Language Models for Law and Social Science

5. Word Embeddings

Last week's quizz 4.2 (true/false)

1. An advantage of xgboost over elastic net or neural nets is that it is less likely to learn from confounding text characteristics, and therefore generalize better out of domain.
2. Rectified linear unit (ReLU) should be used as the activation function in MLPs.
3. Neural nets tend to out-perform xgboost on text classification tasks.
4. Early stopping means splitting into three sets (train, validation, test), and training the model until performance starts decreasing in the validation set.
5. Number of hidden layers, and number of neurons per layer, are hyperparameters that can be learned by cross-validation in the training set.

What have we been doing? ***Learning representations*** of the data

- Dictionary methods: document is represented as a count over the lexicon
- N-grams: document is a count over a vocabulary of phrases
- Topic models: document is a vector of shares over topics

What have we been doing? ***Learning representations*** of the data

- Dictionary methods: document is represented as a count over the lexicon
- N-grams: document is a count over a vocabulary of phrases
- Topic models: document is a vector of shares over topics
- Text classifiers: produces $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \hat{\theta})$, a vector of predicted probabilities across classes for each document i.

What have we been doing? ***Learning representations*** of the data

- Dictionary methods: document is represented as a count over the lexicon
- N-grams: document is a count over a vocabulary of phrases
- Topic models: document is a vector of shares over topics
- Text classifiers: produces $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \hat{\theta})$, a vector of predicted probabilities across classes for each document i.
  - The vector of class probabilities $\hat{\boldsymbol{y}}_i$ is a **compressed representation** of the outcome-predictive text features $\boldsymbol{x}_i$

What have we been doing? ***Learning representations*** of the data

- Dictionary methods: document is represented as a count over the lexicon
- N-grams: document is a count over a vocabulary of phrases
- Topic models: document is a vector of shares over topics
- Text classifiers: produces $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \hat{\theta})$, a vector of predicted probabilities across classes for each document i.
  - The vector of class probabilities $\hat{\boldsymbol{y}}_i$ is a **compressed representation** of the outcome-predictive text features $\boldsymbol{x}_i$
  - the vector of features, $\boldsymbol{x}_i$, is itself a compressed representation of the unprocessed document $\mathcal{D}_i$.

What have we been doing? ***Learning representations*** of the data

- Dictionary methods: document is represented as a count over the lexicon
- N-grams: document is a count over a vocabulary of phrases
- Topic models: document is a vector of shares over topics
- Text classifiers: produces $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i; \hat{\theta})$, a vector of predicted probabilities across classes for each document i.
  - The vector of class probabilities $\hat{\boldsymbol{y}}_i$ is a **compressed representation** of the outcome-predictive text features $\boldsymbol{x}_i$
  - the vector of features, $\boldsymbol{x}_i$, is itself a compressed representation of the unprocessed document $\mathcal{D}_i$.
- Further: the learned parameters $\hat{\theta}$ can also be understood as a **learned compressed representation of the whole dataset**:
  - it contains information about the training corpus, the text features, and the outcome classes.
- **B**, the $K \times V$ topic-word matrix in a topic model, is also such a compressed representation

Information in $\hat{\theta}$

Say we train a multinomial logistic regression on a bag-of-words representation $\boldsymbol{x}_i$ to predict classes $\boldsymbol{y}_i$:

- Let θ be the learned matrix of parameters relating input words to outcome classes:
  - It contains n_y columns, which are n_x-vectors representing outcome classes as weights over words.

Information in $\hat{\theta}$

Say we train a multinomial logistic regression on a bag-of-words representation $\boldsymbol{x}_i$ to predict classes $\boldsymbol{y}_i$:

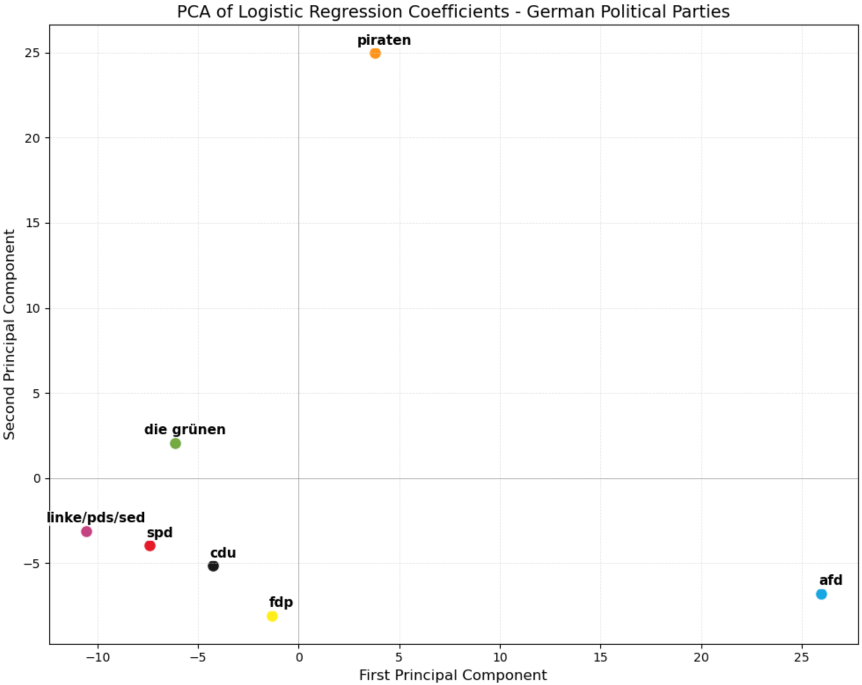
- Let θ be the learned matrix of parameters relating input words to outcome classes:
    - It contains n_y columns, which are n_x-vectors representing outcome classes as weights over words.
    - It contains n_x rows, which are n_y-vectors representing each word as weights over classes.

Information in $\hat{\theta}$

Say we train a multinomial logistic regression on a bag-of-words representation $\boldsymbol{x}_i$ to predict classes $\boldsymbol{y}_i$:

- Let θ be the learned matrix of parameters relating input words to outcome classes:
  - It contains n_y columns, which are n_x-vectors representing outcome classes as weights over words.
  - It contains n_x rows, which are n_y-vectors representing each word as weights over classes.
- θ is an interesting object. How can we use it?
- e.g.:
  - cluster the column vectors $\rightarrow$ which outcome classes are similar/related.
  - cluster the row vectors $\rightarrow$ which input features are similar/related.

Berlin Parliament Debates



$\hat{\theta}$ Contains Word Embeddings

$\theta =$ matrix of parameters learned from logit, relating words to outcomes.

- If $\boldsymbol{x}$ is a bag-of-words representation for a document consisting of a list of tokens $\{w_1, ..., w_t, ..., w_n\}$, we can write

$$\boldsymbol{x} = \frac{1}{n}\sum_{t=1}^{n} x_t$$

  - where x_t is an n_x-dimensional one-hot vector – all entries are zero except equals one for the word at t.

$\hat{\theta}$ Contains Word Embeddings

θ = matrix of parameters learned from logit, relating words to outcomes.

- If $\boldsymbol{x}$ is a bag-of-words representation for a document consisting of a list of tokens $\{w_1, ..., w_t, ..., w_n\}$, we can write

$$\boldsymbol{x} = \frac{1}{n}\sum_{t=1}^{n} x_t$$

  - where x_t is an n_x-dimensional one-hot vector – all entries are zero except equals one for the word at t.
- Let θ_t be the row of θ corresponding to the word w_t: a **word embedding** for w_t containing the outcome-relevant information for that word.

$\hat{\theta}$ Contains Word Embeddings

θ = matrix of parameters learned from logit, relating words to outcomes.

- If $\boldsymbol{x}$ is a bag-of-words representation for a document consisting of a list of tokens $\{w_1, ..., w_t, ..., w_n\}$, we can write

$$\boldsymbol{x} = \frac{1}{n}\sum_{t=1}^{n} x_t$$

  - where x_t is an n_x-dimensional one-hot vector – all entries are zero except equals one for the word at t.
- Let θ_t be the row of θ corresponding to the word w_t: a **word embedding** for w_t containing the outcome-relevant information for that word.
- We can construct a **document vector**

$$\vec{\boldsymbol{d}} = \sum_{t=1}^{n_i} \theta_t$$

the sum of the n_y-dimensional word representations (the row vectors from above).

