Natural Language Processing for Law and Social Science

6. Word Embeddings

Outline

Embedding Layers

Word Embeddings

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 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 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 document.
- Not embeddings:
  - counts over LIWC dictionary categories.
  - sklearn CountVectorizer count vectors
- Embeddings:
  - PCA reductions of the word count vectors
  - LDA topic shares
  - compressed encodings from an autoencoder

Categorical Embeddings = dense representations of categorical variables

Say we have a binary classification problem with outcome Y:

- we have a high-dimensonal categorical variable (e.g. area of law with 1000 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-dimensonal categorical variable (e.g. area of law with 1000 categories)
- including dummy variables A for each category in your ML model is computationally expensive.

Embedding approaches:

1. 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-dimensonal categorical variable (e.g. area of law with 1000 categories)
- including dummy variables A for each category in your ML model is computationally expensive.

Embedding approaches:

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

Categorical Embeddings = dense representations of categorical variables

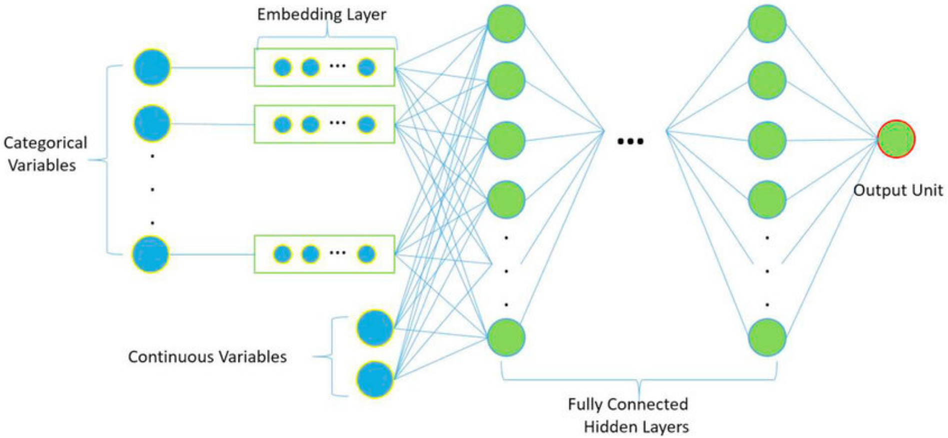
Say we have a binary classification problem with outcome Y:

- we have a high-dimensonal categorical variable (e.g. area of law with 1000 categories)
- including dummy variables A for each category in your ML model is computationally expensive.

Embedding approaches:

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

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



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.
- 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.

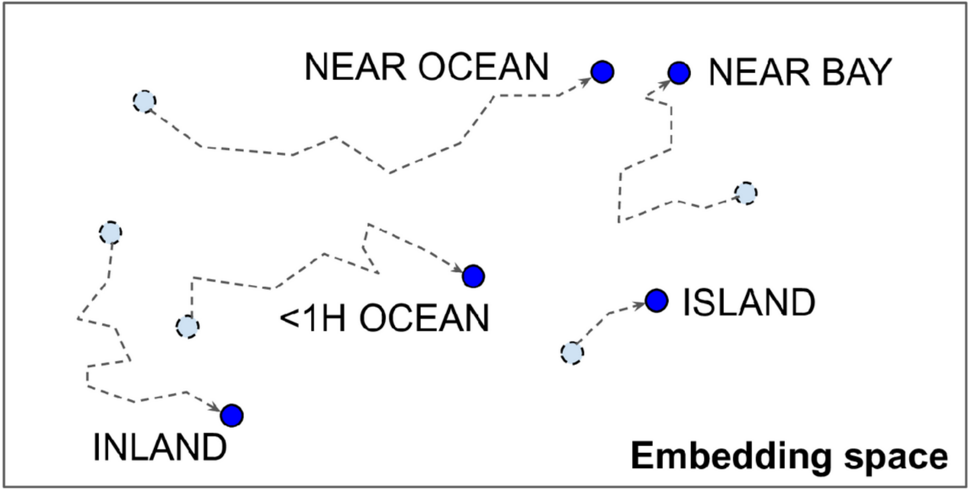


Figure 13-4. Embeddings will gradually improve during training

Embedding Layers versus Dense Layers

- An embedding layer is statistically equivalent to a fully-connected dense layer with one-hot vectors as input and linear activation.
  - embedding layers are much faster for many categories ($>\sim$50)

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; shorter documents can be padded with a null token. This requirement can be relaxed with recurrent neural networks.

