$\sum_{(t,c)\in +} logP(+|t,c) + \sum_{(t,c)\in -} logP(-|t,c)$$

Skip-gram: training objective

- Given target w, positive context c, (negative context c neg) x k
  - **Maximize** the similarity (score) for positive word-context pairs (label = +1).
  - **Minimize** the similarity (score) for negative word-context pairs (label = –1).

$$L = -\log\left[P(+|w, c_{pos})\prod_{i=1}^{k}P(-|w, c_{neg_i})\right]$$

<u>**Maximize**</u> <u>**Minimize**</u>

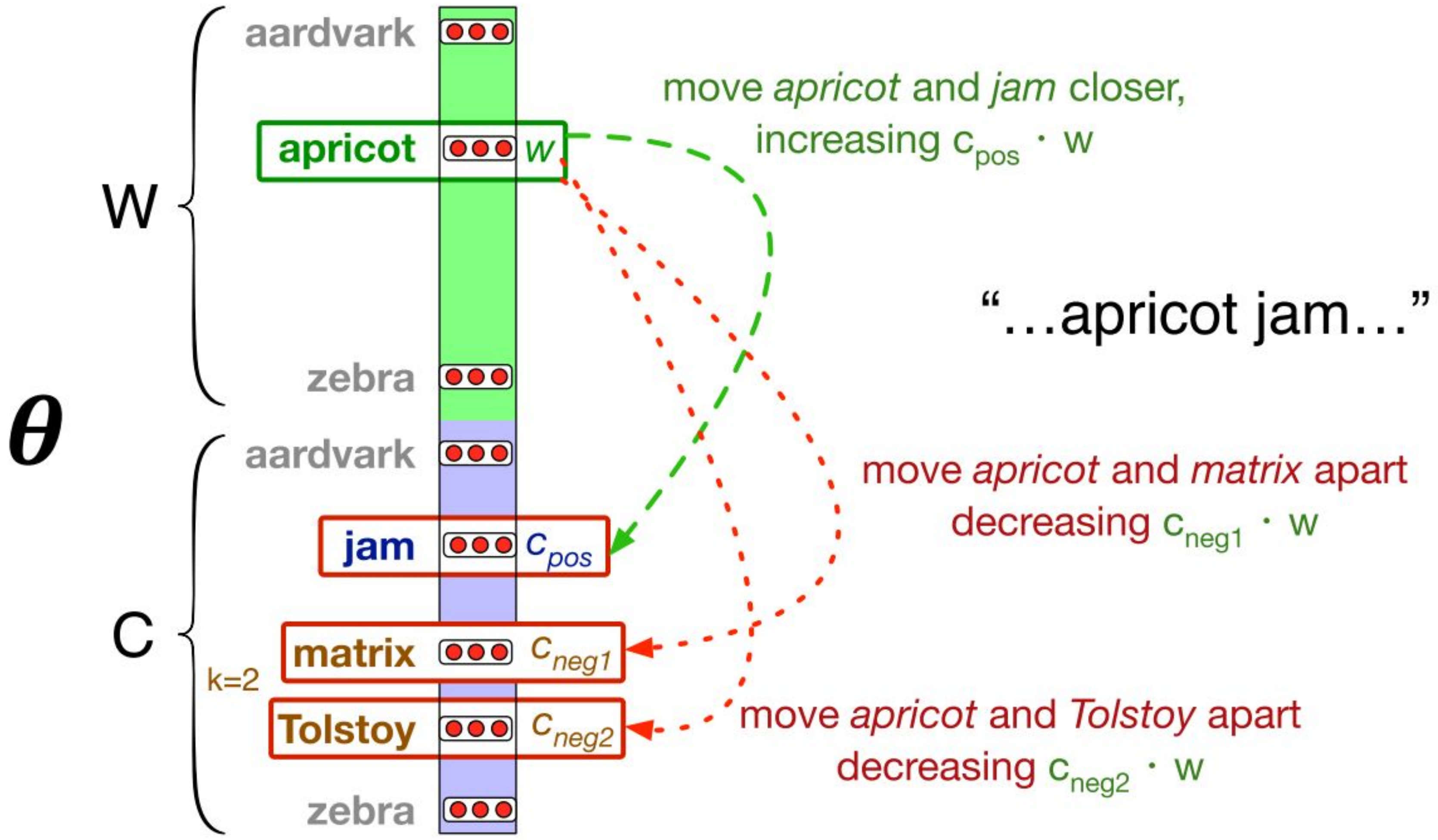
positive examples + | **negative examples -** | k=2

