Natural Language Processing (CSE 447/547M): Word Representations

Noah Smith

University of Washington
nasmith@cs.washington.edu

January 30, 2019

Preliminaries

Token: an instance of a word observed in text

Type: the word in the abstract

How to Represent an English Word in Code

How to Represent an English Word in Code

1. Strings: apple $\rightarrow$ `"apple"` ; banana $\rightarrow$ `"banana"`

How to Represent an English Word in Code

1. Strings: apple $\rightarrow$ `"apple"` ; banana $\rightarrow$ `"banana"`
2. Integers: apple $\rightarrow 211$; banana $\rightarrow 633$

How to Represent an English Word in Code

1. Strings: apple $\rightarrow$ `"apple"` ; banana $\rightarrow$ `"banana"`
2. Integers: apple $\rightarrow 211$; banana $\rightarrow 633$
3. One-hot vector: apple $\rightarrow [\underbrace{0,\ldots,0}_{\text{210 0s}},1,0,\ldots]$; banana $\rightarrow [\underbrace{0,\ldots,0}_{\text{632 0s}},1,0,\ldots]$

How to Represent an English Word in Code

1. Strings: apple $\rightarrow$ `"apple"` ; banana $\rightarrow$ `"banana"`
2. Integers: apple $\rightarrow 211$; banana $\rightarrow 633$
3. One-hot vector: apple $\rightarrow [\underbrace{0,\ldots,0}_{\text{210 0s}},1,0,\ldots]$; banana $\rightarrow [\underbrace{0,\ldots,0}_{\text{632 0s}},1,0,\ldots]$
4. Continuous vector

How to Represent an English Word in Code

1. Strings: apple $\rightarrow$ `"apple"` ; banana $\rightarrow$ `"banana"`
2. Integers: apple $\rightarrow 211$; banana $\rightarrow 633$
3. One-hot vector: apple $\rightarrow [\underbrace{0, \ldots, 0}_{\text{210 0s}}, 1, 0, \ldots]$; banana $\rightarrow [\underbrace{0, \ldots, 0}_{\text{632 0s}}, 1, 0, \ldots]$
4. Continuous vector
   - Learned as parameters of a neural language model (January 16 lecture)

How to Represent an English Word in Code

1. Strings: apple $\rightarrow$ `"apple"` ; banana $\rightarrow$ `"banana"`
2. Integers: apple $\rightarrow 211$; banana $\rightarrow 633$
3. One-hot vector: apple $\rightarrow [\underbrace{0, \ldots, 0}_{\text{210 0s}}, 1, 0, \ldots]$; banana $\rightarrow [\underbrace{0, \ldots, 0}_{\text{632 0s}}, 1, 0, \ldots]$
4. Continuous vector
   - Learned as parameters of a neural language model (January 16 lecture)
   - Learned as parameters of a neural text classifier (January 28 lecture)

How to Represent an English Word in Code

1. Strings: apple $\to$ `"apple"` ; banana $\to$ `"banana"`
2. Integers: apple $\to 211$; banana $\to 633$
3. One-hot vector: apple $\to [\underbrace{0, \ldots, 0}_{\text{210 0s}}, 1, 0, \ldots]$; banana $\to [\underbrace{0, \ldots, 0}_{\text{632 0s}}, 1, 0, \ldots]$
4. Continuous vector
   - Learned as parameters of a neural language model (January 16 lecture)
   - Learned as parameters of a neural text classifier (January 28 lecture)
   - More methods today

The Word-Document Matrix

Let $\mathbf{A} \in \mathbb{R}^{V \times C}$ contain statistics of association between words in $\mathcal{V}$ and C documents.

N is the total number of word tokens.

Tiny example, three documents:

- `yes , we have no bananas`
- `say yes for bananas`
- `no bananas , we say`

	1	2	3
`,`	1	0	1
`bananas`	1	1	1
`for`	0	1	0
`have`	1	0	0
`no`	1	0	1
`say`	0	1	1
`we`	1	0	1
`yes`	1	1	0

Count matrix: $[\mathbf{A}]_{v,c} = c_{\boldsymbol{x}_c}(v)$

A Word-Context Matrix

Let $\mathbf{A} \in \mathbb{R}^{V \times C}$ contain statistics of association between words in $\mathcal{V}$ and symbols that occur just before them

Tiny example, three sentences:

