Natural Language Processing (CSE 517): Vector Embeddings for Documents and Words

Noah Smith

University of Washington
nasmith@cs.washington.edu

Winter 2023

Readings: Eisenstein (2019) 14 and Smith (2020)

A Mismatch

Theories of language tend to view the data (words, sentences, documents) and abstractions over it as *symbolic* or categorical.

Machine learning algorithms built on optimization thrive on a diet of *continuous* data.

This Lecture: Documents and Words as Vectors

A common thread: we derive the vectors from a corpus (collection of documents), with *no annotation*.

- Variously called "unsupervised" and "self-supervised" learning.

The situation is similar to language modeling.

Topic Models

- Words are not IID!
  - Predictable given history: n-gram/Markov models
  - Predictable given *other words* in the document: topic models

Topic Models

- Words are not IID!
  - Predictable given history: n-gram/Markov models
  - Predictable given *other words* in the document: topic models
- Let $\mathcal{Z} = \{1, \ldots, K\}$ be a set of "topics" or "themes" that will help us capture the interdependence of words in a document.
  - Usually these are not named or characterized in advance; they are just K different values with no *a priori* meaning.

Topic Models

- Words are not IID!
  - Predictable given history: n-gram/Markov models
  - Predictable given *other words* in the document: topic models
- Let $\mathcal{Z} = \{1, \ldots, K\}$ be a set of "topics" or "themes" that will help us capture the interdependence of words in a document.
  - Usually these are not named or characterized in advance; they are just K different values with no *a priori* meaning.
- We'll start with a classical topic model, then turn to probabilistic ones.

Notation

- $\boldsymbol{x}$ is the corpus; it contains C documents
- $\boldsymbol{x}_c$ is the cth document in the corpus
- ℓ_c is the length of $\boldsymbol{x}_c$ (in tokens)
- N is the total count of tokens in the corpus, $N = \sum_{c=1}^{C} \ell_c$

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. Tiny example, three documents:

$\boldsymbol{x}_1$: `yes , we have no bananas`

$\boldsymbol{x}_2$: `say yes for bananas`

$\boldsymbol{x}_3$: `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: $[\mathbf{A}]_{v,c} = \text{count}_{\boldsymbol{x}_c}(v)$ (count of word v in the cth document)

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, if all documents were essentially similar?

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, if all documents were essentially similar?

Chance (under a unigram model) would be $\frac{\text{count}_{\boldsymbol{x}}(v)}{N}$ 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, if all documents were essentially similar?

Chance (under a unigram model) would be $\frac{\text{count}_{\boldsymbol{x}}(v)}{N}$ words out of the ℓ_c words in document c.

Intuition: consider the ratio of *observed* frequency of word v in document c ($\text{count}_{\boldsymbol{x}_c}(v)$) to "chance" under independence ($\frac{\text{count}_{\boldsymbol{x}}(v)}{N} \cdot \ell_c$).

Pointwise Mutual Information

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

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

where $[x]_+ = \max(0, x)$. From our example:

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

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

	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

A Nod to Information Theory

Pointwise mutual information for two random variables A and B:

$$\begin{aligned}\mathsf{PMI}(a, b) &= \log \frac{p(A = a, B = b)}{p(A = a) \cdot p(B = b)} \\ &= \log \frac{p(A = a \mid B = b)}{p(A = a)} \\ &= \log \frac{p(B = b \mid A = a)}{p(B = b)}\end{aligned}$$

A Nod to Information Theory

Pointwise mutual information for two random variables A and B:

$$\text{PMI}(a, b) = \log \frac{p(A = a, B = b)}{p(A = a) \cdot p(B = b)}$$

The **average mutual information** is given by:

$$\text{MI}(A, B) = \sum_{a,b} p(A = a, B = b) \cdot \text{PMI}(a, b)$$

This comes from information theory; it is the amount of information each r.v. offers about the other.

(Recall entropy; the amount of information or uncertainty in a single random variable.)

Pointwise Mutual Information

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

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

where $[x]_+ = \max(0, x)$.
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**:

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

where $[x]_+ = \max(0, x)$.
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, their PMI ($[\mathbf{A}]_{v,c}$) will be large and positive.

Pointwise Mutual Information

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

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

where $[x]_+ = \max(0, x)$.
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, their PMI ($[\mathbf{A}]_{v,c}$) 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**:

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

where $[x]_+ = \max(0, x)$.
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, their PMI ($[\mathbf{A}]_{v,c}$) 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.

Reflection

We could use this association matrix $\mathbf{A}$ as $\mathbf{M}$, the embeddings matrix in a neural language model. Can you think of some advantanges and disadvantages of doing so?