  - this is called a "continuous bag of words (CBOW)" representation (Goldberg 2017).
  - Note that $\vec{\boldsymbol{d}} = \theta \cdot \boldsymbol{x}$, we thus call θ a **word embedding matrix**.

Outline

Word Embedding without Neural Nets

Embedding Layers

Word Embedding with Neural Nets

Word Embedding with Local Context

- Word2Vec and GloVe are two of the more well-known and commonly used models for producing word embeddings (vector representations of words).
  - the goal: represent the meaning of words by the neighboring words – their **contexts**.
  - these models get good performance on a range of similarity, analogy, and prediction tasks.

Word Embedding with Local Context

- Word2Vec and GloVe are two of the more well-known and commonly used models for producing word embeddings (vector representations of words).
  - the goal: represent the meaning of words by the neighboring words – their **contexts**.
  - these models get good performance on a range of similarity, analogy, and prediction tasks.
- "You shall know a word by the company it keeps" (Firth)
  - "He filled the **wampimuk**, passed it around and we all drunk some."
  - "We found a little, hairy **wampimuk** sleeping behind the tree."

Words and Contexts

Consider a **word-context matrix** $\boldsymbol{M}$:

- each row w represents a **word** (e.g. "income"), each column c represents a linguistic **context** in which a word can occur (e.g. "corporate ___ tax", "Eidgenössische ___ Hochschule").
  - A matrix entry $\boldsymbol{M}_{[w,c]}$ quantifies the strength of association between a word and a context in a large corpus
  - Similar to the input to topic models: there, the "context" is just the entire document

Words and Contexts

Consider a **word-context matrix $\boldsymbol{M}$**:

- each row w represents a **word** (e.g. "income"), each column c represents a linguistic **context** in which a word can occur (e.g. "corporate ___ tax", "Eidgenössische ___ Hochschule").
  - A matrix entry $\boldsymbol{M}_{[w,c]}$ quantifies the strength of association between a word and a context in a large corpus
  - Similar to the input to topic models: there, the "context" is just the entire document
- each word (row) $\boldsymbol{M}_{[w,:]}$ gives frequencies over contexts.
  - different definitions of contexts and different measures of association $\rightarrow$ different types of **word vectors**.
  - these vectors often have a **spatial interpretation** $\rightarrow$ geometric distances between word vectors reflect semantic distances between words.

Defining the context

- The simplest definition of context is neighboring words:
  - for "the tabby **cat**": we get $(w = \text{"cat"}, c = \text{"tabby"})$

Defining the context

- The simplest definition of context is neighboring words:
  - for “the tabby **cat**”: we get $(w = \text{"cat"}, c = \text{"tabby"})$
- Could extend this to words within a window of two:
  - add $(w = \text{"cat"}, c = \text{"the"})$
  - etc.

Defining the context

- The simplest definition of context is neighboring words:
  - for "the tabby **cat**": we get ($w =$ "cat", $c =$ "tabby")
- Could extend this to words within a window of two:
  - add ($w =$ "cat", $c =$ "the")
  - etc.
- Popular embeddings (word2vec and glove) generally use 5- or 10-word windows.
- Alternatives:
  - all words in the same sentence or same paragraph
  - syntactically connected words (from the parse tree)
  - Etc.

Defining an Association Measure

- Let $\boldsymbol{M}_{[w,c]} = f_M(w, c)$ where w and c are lookups to words in the w vocabulary and c vocabulary.
  - "word" could also mean phrases or more complicated objects.

Defining an Association Measure

- Let $\boldsymbol{M}_{[w,c]} = f_M(w,c)$ where w and c are lookups to words in the w vocabulary and c vocabulary.
  - "word" could also mean phrases or more complicated objects.
- e.g. **counts**: $f_M(w,c) = \#(w,c)$, the number of times w appeared along with context c, or **document frequencies**: $f_M(w,c) = \frac{\#(w,c)}{n_D}$

Defining an Association Measure

- Let $\boldsymbol{M}_{[w,c]} = f_M(w,c)$ where w and c are lookups to words in the w vocabulary and c vocabulary.
  - "word" could also mean phrases or more complicated objects.
- e.g. **counts**: $f_M(w,c) = \#(w,c)$, the number of times w appeared along with context c, or **document frequencies**: $f_M(w,c) = \frac{\#(w,c)}{n_D}$
- Downside: puts high weight on common contexts shared across many words (e.g., "the **cat** is" will be weighted higher than "the **cat** meowed")

Defining an Association Measure

- Let $\boldsymbol{M}_{[w,c]} = f_M(w,c)$ where w and c are lookups to words in the w vocabulary and c vocabulary.
  - "word" could also mean phrases or more complicated objects.
- e.g. **counts**: $f_M(w,c) = \#(w,c)$, the number of times w appeared along with context c, or **document frequencies**: $f_M(w,c) = \frac{\#(w,c)}{n_D}$
- Downside: puts high weight on common contexts shared across many words (e.g., "the **cat** is" will be weighted higher than "the **cat** meowed")
- Better: **Point-wise mutual information** (PMI):

$$f_M(w,c) = \frac{\Pr(w,c)}{\Pr(w)\Pr(c)} = \frac{\frac{\#(w,c)}{n_D}}{\frac{\#(w)}{n_D}\frac{\#(c)}{n_D}} = \frac{n_D\#(w,c)}{\#(w)\#(c)}$$

where $\#(w)$ and $\#(c)$ are the corpus counts for w and c, respectively.

$\boldsymbol{M}$ is too high-dimensional

- $\boldsymbol{M}$ is $n_w \times n_c$
    - if c is drawn from from the vocabulary of a reasonably large corpus, most associated word vectors $\{v_1 = \boldsymbol{M}_{[w_1,:]}, v_2 = \boldsymbol{M}_{[w_2,:]}, \ldots\}$ are very sparse, and will have near-zero (or zero) cosine similarity: hard to relate them to one another

$\boldsymbol{M}$ is too high-dimensional

- $\boldsymbol{M}$ is $n_w \times n_c$
  - if c is drawn from from the vocabulary of a reasonably large corpus, most associated word vectors $\{v_1 = \boldsymbol{M}_{[w_1,:]}, v_2 = \boldsymbol{M}_{[w_2,:]}, \ldots\}$ are very sparse, and will have near-zero (or zero) cosine similarity: hard to relate them to one another
- Going back to dimension reduction: can use singular value decomposition (SVD):
  - factorize $\boldsymbol{M} \in \mathbb{R}^{n_w \times n_c}$ into a word matrix $\boldsymbol{W} \in \mathbb{R}^{n_w \times n_E}$ and context matrix $\boldsymbol{C} \in \mathbb{R}^{n_c \times n_E}$
  - such that $\tilde{\boldsymbol{M}} = \boldsymbol{W}\boldsymbol{C}'$ is the best rank-n_E approximation of $\boldsymbol{M}$.
  - NB: topic models are *also* (bayesian, non-negative) matrix factorization of the document-term matrix: $\mathbf{D} \in \mathbb{N}^{n_w \times n_D}$

$\boldsymbol{M}$ is too high-dimensional

- $\boldsymbol{M}$ is $n_w \times n_c$
  - if c is drawn from from the vocabulary of a reasonably large corpus, most associated word vectors $\{v_1 = \boldsymbol{M}_{[w_1,:]}, v_2 = \boldsymbol{M}_{[w_2,:]}, \ldots\}$ are very sparse, and will have near-zero (or zero) cosine similarity: hard to relate them to one another
- Going back to dimension reduction: can use singular value decomposition (SVD):
  - factorize $\boldsymbol{M} \in \mathbb{R}^{n_w \times n_c}$ into a word matrix $\boldsymbol{W} \in \mathbb{R}^{n_w \times n_E}$ and context matrix $\boldsymbol{C} \in \mathbb{R}^{n_c \times n_E}$
  - such that $\tilde{\boldsymbol{M}} = \boldsymbol{W}\boldsymbol{C}'$ is the best rank-n_E approximation of $\boldsymbol{M}$.
  - NB: topic models are *also* (bayesian, non-negative) matrix factorization of the document-term matrix: $\mathbf{D} \in \mathbb{N}^{n_w \times n_D}$
  - $\tilde{\boldsymbol{M}}$ can be seen as a "smoothed" version of $\boldsymbol{M}$; "missing" values are filled in, etc.
  - example for a word-document matrix: cosine distance between "arbeit" and "arbeiten" on $\boldsymbol{M}$: 0.09. On $\boldsymbol{W}$: 0.45