- `yes , we have no bananas`
- `say yes for bananas`
- `no bananas , we say`

	○ _	, _	bananas _	for _	have _	no _	say _	we _	yes _
,	0	0	1	0	0	0	0	0	1
bananas	0	0	0	1	0	2	0	0	0
for	0	0	0	0	0	0	0	0	1
have	0	0	0	0	0	0	0	1	0
no	1	0	0	0	1	0	0	0	0
say	1	0	0	0	0	0	0	1	0
we	0	2	0	0	0	0	0	0	0
yes	1	0	0	0	0	0	1	0	0

Count matrix: $[\mathbf{A}]_{v,v'} = c_{\boldsymbol{x}}(v'v)$

Association Score

What we really want here is some way to get at “surprise.”

Association Score

What we really want here is some way to get at "surprise."

One way to think about this is, is the occurrence of word v in document c surprisingly high (or low), given what we'd expect due to chance?

Association Score

What we really want here is some way to get at "surprise."

One way to think about this is, is the occurrence of word v in document c surprisingly high (or low), given what we'd expect due to chance?

Chance would be $\frac{c_{\boldsymbol{x}_{1:C}}(v)}{N}$ words out of the ℓ_c words in document c.

Association Score

What we really want here is some way to get at "surprise."

One way to think about this is, is the occurrence of word v in document c surprisingly high (or low), given what we'd expect due to chance?

Chance would be $\frac{c_{\boldsymbol{x}_{1:C}}(v)}{N}$ words out of the ℓ_c words in document c.

Intuition: consider the ratio of *observed* frequency ($c_{\boldsymbol{x}_c}(v)$) to "chance" under independence ($\frac{c_{\boldsymbol{x}_{1:C}}(v)}{N} \cdot \ell_c$).

Pointwise Mutual Information

A common starting point is positive **pointwise mutual information**:

$$[\mathbf{A}]_{v,c} = \left[\log \frac{c_{\boldsymbol{x}_c}(v)}{\frac{c_{\boldsymbol{x}_{1:C}}(v)}{N} \cdot \ell_c}\right]_+ = \left[\log \frac{N \cdot c_{\boldsymbol{x}_c}(v)}{c_{\boldsymbol{x}_{1:C}}(v) \cdot \ell_c}\right]_+$$

From our example:

$$[\mathbf{A}]_{\text{bananas},1} = \log \frac{15 \cdot 1}{3 \cdot 6} \approx -0.18 \to 0$$

$$[\mathbf{A}]_{\text{for},2} = \log \frac{15 \cdot 1}{1 \cdot 4} \approx 1.32$$

	1	2	3
,	1	0	1
bananas	1	1	1
for	0	1	0
have	1	0	0
no	1	0	1
say	0	1	1
we	1	0	1
yes	1	1	0

Pointwise Mutual Information

A common starting point is positive **pointwise mutual information**:

$$[\mathbf{A}]_{v,c} = \left[\log \frac{c_{\boldsymbol{x}_c}(v)}{\frac{c_{\boldsymbol{x}_{1:C}}(v)}{N} \cdot \ell_c}\right]_+ = \left[\log \frac{N \cdot c_{\boldsymbol{x}_c}(v)}{c_{\boldsymbol{x}_{1:C}}(v) \cdot \ell_c}\right]_+$$

Notes:

- If a word v appears with nearly the same frequency in every document, its row $[\mathbf{A}]_{v,*}$ will be all nearly zero.

Pointwise Mutual Information

A common starting point is positive **pointwise mutual information**:

$$[\mathbf{A}]_{v,c} = \left[\log \frac{c_{\boldsymbol{x}_c}(v)}{\frac{c_{\boldsymbol{x}_{1:C}}(v)}{N} \cdot \ell_c}\right]_+ = \left[\log \frac{N \cdot c_{\boldsymbol{x}_c}(v)}{c_{\boldsymbol{x}_{1:C}}(v) \cdot \ell_c}\right]_+$$

Notes:

- If a word v appears with nearly the same frequency in every document, its row $[\mathbf{A}]_{v,*}$ will be all nearly zero.
- If a word v occurs *only* in document c, PMI will be large and positive.