Topic Models: Latent Semantic Indexing/Analysis

(Deerwester et al., 1990)

LSI/A seeks to solve (for $\mathbf{M}$, $\mathbf{s}$, and $\mathbf{C}$):

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

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

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

Topic Models: Latent Semantic Indexing/Analysis

(Deerwester et al., 1990)

LSI/A seeks to solve (for $\mathbf{M}$, $\mathbf{s}$, and $\mathbf{C}$):

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

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

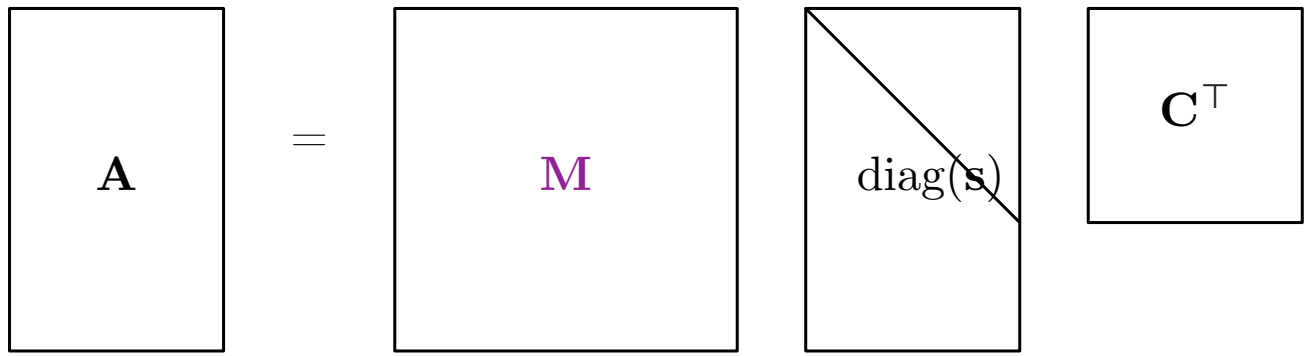
$$[\mathbf{A}]_{v,c} \approx \sum_{i=1}^{d} [\mathbf{v}_v]_i \cdot [\mathbf{s}]_i \cdot [\mathbf{c}_c]_i$$

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

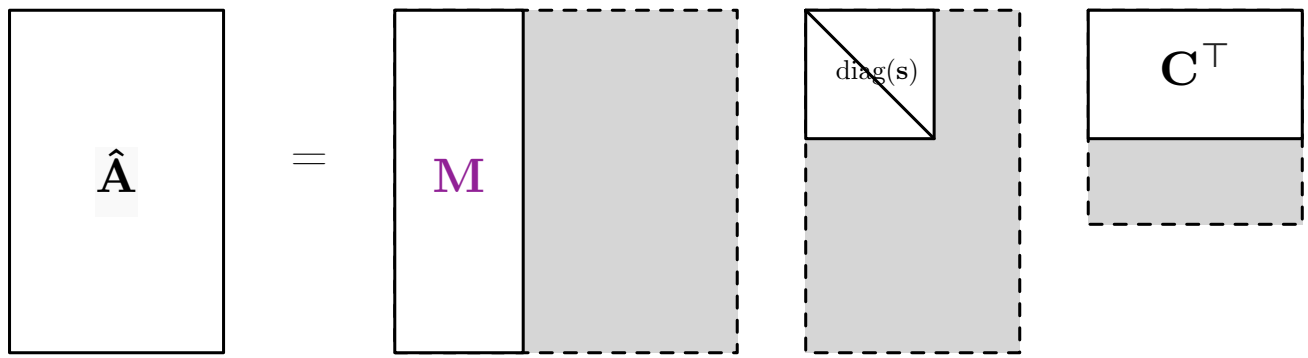
- $\mathbf{M}$ contains left singular vectors of $\mathbf{A}$
- $\mathbf{C}$ contains right singular vectors of $\mathbf{A}$
- $\mathbf{s}$ are singular values of $\mathbf{A}$; they are nonnegative and conventionally organized in decreasing order.

Truncated Singular Value Decomposition

SVD:



truncated at d:



A Nod to Linear Algebra

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

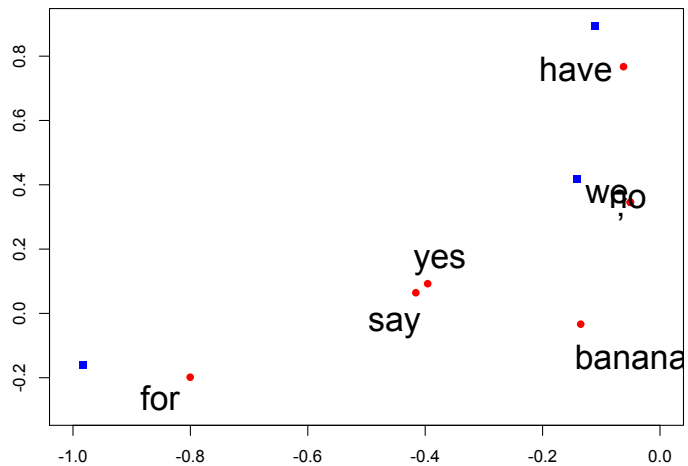
- The columns of $\mathbf{M}$ form an orthonormal basis, $\mathbf{M}$ are eigenvectors of $\mathbf{A}\mathbf{A}^\top$, with eigenvalues $\mathbf{s}^2$.
- The columns of $\mathbf{C}$ form an orthonormal basis, $\mathbf{C}$ are eigenvectors of $\mathbf{A}^\top\mathbf{A}$, with eigenvalues $\mathbf{s}^2$.