$\boldsymbol{M}$ is too high-dimensional

- $\boldsymbol{M}$ is $n_w \times n_c$
  - if c is drawn from from the vocabulary of a reasonably large corpus, most associated word vectors $\{v_1 = \boldsymbol{M}_{[w_1,:]}, v_2 = \boldsymbol{M}_{[w_2,:]}, \ldots\}$ are very sparse, and will have near-zero (or zero) cosine similarity: hard to relate them to one another
- Going back to dimension reduction: can use singular value decomposition (SVD):
  - factorize $\boldsymbol{M} \in \mathbb{R}^{n_w \times n_c}$ into a word matrix $\boldsymbol{W} \in \mathbb{R}^{n_w \times n_E}$ and context matrix $\boldsymbol{C} \in \mathbb{R}^{n_c \times n_E}$
  - such that $\tilde{\boldsymbol{M}} = \boldsymbol{W}\boldsymbol{C}'$ is the best rank-n_E approximation of $\boldsymbol{M}$.
  - NB: topic models are *also* (bayesian, non-negative) matrix factorization of the document-term matrix: $\mathbf{D} \in \mathbb{N}^{n_w \times n_D}$
  - $\tilde{\boldsymbol{M}}$ can be seen as a "smoothed" version of $\boldsymbol{M}$; "missing" values are filled in, etc.
  - example for a word-document matrix: cosine distance between "arbeit" and "arbeiten" on $\boldsymbol{M}$: 0.09. On$\boldsymbol{W}$: 0.45
- $\boldsymbol{W}$ is the matrix of word vectors (word embeddings):
  - relatively low-dimensional ($n_E << n_w$, typically between 50 and 300)
  - dense, rather than sparse.

$\boldsymbol{M}$ is too high-dimensional

- $\boldsymbol{M}$ is $n_w \times n_c$
  - if c is drawn from from the vocabulary of a reasonably large corpus, most associated word vectors $\{v_1 = \boldsymbol{M}_{[w_1,:]}, v_2 = \boldsymbol{M}_{[w_2,:]}, \ldots\}$ are very sparse, and will have near-zero (or zero) cosine similarity: hard to relate them to one another
- Going back to dimension reduction: can use singular value decomposition (SVD):
  - factorize $\boldsymbol{M} \in \mathbb{R}^{n_w \times n_c}$ into a word matrix $\boldsymbol{W} \in \mathbb{R}^{n_w \times n_E}$ and context matrix $\boldsymbol{C} \in \mathbb{R}^{n_c \times n_E}$
  - such that $\tilde{\boldsymbol{M}} = \boldsymbol{W}\boldsymbol{C}'$ is the best rank-n_E approximation of $\boldsymbol{M}$.
  - NB: topic models are *also* (bayesian, non-negative) matrix factorization of the document-term matrix: $\mathbf{D} \in \mathbb{N}^{n_w \times n_D}$
  - $\tilde{\boldsymbol{M}}$ can be seen as a "smoothed" version of $\boldsymbol{M}$; "missing" values are filled in, etc.
  - example for a word-document matrix: cosine distance between "arbeit" and "arbeiten" on $\boldsymbol{M}$: 0.09. On $\boldsymbol{W}$: 0.45
- $\boldsymbol{W}$ is the matrix of word vectors (word embeddings):
  - relatively low-dimensional ($n_E << n_w$, typically between 50 and 300)
  - dense, rather than sparse.
- **similarity measures between rows of $\boldsymbol{W}$ approximate similarity measures between rows of $\boldsymbol{M}$**

GloVe Embeddings

Pennington et al (2014) (GloVe = Global Vectors) learns vectors without a neural net

- Input: C_{ij} = local co-occurrence counts between words $i, j \in \{1, ..., n_w\}$ within some co-occurence window, e.g. ten words.

GloVe Embeddings

Pennington et al (2014) (GloVe = Global Vectors) learns vectors without a neural net

- Input: C_{ij} = local co-occurrence counts between words $i, j \in \{1, \ldots, n_w\}$ within some co-occurence window, e.g. ten words.

Learn word vectors $\boldsymbol{w} = (w_1, \ldots, w_i, \ldots, w_{n_w})$, initialized randomly and $w_i \in (-1, 1)^{n_E}$, to solve

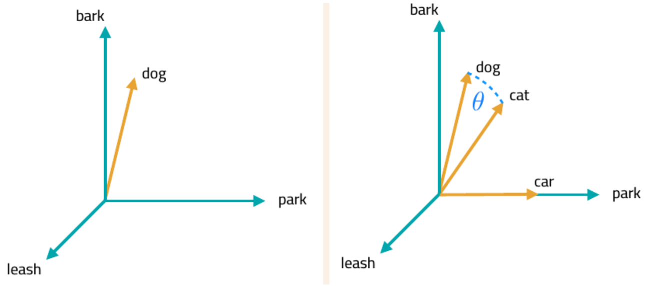
$$\min_{\boldsymbol{w}} \sum_{i,j} f(C_{ij}) \left(w_i^T w_j - \log(C_{ij}) \right)^2$$

where $f(\cdot)$ is a weighting function to down-weight frequent words.

- Minimizes **squared difference** between**:**
  - **dot product of word vectors,** $w_i^T w_j$
  - **empirical co-occurrence,** $\log(C_{ij})$
    [Arora et al (2016) put the PMI here instead of co-occurence counts]
- Intuitively: words that co-occur should have high correlation (dot product)

Word Similarity

- Once words are represented as vectors, we can use linear algebra to understand the relationships between words:
  - Words that are geometrically close to each other are similar: e.g. "dog" and "cat":

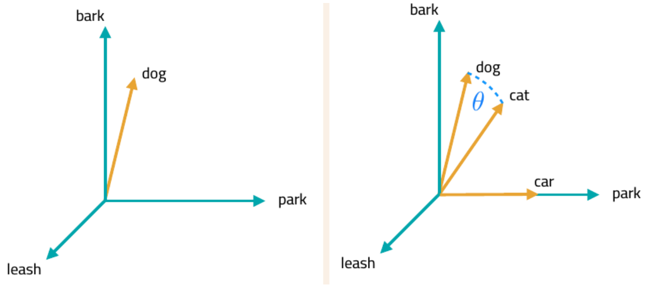


- The standard metric for comparing vectors is cosine similarity:

$$\cos\theta = \frac{w_1 \cdot w_2}{||w_1||||w_2||}$$

Word Similarity

- Once words are represented as vectors, we can use linear algebra to understand the relationships between words:
  - Words that are geometrically close to each other are similar: e.g. "dog" and "cat":



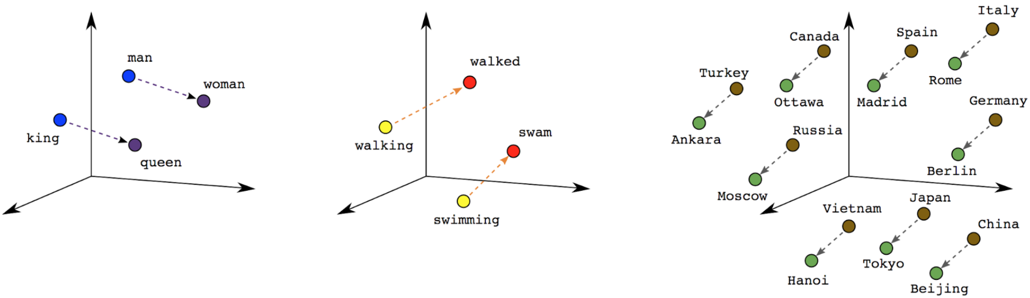
- The standard metric for comparing vectors is cosine similarity:

$$\cos\theta = \frac{w_1 \cdot w_2}{||w_1||||w_2||}$$

- Thanks to linearity, can compute similarities between groups of words by averaging the groups.