Pointwise Mutual Information

A common starting point is positive **pointwise mutual information**:

$$[\mathbf{A}]_{v,c} = \left[\log \frac{c_{\boldsymbol{x}_c}(v)}{\frac{c_{\boldsymbol{x}_{1:C}}(v)}{N} \cdot \ell_c}\right]_+ = \left[\log \frac{N \cdot c_{\boldsymbol{x}_c}(v)}{c_{\boldsymbol{x}_{1:C}}(v) \cdot \ell_c}\right]_+$$

Notes:

- If a word v appears with nearly the same frequency in every document, its row $[\mathbf{A}]_{v,*}$ will be all nearly zero.
- If a word v occurs *only* in document c, PMI will be large and positive.
- PMI is very sensitive to rare occurrences; usually we smooth the frequencies and filter rare words.

Pointwise Mutual Information

A common starting point is positive **pointwise mutual information**:

$$[\mathbf{A}]_{v,c} = \left[\log \frac{c_{\boldsymbol{x}_c}(v)}{\frac{c_{\boldsymbol{x}_{1:C}}(v)}{N} \cdot \ell_c}\right]_+ = \left[\log \frac{N \cdot c_{\boldsymbol{x}_c}(v)}{c_{\boldsymbol{x}_{1:C}}(v) \cdot \ell_c}\right]_+$$

Notes:

- If a word v appears with nearly the same frequency in every document, its row $[\mathbf{A}]_{v,*}$ will be all nearly zero.
- If a word v occurs *only* in document c, PMI will be large and positive.
- PMI is very sensitive to rare occurrences; usually we smooth the frequencies and filter rare words.
- One way to think about PMI: it's telling us where a unigram model is most wrong.

Pointwise Mutual Information

A common starting point is positive **pointwise mutual information**:

$$[\mathbf{A}]_{v,c} = \left[\log \frac{c_{\boldsymbol{x}_c}(v)}{\frac{c_{\boldsymbol{x}_{1:C}}(v)}{N} \cdot \ell_c}\right]_+ = \left[\log \frac{N \cdot c_{\boldsymbol{x}_c}(v)}{c_{\boldsymbol{x}_{1:C}}(v) \cdot \ell_c}\right]_+$$

Notes:

- If a word v appears with nearly the same frequency in every document, its row $[\mathbf{A}]_{v,*}$ will be all nearly zero.
- If a word v occurs *only* in document c, PMI will be large and positive.
- PMI is very sensitive to rare occurrences; usually we smooth the frequencies and filter rare words.
- One way to think about PMI: it's telling us where a unigram model is most wrong.
- We could use $\mathbf{A}$ as $\mathbf{M}$ (though d is usually much smaller than C) ...

Topic Models: Latent Semantic Indexing/Analysis

(Deerwester et al., 1990)

LSI/A seeks to solve:

$$\underset{V \times C}{\mathbf{A}} \approx \hat{\mathbf{A}} = \underset{V \times d}{\mathbf{M}} \times \underset{d \times d}{\operatorname{diag}(\mathbf{s})} \times \underset{d \times C}{\mathbf{C}}^{\top}$$

where $\mathbf{M}$ contains embeddings of words, $\mathbf{C}$ contains embeddings of documents.

$$[\mathbf{A}]_{v,c} \approx \sum_{i=1}^{d} [\mathbf{m}_v]_i \cdot [\mathbf{s}]_i \cdot [\mathbf{c}_c]_i$$

Topic Models: Latent Semantic Indexing/Analysis

(Deerwester et al., 1990)

LSI/A seeks to solve:

$$\underset{V \times C}{\mathbf{A}} \approx \hat{\mathbf{A}} = \underset{V \times d}{\mathbf{M}} \times \underset{d \times d}{\operatorname{diag}(\mathbf{s})} \times \underset{d \times C}{\mathbf{C}}^\top$$

where $\mathbf{M}$ contains embeddings of words, $\mathbf{C}$ contains embeddings of documents.

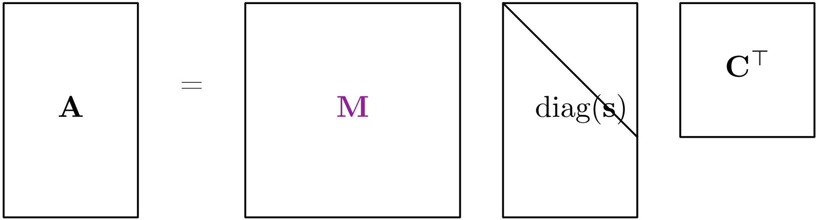
$$[\mathbf{A}]_{v,c} \approx \sum_{i=1}^{d} [\mathbf{m}_v]_i \cdot [\mathbf{s}]_i \cdot [\mathbf{c}_c]_i$$

This can be solved by applying singular value decomposition to $\mathbf{A}$, then truncating to d dimensions.