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

$d = 2$

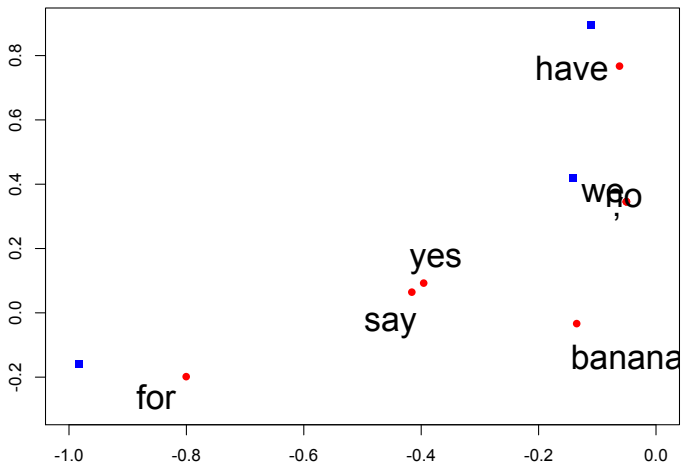


	1	2	3
,	1	0	1
bananas	1	1	1
for	0	1	0
have	1	0	0
no	1	0	1
say	0	1	1
we	1	0	1
yes	1	1	0

Words and documents in two dimensions.

LSI/A Example

$d = 2$



	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

- It creates a mapping of words and documents into the same low-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' meanings 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-d approximation to $\mathbf{A}$.

Probabilistic Topic Models

As a language model, LSI/A is kind of broken.

- It assumes the elements of $\mathbf{A}$ are the result of Gaussian noise.

Hofmann (1999) proposed instead to model the probability distribution $p(\boldsymbol{X}_c = \boldsymbol{x}_c \mid c)$, for each document c.

- This is a particular kind of *conditional* language model.

Probabilistic Latent Semantic Analysis

(Hofmann, 1999)

For every document c in the corpus:

$$p(\boldsymbol{x}_c \mid c) = \sum_{\boldsymbol{z} \in \{1,\ldots,K\}^{\ell_c}} p(\boldsymbol{x}_c, \boldsymbol{z} \mid c)$$

$$p(\boldsymbol{x}_c, \boldsymbol{z} \mid c) = \prod_{i=1}^{\ell_c} p_{topic}(z_i \mid c) \cdot p_{word}(x_{c,i} \mid z_i)$$

One way to view PLSA is that each document is generated by a separate mixture of K unigram models.

Parameters:

- p_{topic}, a distribution over K topics, for each document c
- p_{word}, a (unigram) distribution over $\mathcal{V}$, for each topic

There is no closed form for the MLE!

A Chicken/Egg Problem

If we knew which topic each word token belonged to (i.e., which unigram distribution p_{word} generated it), we could use relative frequency estimation.

If we knew the parameters p_{topic} and p_{word}, we could infer the topic of each word (i.e., which unigram distribution p_{word} generated it).

"Soft Counts"

Assume for the moment a single document c of length ℓ.

When we estimated unigram language models, everything relied on *counts* of words.

Here, if we knew the counts of every word in every topic in every document, then we'd have a closed form MLE.

$$\hat{p}_{topic}(z \mid c) = \frac{\text{count}(z, *)}{\ell}$$

$$\hat{p}_{word}(v \mid z) = \frac{\text{count}(z, v)}{\text{count}(z, *)}$$

"Soft Counts"

Assume for the moment a single document c of length ℓ.

When we estimated unigram language models, everything relied on *counts* of words.

Here, if we knew the counts of every word in every topic in every document, then we'd have a closed form MLE.

$$\hat{p}_{topic}(z \mid c) = \frac{\text{count}(z, *)}{\ell}$$

$$\hat{p}_{word}(v \mid z) = \frac{\text{count}(z, v)}{\text{count}(z, *)}$$

Instead, we will replace counts with "soft counts."

Expectation Maximization

Many ways to understand it. Today, we'll stick with a simple one.

Start with arbitrary (e.g., random) parameter values. Alternate between two steps:

- E step: calculate the posterior distribution over each word's topic assignment.
- M step: treat the posteriors as soft counts, and re-estimate the model.

Doing this is a kind of hill-climbing on the likelihood of the *observed* data.

PLSA: M Step

Assume each word x_i in document c is fractionally assigned to every topic z with (known) value $q_{c,i}(z)$.

PLSA: M Step

Assume each word x_i in document c is fractionally assigned to every topic z with (known) value $q_{c,i}(z)$.

$$\hat{p}_{topic}(z \mid c) = \frac{\sum_{i=1}^{\ell_c} q_{c,i}(z)}{\ell_c}$$

$$= \frac{\text{soft count of } c\text{'s words assigned topic } z}{\text{size of } c}$$

PLSA: M Step

Assume each word x_i in document c is fractionally assigned to every topic z with (known) value $q_{c,i}(z)$.

$$\hat{p}_{topic}(z \mid c) = \frac{\sum_{i=1}^{\ell_c} q_{c,i}(z)}{\ell_c}$$

$$= \frac{\text{soft count of } c\text{'s words assigned topic } z}{\text{size of } c}$$

$$\hat{p}_{word}(v \mid z) = \frac{\sum_c \sum_{i:x_{c,i}=v} q_{c,i}(z)}{\sum_c \sum_{i=1}^{\ell_c} q_{c,i}(z)}$$

$$= \frac{\text{soft count of } v \text{ tokens assigned topic } z}{\text{soft count of all tokens assigned topic } z}$$