Vector Directions ↔ Meaning

- Intriguingly, word embedding directions capture conceptual, analogical relationships between words:



Male-Female

Verb Tense

Country-Capital

Word Embeddings for Analogies

$$vec(king) - vec(man) + vec(woman) \approx vec(queen)$$

Word Embeddings for Analogies

$$vec(king) - vec(man) + vec(woman) \approx vec(queen)$$

- More generally: The analogy $a_1 : b_1 :: a_2 : b_2$ can be solved (that is, find b_2 given a_1, b_1, a_2) by

$$\arg\max_{b_2 \in V} \cos(b_2, a_2 - a_1 + b_1)$$

  where V excludes (a_1, b_1, a_2).

Word Embeddings for Analogies

$$vec(king) - vec(man) + vec(woman) \approx vec(queen)$$

- More generally: The analogy $a_1 : b_1 :: a_2 : b_2$ can be solved (that is, find b_2 given a_1, b_1, a_2) by

$$\arg\max_{b_2 \in V} \cos(b_2, a_2 - a_1 + b_1)$$

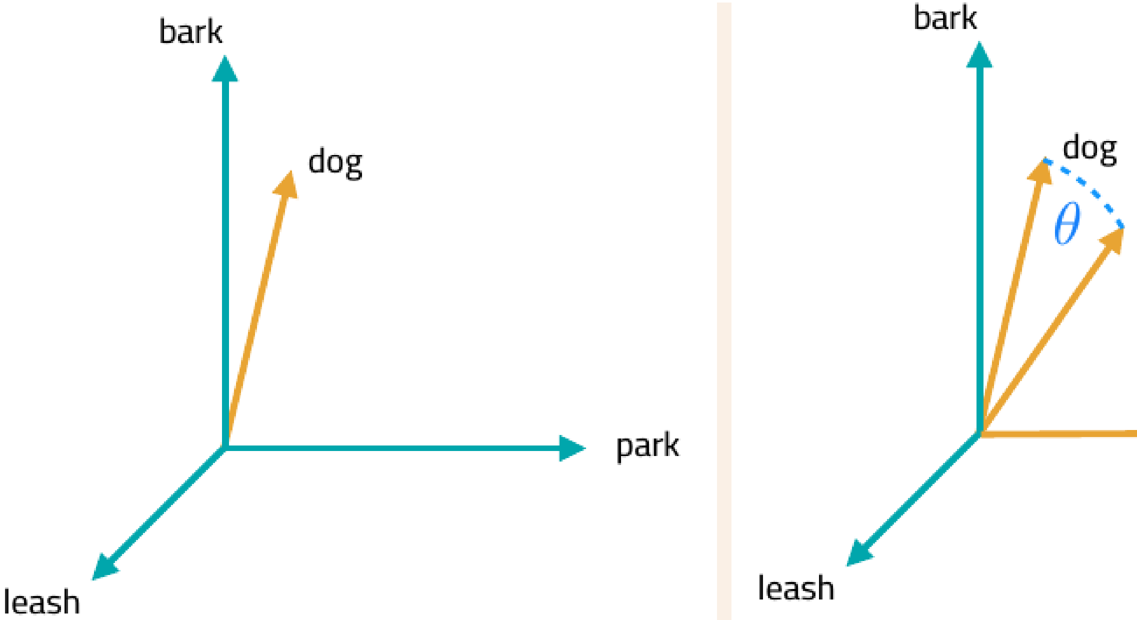
  where V excludes (a_1, b_1, a_2).
- Often works better with normalized vectors (so that one long vector doesn't wash out the others)

Word Embeddings for Analogies

$$vec(king) - vec(man) + vec(woman) \approx vec(queen)$$

- More generally: The analogy $a_1 : b_1 :: a_2 : b_2$ can be solved (that is, find b_2 given a_1, b_1, a_2) by
  $$\arg\max_{b_2 \in V} \cos(b_2, a_2 - a_1 + b_1)$$
  where V excludes (a_1, b_1, a_2).
- Often works better with normalized vectors (so that one long vector doesn't wash out the others)
- Levy and Goldberg (2014) recommend the following "CosMul" metric which tends to perform better:
  $$\arg\max_{b_2 \in V} \frac{\cos(b_2, a_2)\cos(b_2, b_1)}{\cos(b_2, a_1) + \epsilon}$$
  - requires normalized, non-negative vectors (can transform using $(x + 1)/2$)
  - ϵ is a small smoothing parameter.

Kozlowski, Evans, and Taddy (ASR 2019)



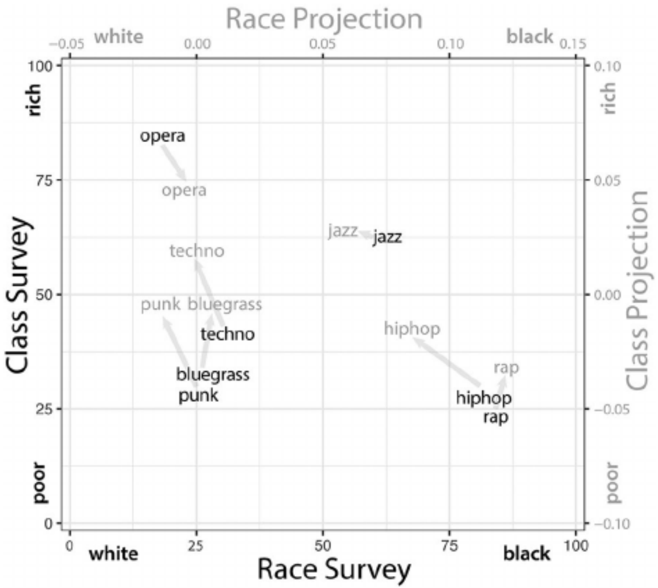


Figure 3. Projection of Music Genres onto Race and Class Dimensions of the Google News Word Embedding (Gray) and Average Survey Ratings for Race and Class Associations (Black)

“Rejecting the Gender Binary”

```
## [17] "father->mother"          "genius->goddess"
## [19] "arrogant->snobby"        "priest->nun"
## [21] "dork->ditz"              "handsome->gorgeous"
## [23] "atheist->feminist"       "himmmm->herrrr"
## [25] "kermit->degeneres"       "mans->womans"
## [27] "hez->shez"               "himmm->herrr"
## [29] "trumpet->flute"          "checkride->clinicals"
## [31] "gay->lesbian"            "surgeon->nurse"
## [33] "daddy->mommy"            "cool->sweet"
## [35] "monsieur->mme"           "jolly->cheerful"
## [37] "jazz->dance"             "wears->outfits"
```

Video Presentation

Outline

Word Embedding without Neural Nets

Embedding Layers

Word Embedding with Neural Nets

What is an Embedding?

- In a broad sense, **“embedding”** refers to a lower-dimensional dense vector representation of a higher-dimensional object.
  - in NLP, this higher-dimensional object will be a word, sentence, or document.

What is an Embedding?

- In a broad sense, **“embedding”** refers to a lower-dimensional dense vector representation of a higher-dimensional object.
  - in NLP, this higher-dimensional object will be a word, sentence, or document.
- Not embeddings:
  - counts over LIWC dictionary categories.
  - sklearn CountVectorizer count vectors

What is an Embedding?

- In a broad sense, **"embedding"** refers to a lower-dimensional dense vector representation of a higher-dimensional object.
  - in NLP, this higher-dimensional object will be a word, sentence, or document.
- Not embeddings:
  - counts over LIWC dictionary categories.
  - sklearn CountVectorizer count vectors
- Embeddings:
  - PCA reductions of the word count vectors
  - LDA document-topic distributions ($\theta_d \in \Delta^{k-1}$)

Categorical Embeddings = dense representations of categorical variables

Say we have a binary classification problem with outcome Y:

- we have a high-dimensional categorical variable (e.g. area of law with 1000 categories)
- sparsity may mean that there are few training examples for certain categories
- including dummy variables A for each category in your ML model is computationally expensive

Categorical Embeddings = dense representations of categorical variables

Say we have a binary classification problem with outcome Y:

- we have a high-dimensional categorical variable (e.g. area of law with 1000 categories)
- sparsity may mean that there are few training examples for certain categories
- including dummy variables A for each category in your ML model is computationally expensive

Embedding approaches:

1. unsupervised: PCA applied to the dummy variables A to get lower-dimensional $\tilde{A}$.