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; shorter documents can be padded with a null token. This requirement can be relaxed with recurrent neural networks.
- The 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 the 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; shorter documents can be padded with a null token. This requirement can be relaxed with recurrent neural networks.
- The embedding layer replaces the list of sparse one-hot vectors with a list of n_E-dimensional ($n_E << n_w$) dense vectors

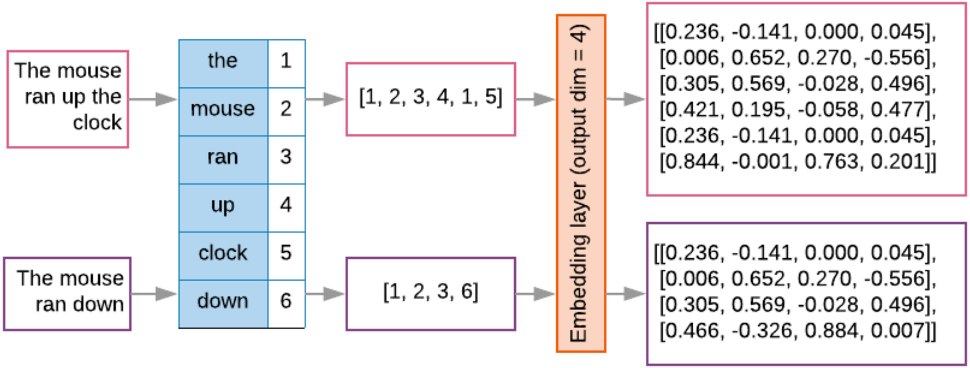
$$\boldsymbol{X} = \begin{bmatrix} x_1 & \dots & x_L \end{bmatrix}$$

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 the 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 & GloVe

- "Word embeddings" often refer to Word2Vec or GloVe – these are particular (popular) models for producing word embeddings.
  - the goal: represent the meaning of words by the neighboring words – their **contexts**.

Word2Vec & GloVe

- "Word embeddings" often refer to Word2Vec or GloVe – these are particular (popular) models for producing word embeddings.
  - the goal: represent the meaning of words by the neighboring words – their **contexts**.
  - rather than predicting some metadata (such as classifying topic labels) they predict the co-occurence of neighboring words.

Word2Vec & GloVe

- "Word embeddings" often refer to Word2Vec or GloVe – these are particular (popular) models for producing word embeddings.
  - the goal: represent the meaning of words by the neighboring words – their **contexts**.
  - rather than predicting some metadata (such as classifying topic labels) they predict the co-occurence of neighboring words.
- "You shall know a word by the company it keeps":
  - "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

A long line of NLP research aims to capture the distributional properties of words using a **word-context matrix** $\boldsymbol{M}$:

- each row w represents a **word** (e.g. "income"), each column c represents a linguistic **context** in which words can occur (e.g. "corporate ___ tax").
  - A matrix entry $\boldsymbol{M}_{[w,c]}$ quantifies the strength of association between a word and a context in a large corpus.