PLSA: M Step

Assume each word x_i in document c is fractionally assigned to every topic z with (known) value $q_{c,i}(z)$.

$$\hat{p}_{topic}(z \mid c) = \frac{\sum_{i=1}^{\ell_c} q_{c,i}(z)}{\ell_c}$$

$$= \frac{\text{soft count of } c\text{'s words assigned topic } z}{\text{size of } c}$$

$$\hat{p}_{word}(v \mid z) = \frac{\sum_c \sum_{i:x_{c,i}=v} q_{c,i}(z)}{\sum_c \sum_{i=1}^{\ell_c} q_{c,i}(z)}$$

$$= \frac{\text{soft count of } v \text{ tokens assigned topic } z}{\text{soft count of all tokens assigned topic } z}$$

Note that the p_{word} parameters are shared across the corpus; all of the documents influence our beliefs about the others through these distributions.

PLSA: E Step

Assume we have the parameters p_{topic} and p_{word}.

Calculate, for every document c, for every word x_i in c, its "membership" to every topic:

$$q_{c,i}(z) = \frac{p_{topic}(z \mid c) \cdot p_{word}(x_{c,i} \mid z)}{\sum_{z'} p_{topic}(z' \mid c) \cdot p_{word}(x_{c,i} \mid z')}$$
$$= \frac{\text{joint probability of } x_{c,i} \text{ and } z, \text{ given } c}{\text{marginal probability of } x_{c,i} \text{ given } c}$$

Each word gets to vote on topics; it can spread its vote fractionally across them, but the votes sum to 1.

Expectation Maximization

Very general technique for learning with *incomplete data*. It's been invented over and over in different fields.

Requires that you specify a generative model with two kinds of variables: **observed** (here, documents and words in each document), and **latent** (here, topic for each word). EM (locally) maximizes the likelihood of the observed data.

Like gradient descent for neural networks, we are (usually) optimizing a non-convex function. Many tricks exist to try to cope with that.

In NLP, EM has often been associated with unsupervised learning.

Remarks

- Like LSI/A, PLSA "squeezes" the relationship between words and contexts (documents) through topics.
- A document is now characterized as a *mixture* of corpus-universal topics (each of which is a unigram model): $p_{topics}(* \mid c)$ is a vector representation of $\boldsymbol{x}_c$!
- Topic mixtures can be incorporated into language models; see Iyer and Ostendorf (1999), for example.
- Compared to LSI/A: PLSA is more interpretable (e.g., LSI/A can give negative values!).

But ...

- PLSA cannot assign probability to a text not in the training corpus; it only defines conditional distributions over words given texts in that corpus. Recall that p_{topic} conditions on a specific document!
- The next model overcomes this problem by adding another level of randomness: p_{topic} becomes a random variable, not a parameter.

Quick Probability Review

Consider a joint distribution over two random variables A and B. The **marginal** distribution over A is usually given as:

$$p(A = a) = \sum_b p(A = a, B = b)$$

But if B is *continuous*, then the sum becomes an integral:

$$p(A = a) = \int_b p(A = a, B = b) db$$

Latent Dirichlet Allocation

(Blei et al., 2003)

Widely used in text exploration (e.g., social science research).

$$p(\boldsymbol{x}_c) = \int_{\boldsymbol{p}_{topic}} \sum_{\boldsymbol{z} \in \{1,\ldots,K\}^{\ell_c}} p(\boldsymbol{x}_c, \boldsymbol{z}, \boldsymbol{p}_{topic}) \; d\boldsymbol{p}_{topic}$$

$$p(\boldsymbol{x}_c, \boldsymbol{z}, \boldsymbol{p}_{topic}) = \mathrm{Dir}_{\boldsymbol{\alpha}}(\boldsymbol{p}_{topic}) \prod_{i=1}^{\ell_c} p_{topic}(z_i) \cdot p_{word}(x_{c,i} \mid z_i)$$

Parameters:

- $\boldsymbol{\alpha} \in \mathbb{R}^K_{>0}$, a prior over topic distributions
- $p_{word}(* \mid z), \forall z \in \{1, \ldots, K\}$

There is no closed form for the MLE!

"Being Bayesian"

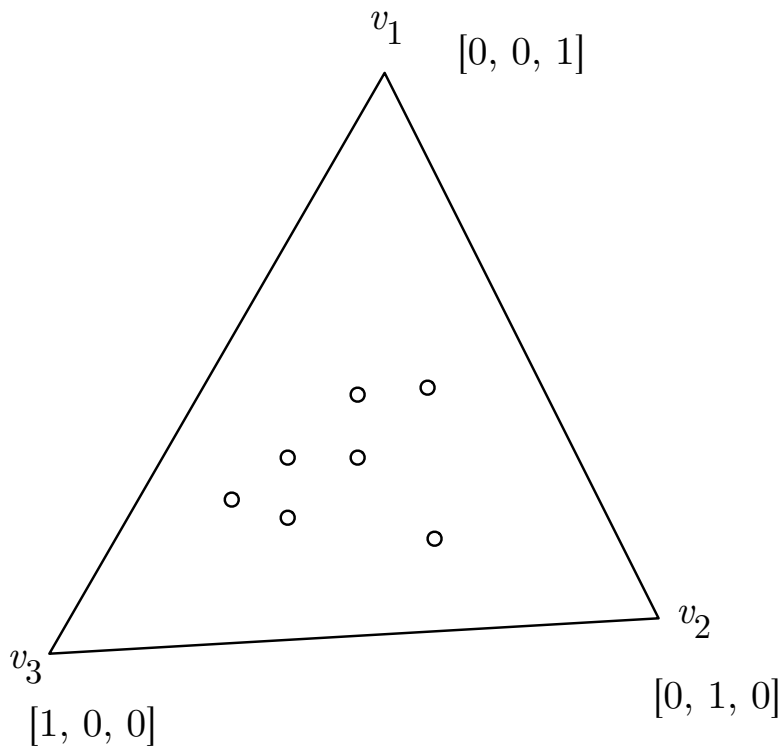
This is another topic that could warrant an entire quarter; see Cohen (2016).