Categorical Embeddings = dense representations of categorical variables

Say we have a binary classification problem with outcome Y:

- we have a high-dimensional categorical variable (e.g. area of law with 1000 categories)
- sparsity may mean that there are few training examples for certain categories
- including dummy variables A for each category in your ML model is computationally expensive

Embedding approaches:

1. unsupervised: PCA applied to the dummy variables A to get lower-dimensional $\tilde{A}$.
2. supervised: Regress Y on A, predict $\hat{Y}(A_i)$, add $\hat{Y}(A_i)$ as a predictor in your model instead of A.

Categorical Embeddings = dense representations of categorical variables

Say we have a binary classification problem with outcome Y:

- we have a high-dimensional categorical variable (e.g. area of law with 1000 categories)
- sparsity may mean that there are few training examples for certain categories
- including dummy variables A for each category in your ML model is computationally expensive

Embedding approaches:

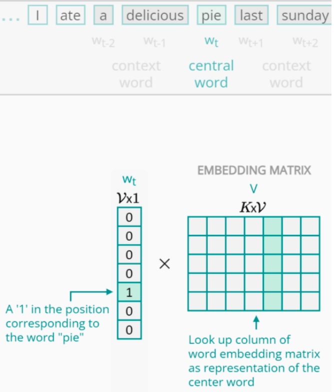
1. unsupervised: PCA applied to the dummy variables A to get lower-dimensional $\tilde{A}$.
2. supervised: Regress Y on A, predict $\hat{Y}(A_i)$, add $\hat{Y}(A_i)$ as a predictor in your model instead of A.

(2) is similar to what embedding layers do in neural nets.

In deep learning, an embedding layer is matrix multiplication:

$$\underbrace{h_1}_{n_E \times 1} = \underbrace{\omega_E}_{n_E \times n_w} \cdot \underbrace{x}_{n_x \times 1}$$

- x = a categorical variable (e.g., representing a word)
  - one-hot vector with a single item equaling one. Input to the embedding layer.
- ω_E = the matrix of learnable parameters.
- h_1 = the first hidden layer of the neural net
  - The output of the embedding layer



The embedding matrix ω_E encodes predictive information about the categories, has a spatial interpretation when projected to two dimensions.

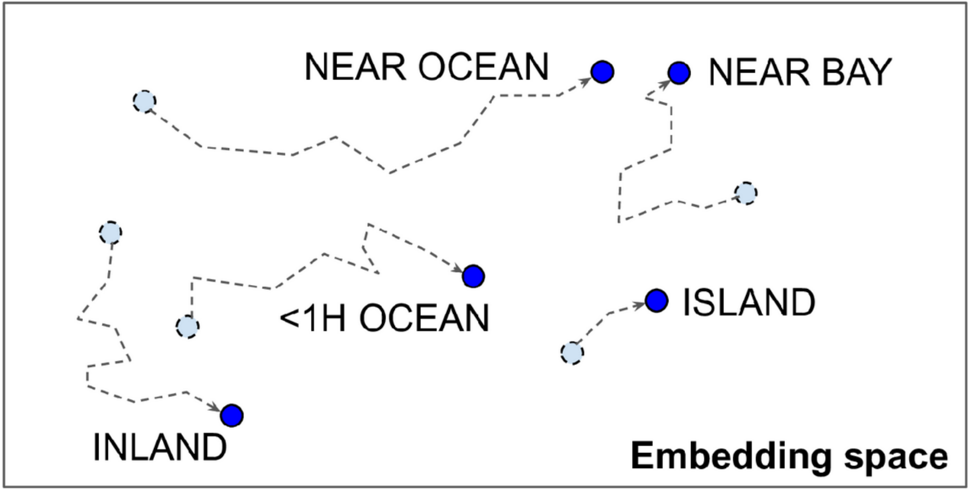


Figure 13-4. Embeddings will gradually improve during training

Embedding Layers versus Dense Layers

- An embedding layer is statistically equivalent to a fully-connected dense layer with one-hot vectors as input and identity activation function.
  - implementations of embedding layers just involve a lookup (indexing by the category indicator) and are therefore faster; should use them when you have 10 or more categories.

Video Presentation

Outline

Word Embeddings = NN layers that map word indexes to dense vectors.

Word Embeddings = NN layers that map word indexes to dense vectors.

- Documents are lists of word indexes $\{w_1, w_2, ..., w_{n_i}\}$.
  - equivalently, let w_i be a one-hot vector (dimensionality n_w = vocab size) where the associated word's index equals one.

Word Embeddings = NN layers that map word indexes to dense vectors.

- Documents are lists of word indexes $\{w_1, w_2, ..., w_{n_i}\}$.
  - equivalently, let w_i be a one-hot vector (dimensionality n_w = vocab size) where the associated word's index equals one.
  - Normalize all documents to the same length L, with shorter documents padded with a null token. (This will be relaxed later.)

Word Embeddings = NN layers that map word indexes to dense vectors.

- Documents are lists of word indexes $\{w_1, w_2, ..., w_{n_i}\}$.
  - equivalently, let w_i be a one-hot vector (dimensionality n_w = vocab size) where the associated word's index equals one.
  - Normalize all documents to the same length L, with shorter documents padded with a null token. (This will be relaxed later.)
- Embedding layer replaces the list of sparse one-hot vectors with a list of n_E-dimensional ($n_E << n_w$) dense vectors

$$\boldsymbol{X} = \begin{bmatrix} x_1 & \dots & x_L \end{bmatrix}$$

where

$$\underbrace{x_j}_{n_E \times 1} = \underbrace{\boldsymbol{E}}_{n_E \times n_w} \cdot \underbrace{w_j}_{n_w \times 1}$$

  - $\boldsymbol{E}$ is a matrix of word vectors. The column associated with the word at j is selected by dot-product with one-hot vector w_j.

Word Embeddings = NN layers that map word indexes to dense vectors.

- Documents are lists of word indexes $\{w_1, w_2, ..., w_{n_i}\}$.
  - equivalently, let w_i be a one-hot vector (dimensionality n_w = vocab size) where the associated word's index equals one.
  - Normalize all documents to the same length L, with shorter documents padded with a null token. (This will be relaxed later.)
- Embedding layer replaces the list of sparse one-hot vectors with a list of n_E-dimensional ($n_E << n_w$) dense vectors

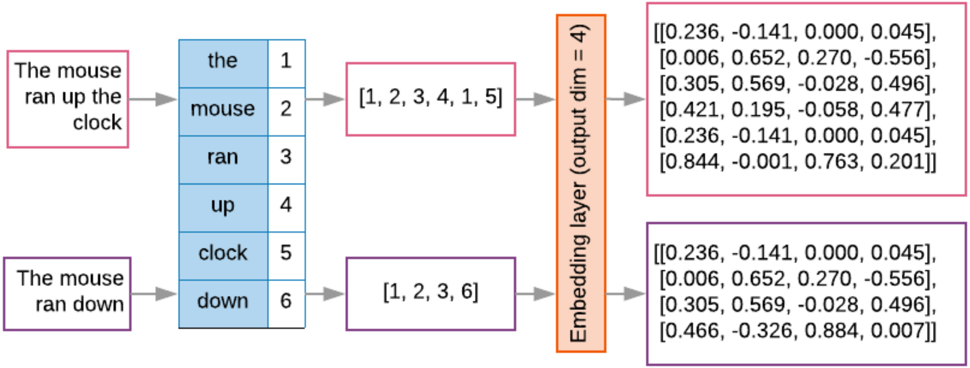
$$\boldsymbol{X} = \begin{bmatrix} x_1 & \ldots & x_L \end{bmatrix}$$

where

$$\underbrace{x_j}_{n_E \times 1} = \underbrace{\boldsymbol{E}}_{n_E \times n_w} \cdot \underbrace{w_j}_{n_w \times 1}$$

  - $\boldsymbol{E}$ is a matrix of word vectors. The column associated with the word at j is selected by dot-product with one-hot vector w_j.
- $\boldsymbol{X}$ is flattened into an $L * n_E$ vector for input to the next layer.