Words and Contexts

A long line of NLP research aims to capture the distributional properties of words using a **word-context matrix $\boldsymbol{M}$**:

- each row w represents a **word** (e.g. "income"), each column c represents a linguistic **context** in which words can occur (e.g. "corporate ___ tax").
  - A matrix entry $\boldsymbol{M}_{[w,c]}$ quantifies the strength of association between a word and a context in a large corpus.
- each word (row) $\boldsymbol{M}_{[w,:]}$ gives a distribution over contexts.
  - different definitions of contexts and different measures of association → different types of **word vectors**.
  - these vectors often have a **spatial interpretation** → 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 ($w =$ "cat", $c =$ "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 $(w = \text{"cat"}, c = \text{"tabby"})$
- Could extend this to words within a window of two:
  - add $(w = \text{"cat"}, c = \text{"the"})$
  - etc.
- Popular embeddings (word2vec and glove) generally use 5- or 10-word windows.

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.
- Context doesn't have to be single words:
  - could be the tuple of the preceding and the subsequent word.
  - Could include all words in the same sentence.

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.
- Popular embeddings (word2vec and glove) generally use 5- or 10-word windows.
- Context doesn't have to be single words:
  - could be the tuple of the preceding and the subsequent word.
  - Could include all words in the same sentence.
  - or same paragraph
  - or nouns in the same sentence
  - or syntactically connected words (from the parse tree)

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.
- Popular embeddings (word2vec and glove) generally use 5- or 10-word windows.
- Context doesn't have to be single words:
  - could be the tuple of the preceding and the subsequent word.
  - Could include all words in the same sentence.
  - or same paragraph
  - or nouns in the same sentence
  - or 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**: $f_M(w, c) = \frac{\#(w,c)}{n_D}$
  - puts high weight on common contexts shared across many words (e.g., "the cat" will be weighted higher than "tabby cat")

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**: $f_M(w,c) = \frac{\#(w,c)}{n_D}$
  - puts high weight on common contexts shared across many words (e.g., "the cat" will be weighted higher than "tabby cat")
- 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.
  - as noted in Week 2, PMI assigns high value to rare word-context pairs $\rightarrow$ impose a minimum count threshold on (w,c) pairs; below the threshold, set to zero.

$\boldsymbol{M}$ 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, the associated word vectors $\{v_1 = \boldsymbol{M}_{[w_1,:]}, v_2 = \boldsymbol{M}_{[w_2,:]}, \ldots\}$ are too high-dimensional to be useful.

$\boldsymbol{M}$ 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, the associated word vectors $\{v_1 = \boldsymbol{M}_{[w_1,:]}, v_2 = \boldsymbol{M}_{[w_2,:]}, \ldots\}$ are too high-dimensional to be useful.
- 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 $\boldsymbol{M}$.

$\boldsymbol{M}$ 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, the associated word vectors $\{v_1 = \boldsymbol{M}_{[w_1,:]}, v_2 = \boldsymbol{M}_{[w_2,:]}, \ldots\}$ are too high-dimensional to be useful.
- 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 $\boldsymbol{M}$.
  - $\tilde{\boldsymbol{M}}$ can be seen as a "smoothed" version of $\boldsymbol{M}$; "missing" values are filled in, etc.

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

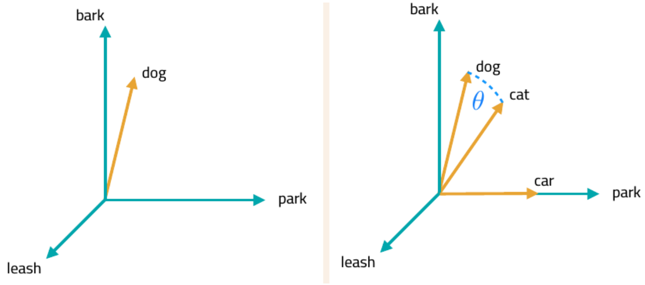
- $\boldsymbol{M}$ is $n_w \times n_c$
  - if c is drawn from from the vocabulary of a reasonably large corpus, the associated word vectors $\{v_1 = \boldsymbol{M}_{[w_1,:]}, v_2 = \boldsymbol{M}_{[w_2,:]}, \ldots\}$ are too high-dimensional to be useful.
- Going back to dimension reduction: can use 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 $\boldsymbol{M}$.
  - $\tilde{\boldsymbol{M}}$ can be seen as a "smoothed" version of $\boldsymbol{M}$; "missing" values are filled in, etc.
- $\boldsymbol{W}$ 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, the associated word vectors $\{v_1 = \boldsymbol{M}_{[w_1,:]}, v_2 = \boldsymbol{M}_{[w_2,:]}, \ldots\}$ are too high-dimensional to be useful.
- 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 $\boldsymbol{M}$.
  - $\tilde{\boldsymbol{M}}$ can be seen as a "smoothed" version of $\boldsymbol{M}$; "missing" values are filled in, etc.
- $\boldsymbol{W}$ 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 $\boldsymbol{M}$**