A summary of the Bayesian philosophy in NLP:

- Because we have finite data, we should be uncertain about every estimated model parameter.
- Bayes' rule gives us a way to manage that uncertainty, *if* we can define a **prior** distribution over model parameters.
- Inference is a "simple matter" of estimating posterior distributions.
  - But exact inference is almost never tractable, so we need approximations.
  - There are many of these, and they tend to be expensive.
  - Some of them look like EM, some don't.
  - "Neural variational inference" methods are the latest to generate excitement (e.g., Miao et al., 2016).

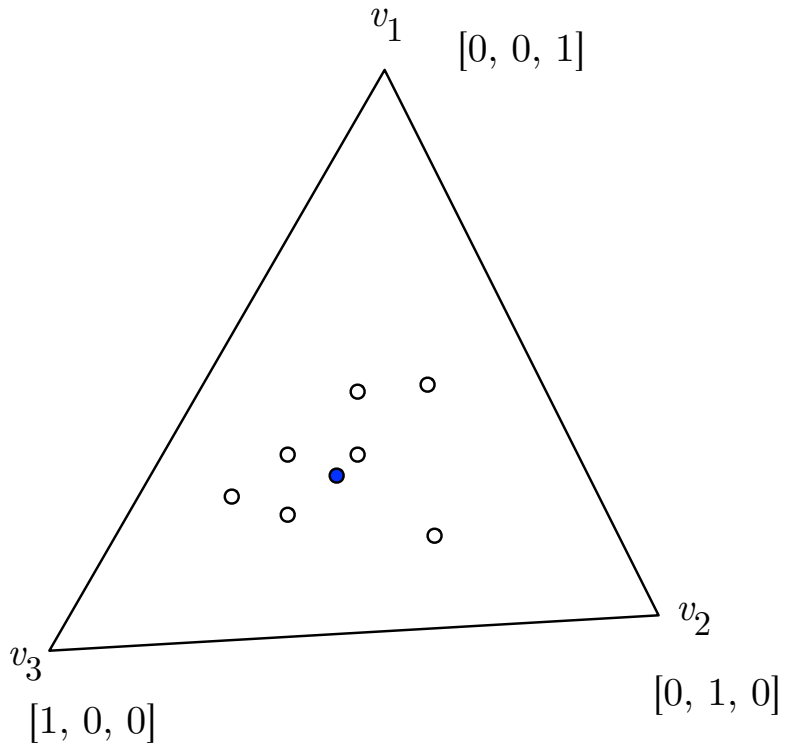
Understanding LDA and other Topic Models

Consider a simple case where $V = 3$. All unigram distributions, and hence all documents, reside in this triangle (white circles):



Understanding LDA and other Topic Models

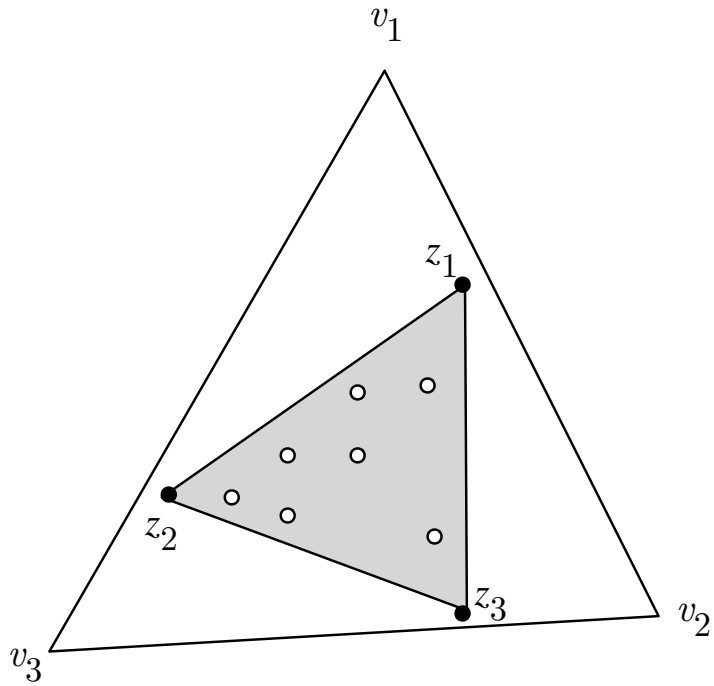
Consider a simple case where $V = 3$. All unigram distributions, and hence all documents, reside in this triangle (white circles):



Unigram model estimates one "topic" for the whole corpus.