Illustration



Word2Vec

- "Word2Vec" is a neural net model that, instead of predicting some metadata (such as classifying topic labels), predicts the co-occurence of neighboring words.
  - an example of **"self-supervision"**

Word2Vec

- "Word2Vec" is a neural net model that, instead of predicting some metadata (such as classifying topic labels), predicts the co-occurence of neighboring words.
  - an example of **"self-supervision"**
- How does it learn the meaning of the word "fox"?
  - By comparing true instances of the word fox ("The <u>quick brown</u> **fox** <u>jumps over</u> the lazy dog")
  - to fake (randomly sampled) ones ("The <u>prescription of</u> **fox** <u>is advised</u> for this diagnosis")

Word2Vec

- "Word2Vec" is a neural net model that, instead of predicting some metadata (such as classifying topic labels), predicts the co-occurence of neighboring words.
  - an example of **"self-supervision"**
- How does it learn the meaning of the word "fox"?
  - By comparing true instances of the word fox ("The quick brown **fox** jumps over the lazy dog")
  - to fake (randomly sampled) ones ("The prescription of **fox** is advised for this diagnosis")
- Word2Vec learns embedding vectors for the target word ("fox") and context words (neighbors of "fox") to distinguish true from false samples.

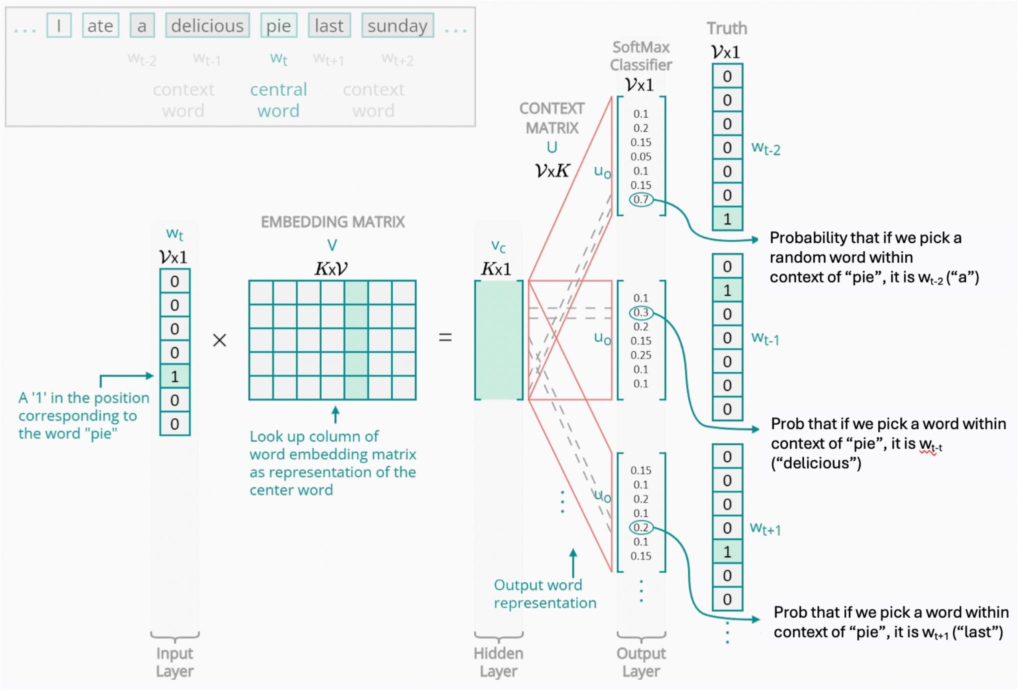


Image credit: Lorena Calvo-Bartolomé

Word2Vec Skip-Gram Objective

- Word2Vec has two variants: skip-gram,$P(c \mid w)$, and continuous bag-of-words,$P(w \mid c)$
- Skip-gram: probability of seeing context given a word
  - $P(c \mid w) = \frac{P(w,c)}{P(c)} = \frac{P(w,c)}{\sum_{c' \in \mathcal{V}} P(w,c')}$
- Which we operationalize as
  - $P(c \mid w) = \frac{exp(\mathbf{u}_w^T \cdot \mathbf{v}_c)}{\sum_{c' \in \mathcal{V}} exp(\mathbf{u}_{w'}^T \cdot \mathbf{v}_c)}$, where $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{|\mathcal{V}| \times d}$ are learned embedding matrices
- Problem! This summation is expensive—have to normalize over the whole vocabulary.
  - Solution: negative sampling

Skip-Gram with Negative Sampling

- We have a binary classification problem: predict whether the (word, context) pairs were actually observed (or not) by sampling "fake" negative examples
- The dataset is a collection of context pairs indexed by i:
  - $y_i = 1$ means correct (it appeared in the corpus)
  - $y_i = 0$ means incorrect (it was randomly drawn $\rightarrow$ **negative sample**)
- The target and context words are looked up in two embedding matrices, resulting in $\mathbf{u}_w, \mathbf{v}_c$
  - These are passed through a sigmoid

$$\hat{y}(w,c) = \frac{1}{1 + \exp(-\mathbf{u}_w \cdot \mathbf{v}_c)}$$

which gives the predicted probability of a correct rather than random pair.

Skip-Gram with Negative Sampling

- We have a binary classification problem: predict whether the (word, context) pairs were actually observed (or not) by sampling "fake" negative examples
- The dataset is a collection of context pairs indexed by i:
  - $y_i = 1$ means correct (it appeared in the corpus)
  - $y_i = 0$ means incorrect (it was randomly drawn $\rightarrow$ **negative sample**)
- The target and context words are looked up in two embedding matrices, resulting in $\mathbf{u}_w, \mathbf{v}_c$
  - These are passed through a sigmoid

$$\hat{y}(w,c) = \frac{1}{1+\exp(-\mathbf{u}_w \cdot \mathbf{v}_c)}$$

    which gives the predicted probability of a correct rather than random pair.

- Word2Vec objective is then to minimize the binary cross-entropy (as in classification)

$$\min_{\mathbf{u}_w,\mathbf{v}_c} L(w,c) = -\log \hat{y}_i(w,c) - \sum_{c' \sim P(\mathcal{V})}^{n_s} \log(1-\hat{y}(w,c'))$$

How does Word2Vec relate to the $\boldsymbol{M}$ matrix?

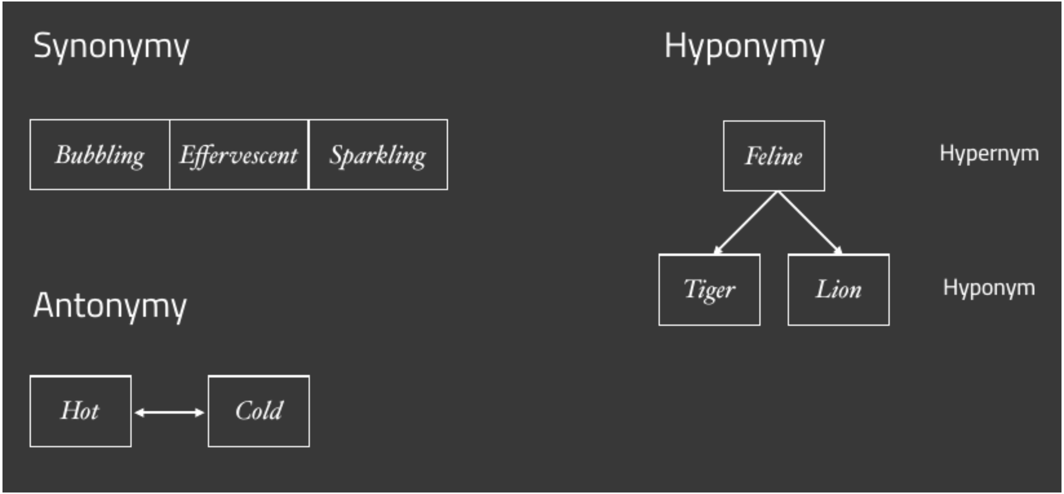
- Word2Vec produces embedding matrices $\boldsymbol{W}$ and $\boldsymbol{C}$.
  - generally, context embeddings are discarded after training.
- Levy and Goldberg (2014):
  - If we take $\tilde{\boldsymbol{M}} = \boldsymbol{W}\boldsymbol{C}'$, word2vec is equivalent to factorizing a matrix $\boldsymbol{M}$ with items

    $$\boldsymbol{M}_{[w,c]} = \mathrm{PMI}(w, c) - \log a$$

    where a is a constant calibrating the amount of negative sampling.