Word Similarity

- Once words are represented as vectors $\{v_1 = \boldsymbol{M}_{[w_1,:]},\ v_2 = \boldsymbol{M}_{[w_2,:]}, \ldots\}$, 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":



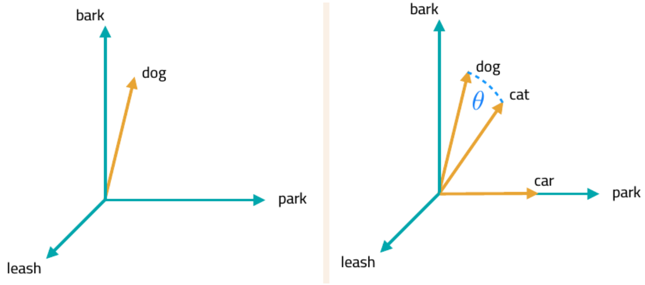
- The standard metric for comparing vectors is cosine similarity:

$$\cos\theta = \frac{v_1 \cdot v_2}{||v_1||||v_2||}$$

  - alternatives include e.g. Jaccard similarity (Goldberg 2017)

Word Similarity

- Once words are represented as vectors $\{v_1 = \boldsymbol{M}_{[w_1,:]},\ v_2 = \boldsymbol{M}_{[w_2,:]}, \ldots\}$, 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:

$$\cos\theta = \frac{v_1 \cdot v_2}{||v_1||||v_2||}$$

  - alternatives include e.g. Jaccard similarity (Goldberg 2017)
- Thanks to linearity, can compute similarities between groups of words by averaging the groups.

Word2Vec

- When people mention “word2vec”, they are usually talking about a particular word-embedding model with good performance on a range of analogy and prediction tasks.

Word2Vec

- When people mention "word2vec", they are usually talking about a particular word-embedding model with good performance on a range of analogy and prediction tasks.
- 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")

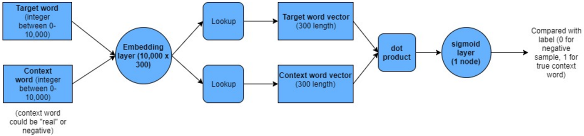
Word2Vec

- When people mention "word2vec", they are usually talking about a particular word-embedding model with good performance on a range of analogy and prediction tasks.
- 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.

Word2Vec Negative Sampling Objective

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**).



- Both words are looked up in the same embedding matrix.
- The concatened embeddings $[\boldsymbol{w}; \boldsymbol{c}]$ are input to a dense layer (no activation) then to sigmoid output:

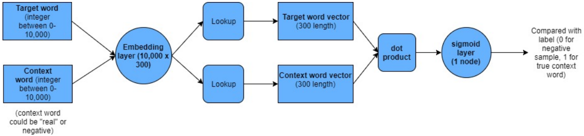
$$\hat{y}(w, c) = \text{sigmoid}(([\boldsymbol{w}; \boldsymbol{c}] \cdot \omega_0) \cdot \omega_1)$$

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

Word2Vec Negative Sampling Objective

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**).