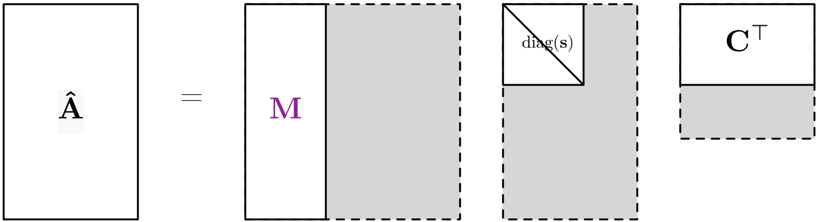
- $\mathbf{M}$ contains left singular vectors of $\mathbf{A}$
- $\mathbf{C}$ contains right singular vectors of $\mathbf{A}$
- $\mathbf{s}$ are singular values of $\mathbf{A}$; they are nonnegative and conventionally organized in decreasing order.

Truncated Singular Value Decomposition

SVD:



truncated at d:



A Nod to Linear Algebra

For (not truncated) singular value decomposition $\mathbf{A} = \mathbf{M} \times \operatorname{diag}(\mathbf{s}) \times \mathbf{C}^\top$:

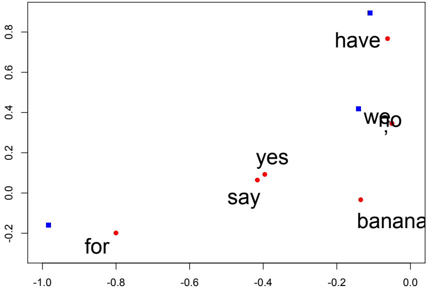
- The columns of $\mathbf{M}$ form an orthonormal basis, $\mathbf{M}$ are eigenvectors of $\mathbf{A}\mathbf{A}^\top$, with eigenvalues $\mathbf{s}^2$.
- The columns of $\mathbf{C}$ form an orthonormal basis, $\mathbf{C}$ are eigenvectors of $\mathbf{A}^\top\mathbf{A}$, with eigenvalues $\mathbf{s}^2$.

If some elements of $\mathbf{s}$ are zero, then $\mathbf{A}$ is "low rank."

Approximating $\mathbf{A}$ by truncating $\mathbf{s}$ equates to a "low rank approximation."

LSI/A Example

$k = 2$

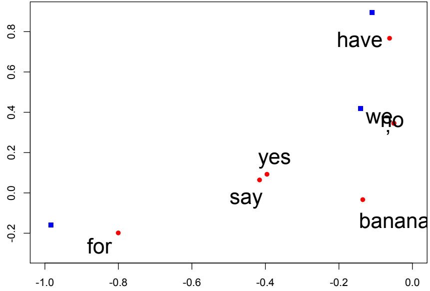


	1	2	3
,	1	0	1
bananas	1	1	1
for	0	1	0
have	1	0	0
no	1	0	1
say	0	1	1
we	1	0	1
yes	1	1	0

Words and documents in two dimensions.

LSI/A Example

$k = 2$



	1	2	3
,	1	0	1
bananas	1	1	1
for	0	1	0
have	1	0	0
no	1	0	1
say	0	1	1
we	1	0	1
yes	1	1	0

Words and documents in two dimensions.

Note how `no`, `we`, and `,` are all in the exact same spot. Why?

Understanding LSI/A

- Mapping words and documents into the same k-dimensional space.
- Bag of words assumption (Salton et al., 1975): a document is nothing more than the distribution of words it contains.
- Distributional hypothesis (Harris, 1954; Firth, 1957): words are nothing more than the distribution of contexts (here, documents) they occur in. Words that occur in similar contexts have similar meanings.
- $\mathbf{A}$ is sparse and noisy; LSI/A "fills in" the zeroes and tries to eliminate the noise.
  - It finds the best rank-k approximation to $\mathbf{A}$.

Local Contexts: Distributional Semantics

Within NLP, emphasis has shifted from word/document associations to the relationship between $v \in \mathcal{V}$ and more local contexts.

For example: LSI/A, but replace documents with "nearby words." This is a way to recover word vectors that capture distributional similarity.