Word Embeddings Encode Linguistic Relations

Word Embeddings Encode Linguistic Relations



Similarity vs. Relatedness (Budansky and Hirst, 2006)

- Semantic **similarity**: words sharing salient attributes / features
  - synonymy (car / automobile)
  - hypernymy (car / vehicle)
  - co-hyponymy (car / van / truck)

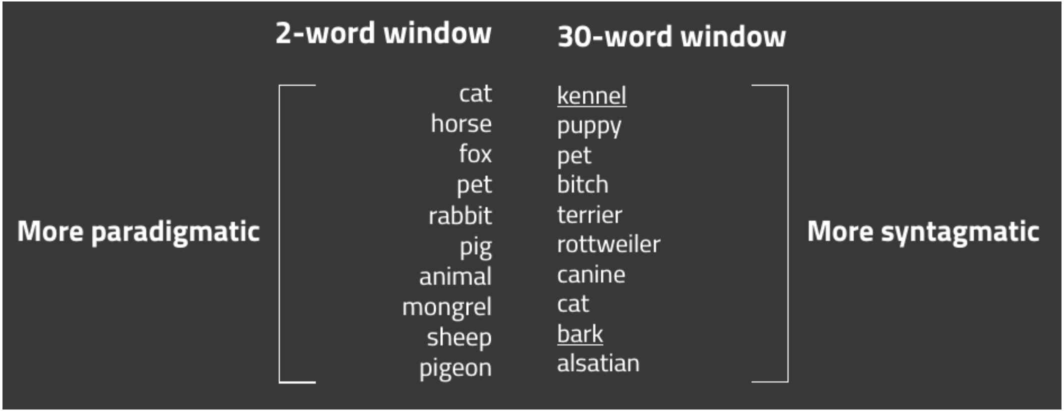
Similarity vs. Relatedness (Budansky and Hirst, 2006)

- Semantic **similarity**: words sharing salient attributes / features
  - synonymy (car / automobile)
  - hypernymy (car / vehicle)
  - co-hyponymy (car / van / truck)
- Semantic **relatedness**: words semantically associated without necessarily being similar
  - function (car / drive)
  - meronymy (car / tire)
  - location (car / road)
  - attribute (car / fast)

Similarity vs. Relatedness (Budansky and Hirst, 2006)

- Semantic **similarity**: words sharing salient attributes / features
  - synonymy (car / automobile)
  - hypernymy (car / vehicle)
  - co-hyponymy (car / van / truck)
- Semantic **relatedness**: words semantically associated without necessarily being similar
  - function (car / drive)
  - meronymy (car / tire)
  - location (car / road)
  - attribute (car / fast)
- Word embeddings will recover one or both of these relations, depending on how contexts and associated are constructed.

Most similar words to dog, depending on context window size



- Small windows pick up substitutable words; large windows pick up topics.

Pre-trained word embeddings

- In many settings (e.g. a small corpus), better to use pre-trained embeddings.
- e,g, spaCy's GloVe embeddings:
  - one million vocabulary entries
  - 300-dimensional vectors
  - trained on the Common Crawl corpus
- Can initialize models with pre-trained embeddings, can fine-tune as needed.

Parts of Speech and Phrases

- In the default model multiple senses of a word are merged.
  - e.g. "I like a bird" (verb) and "I am like a bird" (preposition).

Parts of Speech and Phrases

- In the default model multiple senses of a word are merged.
  - e.g. "I like a bird" (verb) and "I am like a bird" (preposition).
- Can improve the quality of embeddings in these cases by attaching the POS to the word (e.g. "like:verb", "like:prep") before training.

Parts of Speech and Phrases

- In the default model multiple senses of a word are merged.
  - e.g. "I like a bird" (verb) and "I am like a bird" (preposition).
- Can improve the quality of embeddings in these cases by attaching the POS to the word (e.g. "like:verb", "like:prep") before training.
- The default model only works by word, but "new york $\neq$ "new" + "york"
  - can tokenize phrases together (see Week 2 lecture) before training.

The black sheep problem

- The trivial or obvious features of a word are not mentioned in standard corpora.
- For example, although most sheep are white, you rarely see the phrase “white sheep”.
  - so word2vec sometimes tells you sim(black,sheep) $>$ sim(white,sheep).

The black sheep problem

- The trivial or obvious features of a word are not mentioned in standard corpora.
- For example, although most sheep are white, you rarely see the phrase "white sheep".
  - so word2vec sometimes tells you sim(black,sheep) $>$ sim(white,sheep).
- This is really important when interpreting results using embeddings to anayze beliefs/attitudes.
- Relatedly, antonyms are often rated similarly, have to be careful with that.

Review: NLP “Bias” is statistical bias

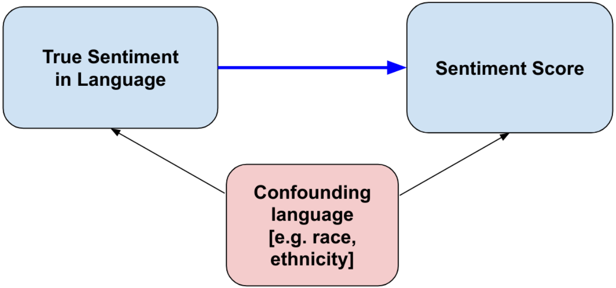
- Self-supervised learning algorithms like Word2Vec learn ***all*** dimensions of word associations; not just ones we are most interested in.
  - e.g., true expressions of attitudes or perceptions.

Review: NLP "Bias" is statistical bias

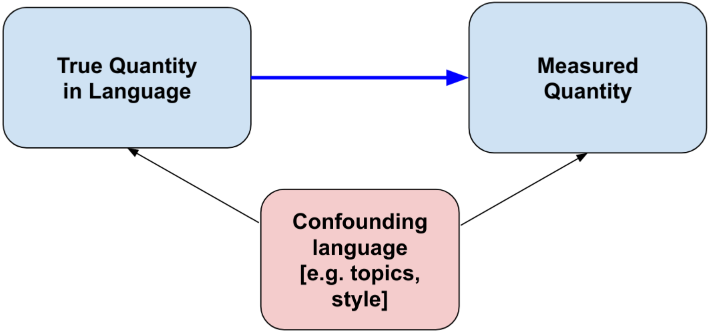
- Self-supervised learning algorithms like Word2Vec learn ***all*** dimensions of word associations; not just ones we are most interested in.
  - e.g., true expressions of attitudes or perceptions.
- e.g., using embeddings to scale social group words in a positive-to-negative dimension can learn correlated associations, not just sincere expressions of such attitudes:

```
text_to_sentiment("Let's go get Italian food")
2.0429166109
text_to_sentiment("Let's go get Chinese food")
1.4094033658
text_to_sentiment("Let's go get Mexican food")
0.3880198556
```

```
text_to_sentiment("My name is Emily")
2.2286179365
text_to_sentiment("My name is Heather")
1.3976291151
text_to_sentiment("My name is Yvette")
0.9846380213
text_to_sentiment("My name is Shaniqua")
-0.4704813178
```

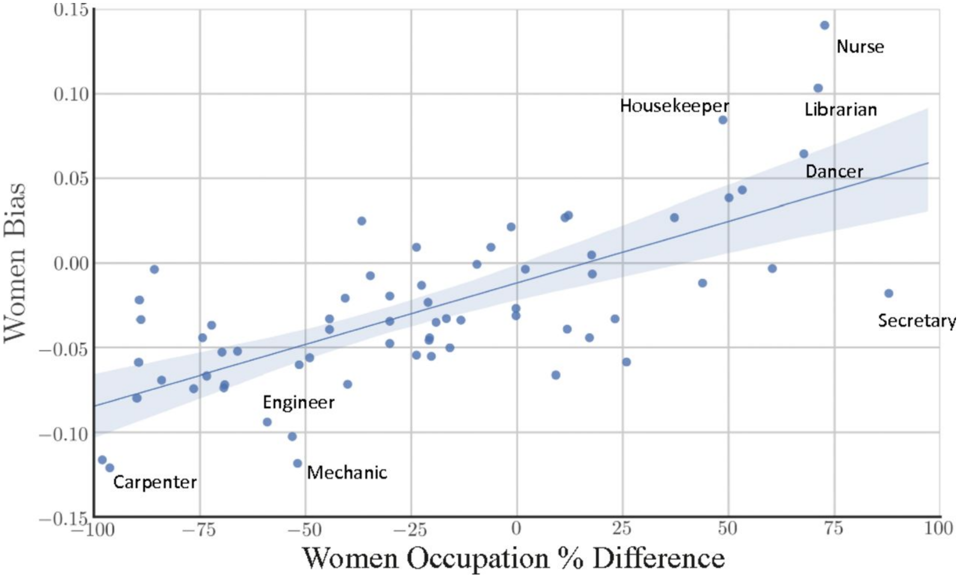


Self-Supervised Models Learn Confounders



- self-supervised language models like Word2Vec learn all lingusitic associations in language.
  - the measured associations might reflect attitudes/perceptions, or might reflect something else.