- Both words are looked up in the same embedding matrix.
- The concatened embeddings $[\boldsymbol{w};\boldsymbol{c}]$ are input to a dense layer (no activation) then to sigmoid output:

$$\hat{y}(w,c) = \text{sigmoid}(([\boldsymbol{w};\boldsymbol{c}]\cdot\omega_0)\cdot\omega_1)$$

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

- Word2Vec minimizes the binary cross-entropy

$$L(\theta) = -\sum_{i=1}^{n_D}[y_i \log \hat{y}_i(w,c;\theta) + [1-y_i]\log(1-\hat{y}_i(w,c;\theta))]$$

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.

GloVe Embeddings

- Pennington et al (2014) (GloVe = Global Vectors) take a different approach:
  - that does not require 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) take a different approach:
  - that does not require 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.

Learn word vectors $\boldsymbol{w} = (w_1, ..., w_i, ..., w_{n_w})$, where $w_i \in (-1, 1)^{n_E}$, 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 weighting function to down-weight frequent words.

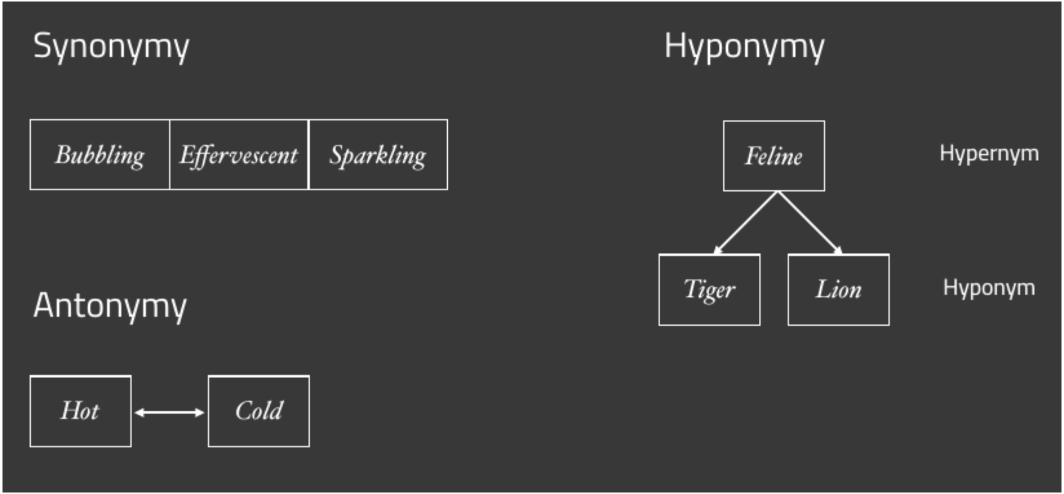
- Minimizes **squared difference** between**:**
  - **dot product of word vectors,** $w_i^T w_j$
  - **empirical co-occurrence,** $\log(C_{ij})$
    [Arora et al (2016) put the PMI here instead of co-occurence counts]
- Intuitively: words that co-occur should have high correlation (dot product)

Check for Understanding

1. What is the difference/connection between an embedding layer and a word embedding?
2. Why use PMI instead of co-occurence frequencies when constructing the word association matrix?
3. What does negative sampling mean in general, and in the case of Word2Vec?
4. What are the main differences between Word2Vec and GloVe?

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)

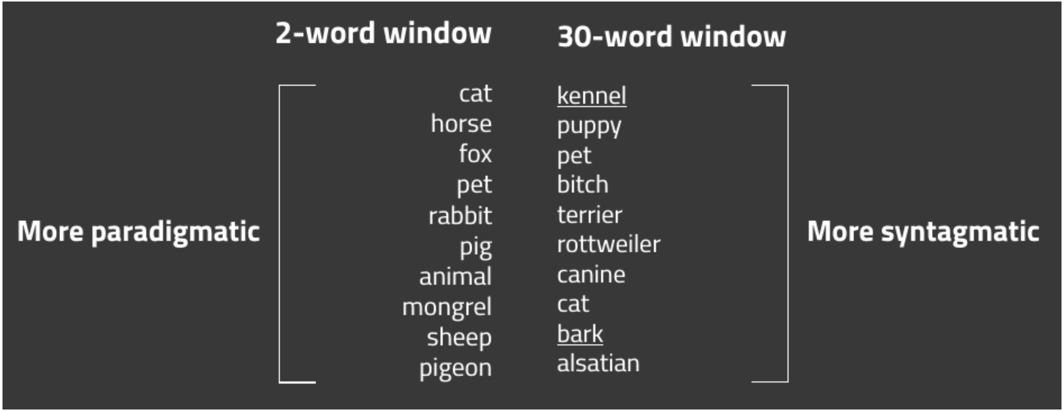
Similarity vs. Relatedness (Budansky and Hirst, 2006)