These models are designed to "guess" a word at position i given a word at a position in $\{i - w, \ldots, i - 1\} \cup \{i + 1, \ldots, i + w\}$. (My example on slide 12 used only position $i - 1$.)

Sometimes such methods are used to "pre-train" word vectors used in other, richer models (like neural language models).

word2vec

(Mikolov et al., 2013a,b)

Two models for word vectors designed to be computationally efficient.

- Continuous bag of words (CBOW): $p(v \mid c)$
  - Similar in spirit to the feedforward neural language model we saw before (Bengio et al., 2003)
- Skip-gram: $p(c \mid v)$

It turns out these are closely related to matrix factorization as in LSI/A (Levy and Goldberg, 2014).

Skip-Gram Model

$$p(C = c \mid X = v) = \frac{1}{Z_v} \exp \mathbf{c}_c^\top \mathbf{v}_v$$

- Two different vectors for each element of $\mathcal{V}$: one when it is "v" $(\mathbf{v})$ and one when it is "c" $(\mathbf{c})$.
- Normalization term Z_v is expensive, so approximations are required for efficiency.
- Can vary the context window, or expand this to be over the whole sentence or document, or otherwise choose which words "count" as contexts.

Word Vector Evaluations

Several popular methods for *intrinsic* evaluations:

Word Vector Evaluations

Several popular methods for *intrinsic* evaluations:

- How well do (cosine) similarities of pairs of words' vectors correlate with judgments of word similarity by humans?

Word Vector Evaluations

Several popular methods for *intrinsic* evaluations:

- How well do (cosine) similarities of pairs of words' vectors correlate with judgments of word similarity by humans?
- TOEFL-like synonym tests, e.g., *rug* $\stackrel{?}{\rightarrow}$ {*sofa*, *ottoman*, *carpet*, *hallway*}

Word Vector Evaluations

Several popular methods for *intrinsic* evaluations:

- How well do (cosine) similarities of pairs of words' vectors correlate with judgments of word similarity by humans?
- TOEFL-like synonym tests, e.g., *rug* $\xrightarrow{?}$ $\{$*sofa*, *ottoman*, *carpet*, *hallway*$\}$
- Syntactic analogies, e.g., "*walking* is to *walked* as *eating* is to what?" Solved via:

$$\min_{v \in \mathcal{V}} \cos\left(\mathbf{v}_v, \mathbf{v}_{\mathit{walking}} - \mathbf{v}_{\mathit{walked}} + \mathbf{v}_{\mathit{eating}}\right)$$

Extrinsic Evaluations

1. Use large unannotated corpus to get your word vectors (called **pretraining**).
2. Use them in a text classifier (or some other NLP system, more examples to be introduced in future lectures). Two options:
   - Plug in word vectors as "frozen" features.
   - Treat them as parameters of the text classifier; pretraining gives initial values, but they get updated, or "finetuned" during supervised learning.
3. Does that system's performance improve?

Stepping Back

Big ideas:

- This is *unsupervised* learning: all you need is lots of raw text (no labels!)
- Large corpora $\rightarrow$ powerful word representations (the distributional hypothesis from linguistics, brought to life through engineering)
- It's all about the relationship between *words* and their *contexts*

Embeddings from Language Models (ELMo)

(Peters et al., 2018)

Why not give every word *token* (in context) its own vector?

- To do that, we need a function that maps contexts to vectors.
- An RNN language model can do that, for the entire left context.
  - $\mathbf{s}_i$ is x_i's history. $\boldsymbol{s}_{i+1} = f_{\text{recurrent}}(\mathbf{e}_{x_i}, \mathbf{s}_i)$ is a (left-)*contextualized* embedding of the token x_i.
  - To get the right-context, run an RNN language model from right to left, and extract the analogous state vector just after reading x_i. Concatenate the two.
- On extrinsic evaluations, this method gave big improvements to state of the art systems.

References I

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3(Feb):1137–1155, 2003. URL `http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf`.

Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6): 391–407, 1990.

J. R. Firth. A synopsis of linguistic theory 1930–1955. In *Studies in Linguistic Analysis*, pages 1–32. Blackwell, 1957.

Zellig Harris. Distributional structure. *Word*, 10(23):146–162, 1954.

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *NIPS*, 2014.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of ICLR*, 2013a. URL `http://arxiv.org/pdf/1301.3781.pdf`.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013b. URL `http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf`.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of NAACL*, 2018.

Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.