Understanding LDA and other Topic Models

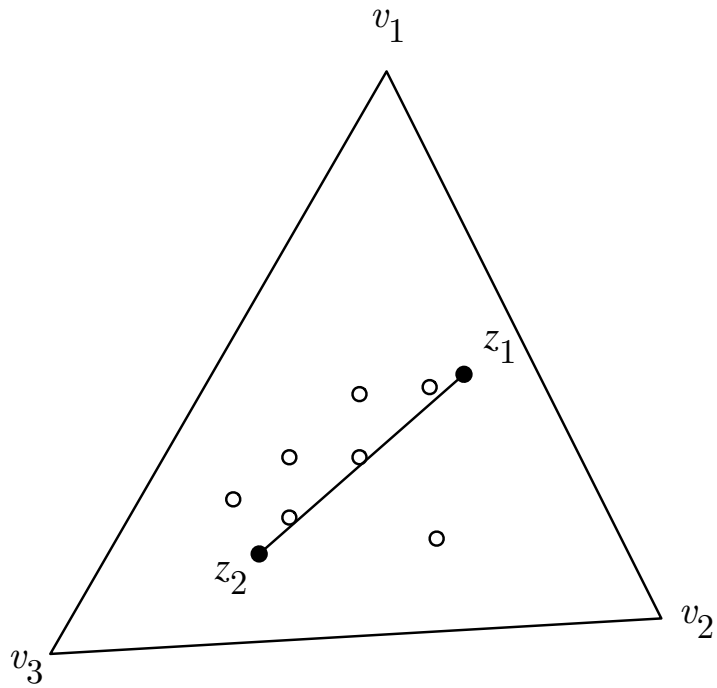
Consider a simple case where $V = 3$. All unigram distributions, and hence all documents, reside in this triangle (white circles):



PLSA chooses a topic simplex within the larger one and places each document at one point in the topic simplex.

Understanding LDA and other Topic Models

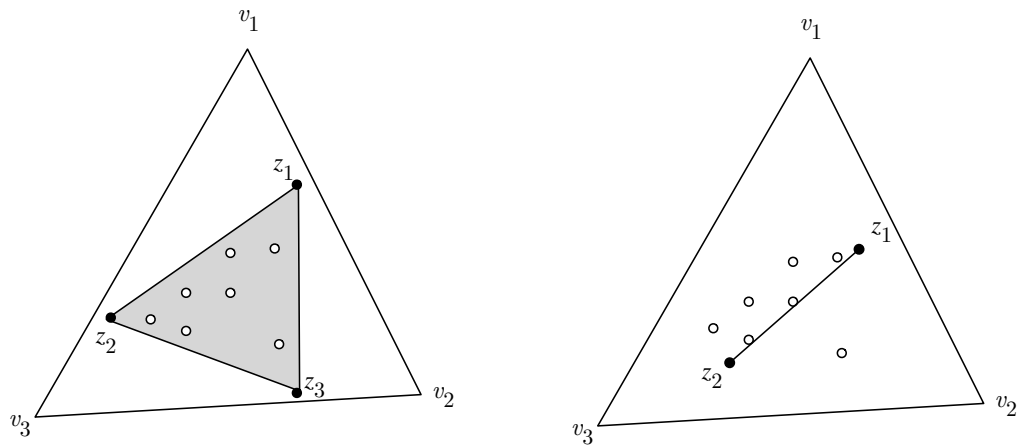
Consider a simple case where $V = 3$. All unigram distributions, and hence all documents, reside in this triangle (white circles):



PLSA chooses a topic simplex within the larger one and places each document at one point in the topic simplex.

Understanding LDA and other Topic Models

Consider a simple case where $V = 3$. All unigram distributions, and hence all documents, reside in this triangle (white circles):



LDA estimates a distribution in the "topic simplex" for each document, given its words (hence, a posterior distribution), and so can generalize to new documents.

LDA

Topics discovered by LDA-like models continue to be interesting:

- As a way of interacting with and exploring large corpora without reading them.
- A demo you can play with: `https://mimno.infosci.cornell.edu/jsLDA/jslda.html`
  - But this is hard to evaluate!
- As a "pivot" for relating to other variables like author (Rosen-Zvi et al., 2004), geography (Eisenstein et al., 2010), and many more.

LDA is also extremely useful as a pedagogical gateway to Bayesian modeling of text (and other discrete data).

- It's right on the boundary between "easy" and "hard" Bayesian models.

Local Contexts: Distributional Semantics

Within NLP, emphasis has shifted from topics 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.

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:

$\boldsymbol{x}_1$: `yes , we have no bananas`

$\boldsymbol{x}_2$: `say yes for bananas`

$\boldsymbol{x}_3$: `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: $[\mathbf{A}]_{v,v'} = \text{count}_{\boldsymbol{x}}(v'v)$

Word Vector Models

These models are designed to "guess" a word at position i given a word at a position in $\{i-w,\ldots,i-1\}\cup\{i+1,\ldots,i+w\}$.

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 in the language model lecture (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" ($\mathbf{c}$).
- This should remind you of a neural network; SGD on the likelihood function is the conventional approach to estimating the vectors.
- Normalization term Z_v is expensive, so approximations are required for efficiency.
- Can expand this to be over the whole sentence or document, or otherwise choose which words "count" as context.