- 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.

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.

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 tells you sim(black,sheep) $>$ sim(white,sheep).

The black sheep problem

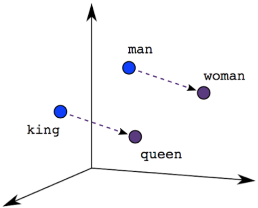
- The trivial or obvious features of a word are not mentioned in standard corpora.
- For example, although most sheep are white, you rarely see the phrase "white sheep".
  - so word2vec tells you sim(black,sheep) $>$ sim(white,sheep).
- This is really important when we will use embeddings to anayze beliefs/attitudes.
  - And I don't see a solution to it.

The black sheep problem

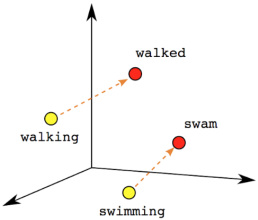
- The trivial or obvious features of a word are not mentioned in standard corpora.
- For example, although most sheep are white, you rarely see the phrase "white sheep".
  - so word2vec tells you sim(black,sheep) $>$ sim(white,sheep).
- This is really important when we will use embeddings to anayze beliefs/attitudes.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

Vector Directions $\leftrightarrow$ Meaning

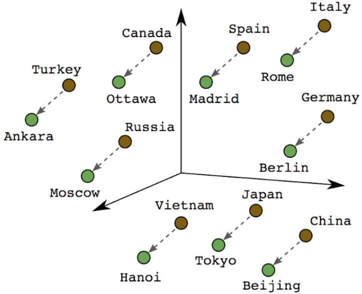
- Intriguingly, word2vec algebra can depict conceptual, analogical relationships between words:



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.

Tokenizing for Word Embeddings

- drop capitalization
- punctuation is optional
- don't drop stopwords/function-words
- 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

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

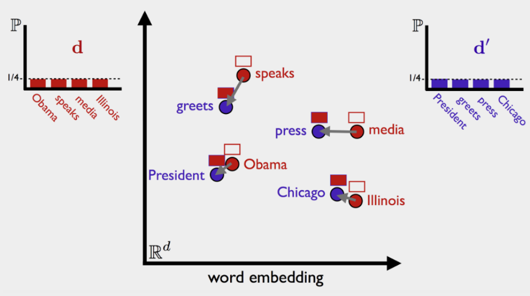
Clustered word embeddings in judicial opinions, from Ash and Nikolaus (2020)

Word Mover Distance

- TF-IDF distance treats synonyms as just as close as totally unrelated words.

Word Mover Distance

- TF-IDF distance treats synonyms as just as close as totally unrelated words.
- Word mover distance (Kusner, Sun, Kolkin, and Weinberger ICML 2015) between two texts is given by:
  - total amount of "mass" needed to move words from one side into the other
  - multiplied by the distance the words need to move
  - uses word embedding distance



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.