t	c	t	c	t	c
apricot	tablespoon	apricot	aardvark	apricot	twelve
apricot	of	apricot	puddle	apricot	hello
apricot	preserves	apricot	where	apricot	dear
apricot	or	apricot	coaxial	apricot	forever

Skip-gram: training objective

- Given target w, positive context c, (negative context c neg) x k
  - **Maximize** the similarity (score) for positive word-context pairs (label = +1).
  - **Minimize** the similarity (score) for negative word-context pairs (label = –1).

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



θ
W
aardvark
apricot
w
zebra
move apricot and jam closer,
increasing c_pos · w
"…apricot jam…"
C
aardvark
jam
c_pos
move apricot and matrix apart
decreasing c_neg1 · w
matrix
c_neg1
k=2
Tolstoy
c_neg2
move apricot and Tolstoy apart
decreasing c_neg2 · w
zebra

How to learn word2vec (skip-gram) embedding (Summary)

- Choose the dimensionality for each word vector, e.g., 300 dimensions (d=300).
- Randomly assign a 300-dimensional vector to each word in the vocabulary (V words total).
- Take a corpus and extract pairs of words that co-occur as positive examples
- Construct negative example pairs of words that do **not** typically co-occur by sampling random words ("noise words").
- Train a logistic regression classifier to distinguish positive from negative examples.
- After training, discard the classifier itself and retain the optimized word vectors—**the learned word embeddings**.

Evaluating embeddings

- Common comparison measures:
  - **Cosine similarity** → Measures angle between vectors (–1 to +1)
    - Ignores vector length, focuses on direction → ideal for high-dimensional text data.
  - **Euclidean distance** → Measures straight-line distance between points.
    - Sensitive to vector magnitude, works best when data is normalized.
- Compare to human scores on word similarity-type tasks:
  - TOEFL dataset
  - WordSim-353 (Finkelstein et al., 2002)
  - Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012)

Semantic Properties of Embeddings

- Context window size L is based on desired goals
- Similarity
  - Small L (2-4) captures similar words (e.g. list of similar things)
- Association
  - Larger L (5+) captures longer distance topical relationships

Word2Vec trained on Wikipedia

Target Word	BoW5	BoW2
batman	nightwing aquaman catwoman superman manhunter	superman superboy aquaman catwoman batgirl
hogwarts	dumbledore hallows half-blood malfoy snape	evernight sunnydale garderobe blandings collinwood
turing	nondeterministic non-deterministic computability deterministic finite-state	non-deterministic finite-state nondeterministic buchi primality
florida	gainesville fla jacksonville tampa lauderdale	fla alabama gainesville tallahassee texas
object-oriented	aspect-oriented smalltalk event-driven prolog domain-specific	aspect-oriented event-driven objective-c dataflow 4gl
dancing	singing dance dances dancers tap-dancing	singing dance dances breakdancing clowning
	L = 5	L = 2