Word Vector Evaluations

Several popular methods for *intrinsic* evaluations:

Word Vector Evaluations

Several popular methods for *intrinsic* evaluations:

- Do (cosine) similarities of pairs of words' vectors correlate with judgments of similarity by humans?

Word Vector Evaluations

Several popular methods for *intrinsic* evaluations:

- Do (cosine) similarities of pairs of words' vectors correlate with judgments of 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:

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

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

Note: The above line contains corrections relative to the video, and the textbook.

Warning!

For both document vectors (e.g., from topic models) and word vectors, it's often interesting to visualize what the model has learned and inspect the space. Examples:

- List the words most strongly associated with each topic
- Start with a seed word and find its nearest neighbors in vector space

Human brains are uncannily good at finding patterns, even when they aren't there. Manual inspection of learned representations will make you feel good, but it doesn't mean your model is working well!

Extrinsic Evaluations

1. Use large unannotated corpus to get your word vectors (sometimes called **pretraining**).
2. Use them in a text classifier (or some other NLP system). Two options:
   - Plug in word vectors as "frozen" features, and estimate the other parameters of your model.
   - 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?

Taking Stock

So far, we've seen:

- Documents as vectors (LSI/A, then topic distributions)
- Words as vectors (LSI/A, then as parameters to the skip-gram model)

Next, a different approach to encoding words based on hierarchical clustering.

Brown Clustering

(Brown et al., 1992)

A greedy way to hierarchically cluster words based on distributional similarity.

Brown Clustering: Sketch of the Algorithm

Given: K (the desired number of clusters)

- Initially, every word v belongs to its own cluster, so that $z_i = x_i$.
- Repeat $V - K$ times:
  - Find the pairwise merge of two clusters z_A and z_B that gives the greatest value for $p(\boldsymbol{x}_{1:n}, \boldsymbol{z}_{1:n})$.
  - Assign every word in cluster z_A or z_B to new cluster z_{new}.

It turns out this is equivalent to using PMI on clusters of adjacent words to score potential merges.

This is very expensive; Brown et al. (1992) and others (later) introduced tricks for efficiency. See Liang (2005) and Stratos et al. (2014), for example.

Added Bonus to Brown Clusters

If you keep track of every merge, you have a *hierarchical* clustering.

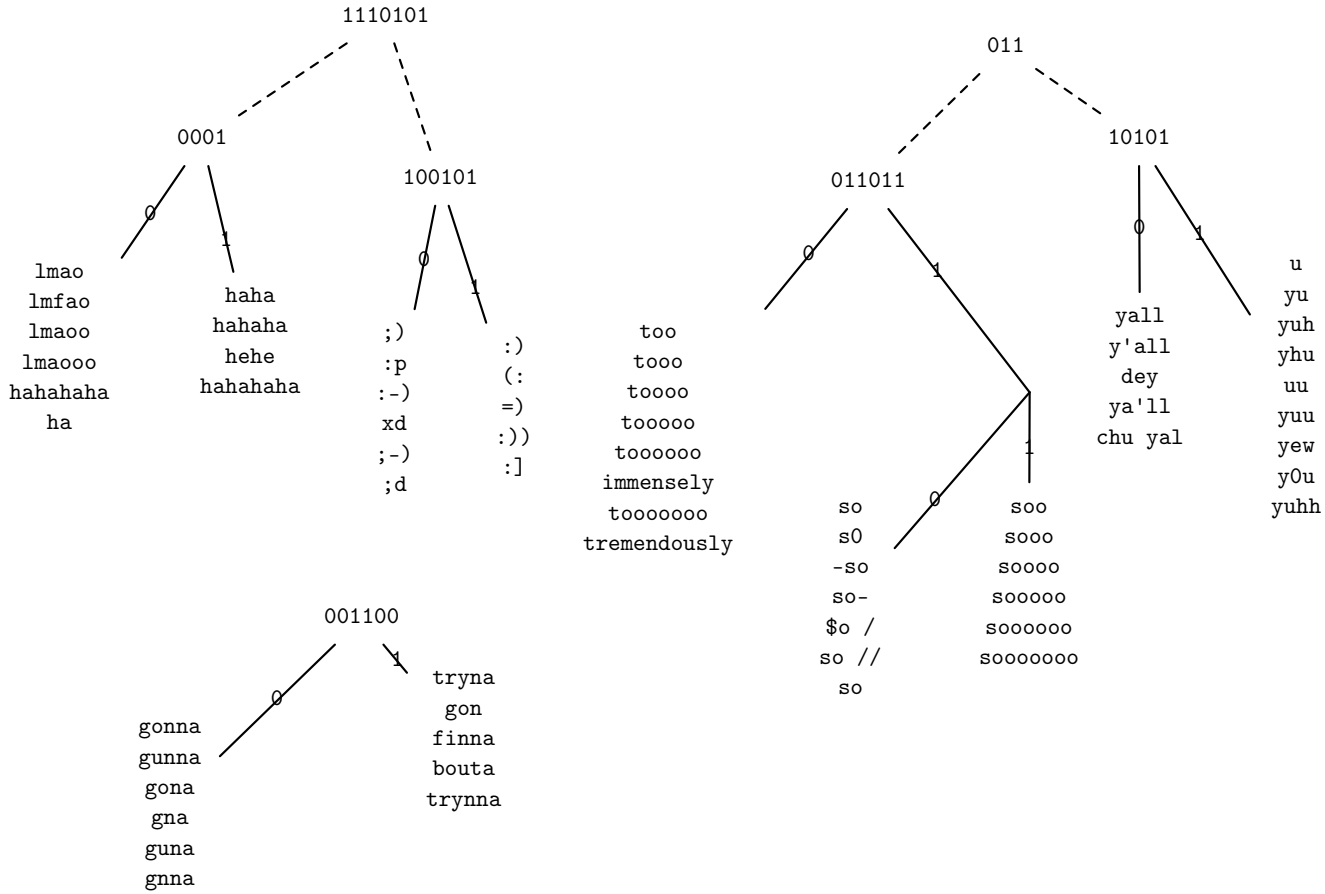
Each cluster is a binary tree with words at the leaves and internal nodes corresponding to merges.