Garg, Schiebinger, Jurafsky, and Zou (PNAS 2018)



Women's occupation relative percentage vs. embedding bias in Google News vectors.

Steps for de-biasing

- Language features that are often confounded with the quantity of interest:
  - stopwords
  - named entities: person/organization/place names
- These can be dropped during pre-processing to reduce the influence of confounders in subsequent measurements.
- Can control for topic or style features or other potential confounders in regressions, or shuffle named entities.

De-Biasing Word Embeddings

De-Biasing Word Embeddings

- Bolukbasi et al (NIPS 2016):
  - "Geometrically, **gender bias is first shown to be captured by a direction in the word embedding**."
  - "Second, gender neutral words are shown to be linearly separable from gender definition words in the word embedding."
    - Embeddings for "secretary" and "nurse" closer to embeddings for "woman" than "man"
    - Perhaps useful for understanding training data, but undesirable for downstream applications
  - "Using these properties, we provide a methodology for modifying an embedding to remove gender stereotypes, such as the association between the words receptionist and female, while maintaining desired associations such as between the words queen and female."

De-Biasing Word Embeddings

- Bolukbasi et al (NIPS 2016):
  - "Geometrically, **gender bias is first shown to be captured by a direction in the word embedding**."
  - "Second, gender neutral words are shown to be linearly separable from gender definition words in the word embedding."
    - Embeddings for "secretary" and "nurse" closer to embeddings for "woman" than "man"
    - Perhaps useful for understanding training data, but undesirable for downstream applications
  - "Using these properties, we provide a methodology for modifying an embedding to remove gender stereotypes, such as the association between the words receptionist and female, while maintaining desired associations such as between the words queen and female."
- But: Gonen and Goldberg (2019):
  - ***"... we argue that this removal is superficial. While the bias is indeed substantially reduced according to the provided bias definition, the actual effect is mostly hiding the bias, not removing it. The gender bias information is still reflected in the distances between 'gender-neutralized' words in the debiased embeddings, and can be recovered from them..."***
- Still more work from Ravfogel et al. (multiple papers)

Tokenizing for Word Embeddings

- capitalization?
- punctuation?
- stopwords/function-words?
- can add special tokens for start of sentence and end of sentence
- for out-of-vocab words, substitute a special token or replace with part-of-speech tag
  - or use FastText embeddings (more below)

Can cluster word embeddings to produce topics

Cluster #	Top 10 Words
174	complicate, depend, crucial, illustrate, elusive, focus, important, straightforward, elide, critical
134	implausible, problematic, exaggeration, skeptical, ascribe, discredit, contradictory, weak, exaggerate, supportable
75	reverse, AFFIRM, affirm, vacate, reversed, REMANDED, forego, foregoing, forgoing, remands
70	importation, import, ecstasy, marihuana, illicit, opium, distilled, export, phencyclidine, narcotic
178	perverse, sensible, tempt, unlikely, unwise, anomalous, would, easy, costly, attractive
32	phrase, meaning, word, synonymous, language, interpret, noun, wording, verb, adjective
169	circumscribe, endow, unfettered, vest, unlimited, boundless, broad, constrain, exercise, unbounded
85	hundred, thousand, many, million, huge, massive, large, enormous, most, dozen
28	emphasis, bracket, alteration, citation, footnote, italic, ellipsis, petcitation, idcitation, punctuation
138	logo, symbol, stylized, imprint, emblem, grille, prefix, lettering, suffix, crosshair
181	wilful, carelessness, recklessness, careless, intentional, willful, conscious, reckless, unintentional, wantonness
158	rigorous, demanding, heightened, reasonableness, rigid, heighten, objective, deferential, flexible, particular
55	agreement, contract, contractual, promise, novation, repudiate, guaranty, enforceable, novate, repurchase
197	summation, admonish, sidebar, prosecutor, admonishment, mistrial, curative, questioning, remark, recess
120	scrivener, typographical, reversible, plain, harmless, clerical, invited, clear, requiresthe, instructional
15	adjudicatory, adjudicative, adversarial, judicial, rulemaking, decisionmaking, administrative, meaningful, rulemake, agency

Clustered word embeddings in judicial opinions, from Ash and Nikolaus (2020)

“Enriching word vectors with subword information” (Bojanowski et al 2017)

- each word is represented as a bag of (hashed) character n-grams. (e.g., spicy = (spi, pic, icy)).
- learn embeddings for the character segments, and construct word embedding by summing over the segment embeddings

“Enriching word vectors with subword information” (Bojanowski et al 2017)

- each word is represented as a bag of (hashed) character n-grams. (e.g., spicy = (spi, pic, icy)).
- learn embeddings for the character segments, and construct word embedding by summing over the segment embeddings
- competitive with word2vec in standard tasks; better in some languages.
- produces good embeddings for unseen words
- Anticipates Transformer’s subword tokenization

Standard word embeddings (e.g. word2vec/glove) have a number of limitations:

- **polysemy**: you get one vector for multiple senses of a word
  (e.g. "**glass** of water" vs "window **glass**")

Standard word embeddings (e.g. word2vec/glove) have a number of limitations:

- **polysemy**: you get one vector for multiple senses of a word (e.g. "**glass** of water" vs "window **glass**")
- **rare words**: a word that shows up just once or twice won't be well-defined
- **n-grams**: does not produce embeddings for multi-word phrases

Scientists attending ACL work on **cutting edge** research in NLP

Petrichor: the earthy scent produce when rain falls on dry soil

Roger Federer won the first **set**^{NN} of the match

Standard word embeddings (e.g. word2vec/glove) have a number of limitations:

- **polysemy**: you get one vector for multiple senses of a word (e.g. "**glass** of water" vs "window **glass**")
- **rare words**: a word that shows up just once or twice won't be well-defined
- **n-grams**: does not produce embeddings for multi-word phrases

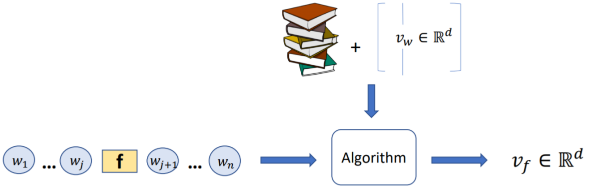
Scientists attending ACL work on **cutting edge** research in NLP

Petrichor: the earthy scent produce when rain falls on dry soil

Roger Federer won the first **set**[NN] of the match

- Goal of Khodak et al (2018): produce embeddings "a la carte" given a context:

Given: Text corpus and high quality word embeddings trained on it



Input: A feature in context(s)

Output: Good quality embedding for the feature

A la carte embeddings

- Given a target word f and its context c, define

$$v_f^{avg} = \frac{1}{|c|} \sum_{w \in c} v_w$$

  the average vector for the words in the context.
- Arora et al (2018) prove that for vectors produced by a generative language model, there exists a matrix A such that

$$v_f \approx A v_f^{avg}$$

A la carte embeddings

- Given a target word f and its context c, define

$$v_f^{avg} = \frac{1}{|c|} \sum_{w \in c} v_w$$

  the average vector for the words in the context.
- Arora et al (2018) prove that for vectors produced by a generative language model, there exists a matrix A such that

$$v_f \approx A v_f^{avg}$$

- The "induction matrix" A can be learned with a least-squares (linear regression) objective

$$A^* = \arg\min_A \sum_w |v_w - A v_w^{avg}|_2^2$$

  where w indexes over all the tokens in the corpus.
- empirically:

$$\text{cosine}(v_f, A^* v_f^{avg}) \geq 0.9$$

In-Class Quiz 5.1

eash.cc/8852

Video Presentation