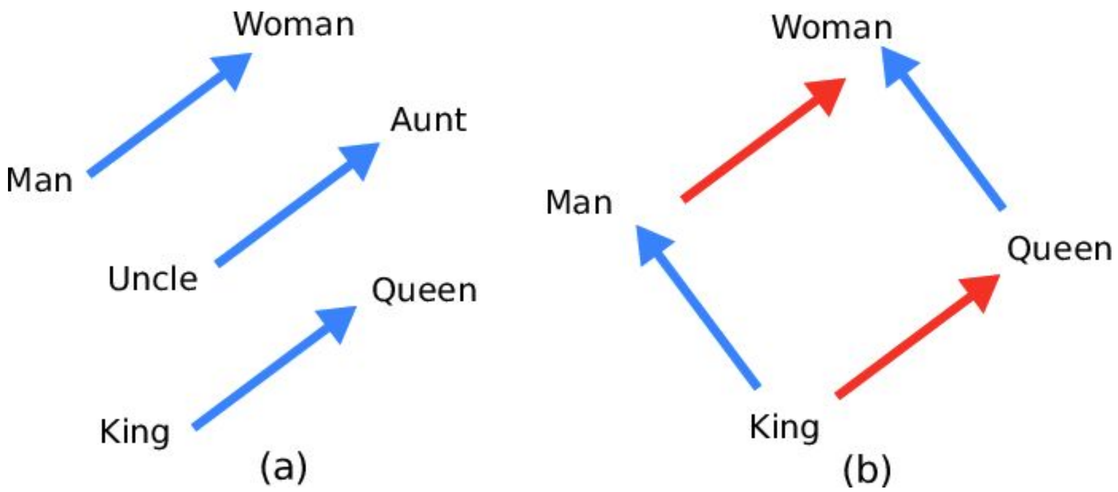
Cite: Levy, O. Goldberg, Y. Dependency-Based Word Embeddings, ACL 2014

Semantic Properties of Embeddings (Analogy):

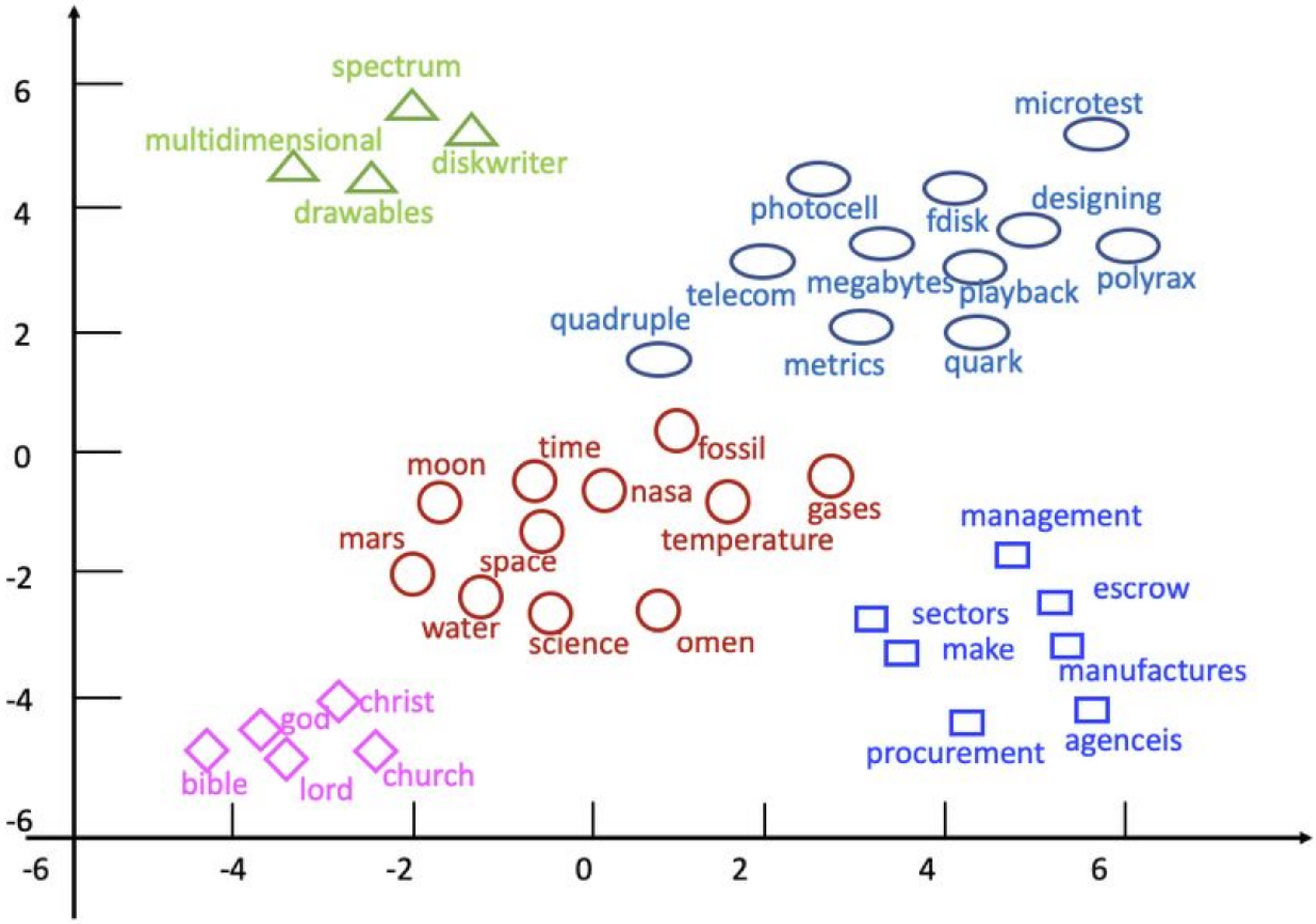
Embeddings capture relational meaning!

vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

vector(*'Paris'*) - vector(*'France'*) + vector(*'Italy'*) ≈ vector('Rome')



Cite: Mikolov, T. Yih, W. Zweig, G. Linguistic Regularities in Continuous Space Word Representations, NAACL 2013


spectrum
multidimensional
diskwriter
drawables
microtest
photocell
fdisk
designing
telecom
megabytes
playback
polyrax
quadruple
metrics
quark
time
fossil
moon
nasa
gases
management
mars
temperature
space
escrow
sectors
make
water
science
omen
manufactures
christ
god
church
bible
lord
procurement
agenceis
6
4
2
0
-2
-4
-6
-6
-4
-2
0
2
4
6

Word Embeddings: Usefulness & Limitations

Usefulness:

- Capture patterns and generalizations across words
- Powerful features for classifiers
- Enable analysis of language usage and meaning shifts in corpora

Limitations:

- Only one vector per word (fails for words with multiple senses)
- Cosine similarity can’t always separate synonyms from antonyms
- Embeddings inherit cultural biases from training data

Bias and Embeddings

- Embeddings reflect biases present in their training data.
- Historical corpora encode period-specific stereotypes (e.g., "Man → Programmer", "Woman → Homemaker").
- This can cause **allocation harm**: algorithms unfairly impact real-world decisions (e.g., filtering loan applicants).
- **Bias amplification**: embeddings often exaggerate patterns, making encoded biases more extreme.
- Implicit biases (race, age, etc.) can be captured and intensified in embeddings.

Cite: Bolukbasi, T. Chang, K. Zou, J. Saligrama, V. Kalai, A.T.
“Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings”, NIPS 2016

Bias and Embeddings

- **Representational harm:** Systems may ignore or demean certain social groups.
- **Debiasing:** Adjust embeddings to reduce stereotypes; helps but cannot eliminate bias entirely.
- Be mindful of biases in training data and algorithms.
  a. Training data questions (Does it underrepresented certain demographics or communities?)
  b. Algorithms questions (Are the results explainable and understandable to stakeholders?)
  c. Regularly check for and attempt to mitigate bias.
  d. If bias remains, declare it transparently.
  e. Decision makers should recognize and account for bias in algorithmic outputs.

Cite: Bolukbasi, T. Chang, K. Zou, J. Saligrama, V. Kalai, A.T.
“Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings”, NIPS 2016

References

- **Book Chapter 6: Vector Semantics and Embeddings** (Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models)
- Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems* 26 (2013).
- Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* (2013).

Wider reading

- Fasttext http://www.fasttext.cc/
    - https://github.com/facebookresearch/fastText
    - https://arxiv.org/pdf/1607.04606 [Paper]
- Glove http://nlp.stanford.edu/projects/glove/
    - https://nlp.stanford.edu/pubs/glove.pdf [Paper]

Thank you for your attention!