Indexing the merge-pairs by 0 and 1 gives a bit-string for each word; prefixes of each word's bit string correspond to the hierarchical clusters it belongs to.

These can be seen as variable-length, binary word embeddings!

Brown Clusters from 56,000,000 Tweets

http://www.cs.cmu.edu/~ark/TweetNLP/cluster_viewer.html



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*

The Main Dish

Preliminaries

Token: an instance of a word observed in text

Type: the word in the abstract

(There are two tokens of type in above.)

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

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
   - Learned as parameters of a neural text classifier

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
   - Learned as parameters of a neural text classifier
   - Learned using methods discussed in this lecture (LSI/A, skip-gram, clustering)

What's Wrong?

Words' meanings, and the signals they give for different tasks, change in different contexts.

All of these models assume the same vector for every token of type v.

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, including ones we have never seen before (so it can't be a table lookup).
- 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.
  - Optional: 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.

Bidirectional Encoder Representations from Transformers (BERT)

(Devlin et al., 2019)

BERT—variants of which are currently used almost everywhere people are doing NLP—advanced over ELMo in several ways:

- Train on more data.
- Replace RNN architecture with transformer (deep model based on self-attention, mentioned last time)
- Rather than training left-to-right and right-to-left language models, train a "masked" language model:

$$\max_{\boldsymbol{\nu}} \sum_i \log p_{\boldsymbol{\nu}} \left(\boldsymbol{x}_i^{(\text{complete})} \mid \boldsymbol{x}_i^{(\text{masked})} \right)$$

where $\boldsymbol{x}_i^{(\text{masked})}$ is a sequence with 15% of its words randomly hidden. So, BERT is trained to predict some words in a sentence given others, with no regard for order.

Unwanted Associations

Always remember that a learned system is the product of both the learning algorithm and the data it was trained on.

Large corpora come from human societies, and those societies' assumptions and stereotypes will likely be picked up in word embeddings.

Bolukbasi et al. (2016) explored this in models trained on news articles, projecting words onto a "he–she" axis and finding considerable evidence for learned stereotypes.

They, and many since, have been developing methods for "debiasing" these methods.

Reflection

Given what you know about NLP so far, do you think it matters that learned representations of language might encode stereotypes? What might be the effects?

Current Research

- More data, deeper networks.
- Pretrain on alternative tasks to language modeling and masked language modeling, e.g., machine translation, or collections of tasks.
- Designing a "curriculum" that changes data and/or tasks over time.
- "Probing" representations to determine the extent to which various linguistic, commonsense, world, or domain knowledge is captured by the representations.

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`.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

Tolga Bolukbasi, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *NeurIPS*, 2016.

Peter F. Brown, Peter V. Desouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

Shay Cohen. *Bayesian Analysis in Natural Language Processing*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool, 2016.

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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of NAACL*, 2019. URL `https://www.aclweb.org/anthology/N19-1423`.

Jacob Eisenstein. *Introduction to Natural Language Processing*. MIT Press, 2019.

References II

Jacob Eisenstein, Brendan O'Connor, Noah A. Smith, and Eric P. Xing. A latent variable model for geographic lexical variation. In *Proc. of EMNLP*, 2010.

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.

Thomas Hofmann. Probabilistic latent semantic indexing. In *Proc. of SIGIR*, 1999.

Rukmini M. Iyer and Mari Ostendorf. Modeling long distance dependence in language: Topic mixtures versus dynamic cache models. *Speech and Audio Processing, IEEE Transactions on*, 7(1):30–39, 1999.

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

Percy Liang. Semi-supervised learning for natural language. Master's thesis, Massachusetts Institute of Technology, 2005.

Yishu Miao, Lei Yu, and Phil Blunsom. Neural variational inference for text processing. In *Proc. of ICML*, 2016.

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`.

References III

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

Michal Rosen-Zvi, Thomas Griffiths, Mark Steyvers, and Padhraic Smyth. The author-topic model for authors and documents. In *Proc. of UAI*, 2004.

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

Noah A. Smith. Contextual word representations: Putting words into computers. *CACM*, 2020. URL `https://arxiv.org/pdf/1902.06006`.

Karl Stratos, Do-kyum Kim, Michael Collins, and Daniel Hsu. A spectral algorithm for learning class-based n-gram models of natural language. In *Proc. of UAI*, 2014.