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

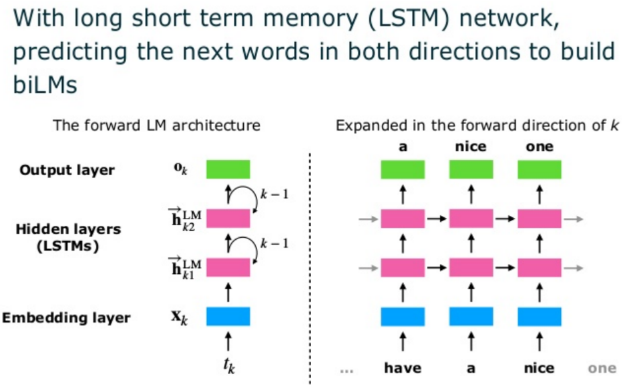
- each word is represented as a bag of (hashed) character n-grams. (e.g., spicy = (spi, pic, icy)).
- learn embeddings for the character segments, and construct word embedding by summing over the segment embeddings

“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.

ELMo (Embedings from Language Models)

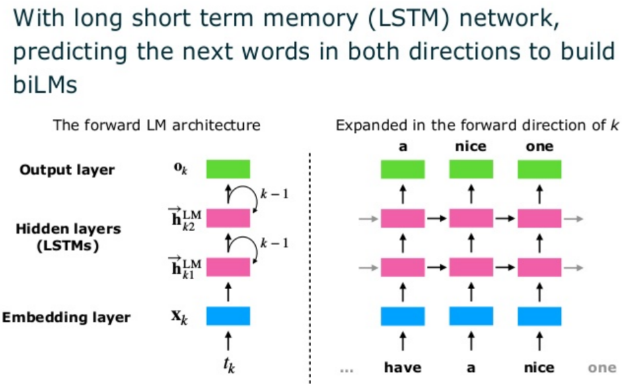
- ELMo is a context-sensitive word embedding model that uses the output of a bidrectional LSTM:



- The task:
  - predict previous and next words in a sentence using a birectional LSTM.

ELMo (Embedings from Language Models)

- ELMo is a context-sensitive word embedding model that uses the output of a bidrectional LSTM:



- The task:
  - predict previous and next words in a sentence using a birectional LSTM.
- embeddings go through two hidden layers before the softmax output:
  - first layer learns syntax
  - second layer learns semantics

GloVe mostly learns *sport*-related context

	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular play on Alusik 's grounder {...}	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent play .
	Olivia De Havilland signed to do a Broadway play for Garson {...}	{...} they were actors who had been handed fat roles in a successful play , and had talent enough to fill the roles competently , with nice understatement .

Table 4: Nearest neighbors to "play" using GloVe and the context embeddings from a biLM.

ELMo can distinguish the word sense based on the context

- Pre-trained ELMo models are available from AllenNLP (allennlp.org/elmo)

Check for Understanding

1. How would it affect my word embeddings to use co-occurence within paragraph, rather than within sentence?
2. How would it my embeddings to drop function words in a pre-processing step?
3. What is the black sheep problem in the context of word embeddings?
4. Think of a setting (and explain) where:
   - using pre-trained embeddings would not work.
   - using embeddings with subword information would help a lot
   - using elmo would work a lot better than glove.
   - you would care more about the first layer or the second layer from elmo.

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:

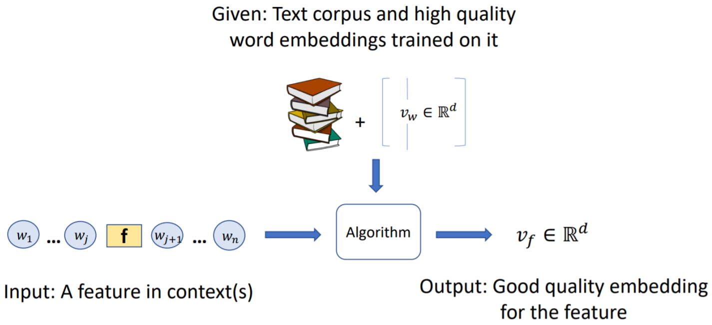
- **polysemy**: you get one vector for multiple senses of a word (e.g. "**glass** of water" vs "window **glass**")
- **rare words**: a word that shows up just once or twice won't be well-defined
- **n-grams**: does not produce embeddings for multi-word phrases

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:



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:

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