Word Embeddings

LING1340 Spring 2019 - Dan Zheng

Word representations

- We've seen things like Count Vectorizer and tf-idf
- These don't account for:
    - Word meaning
    - Word context
- How can we capture things like
    - Apple and orange are fruits
    - king - man + woman = queen

Word Embeddings (simplified)

An embedding of a word is a vector representing that word

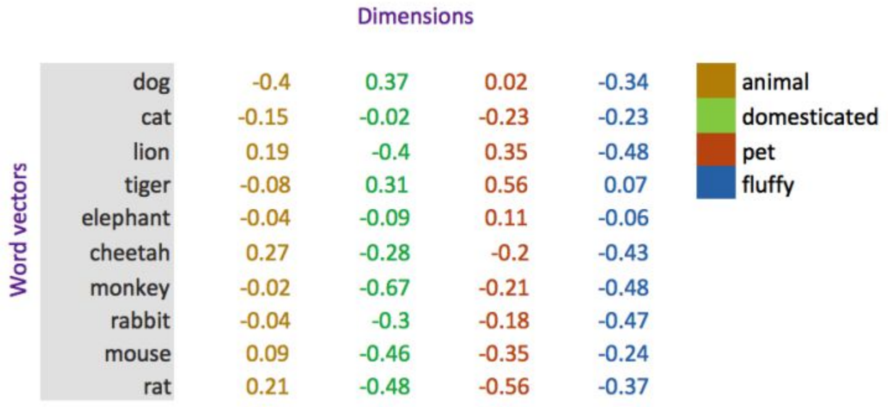
Example 4-dimensional embedding for "dog"

[-0.4, 0.37, 0.02, -0.34]

Word Embeddings (simplified)

- One way to imagine this is to imagine each dimension of the vector representing some meaning or concept related to the word

dog → [-0.4, 0.37, 0.02, -0.34]



Word vectors	Dimensions			
dog	-0.4	0.37	0.02	-0.34
cat	-0.15	-0.02	-0.23	-0.23
lion	0.19	-0.4	0.35	-0.48
tiger	-0.08	0.31	0.56	0.07
elephant	-0.04	-0.09	0.11	-0.06
cheetah	0.27	-0.28	-0.2	-0.43
monkey	-0.02	-0.67	-0.21	-0.48
rabbit	-0.04	-0.3	-0.18	-0.47
mouse	0.09	-0.46	-0.35	-0.24
rat	0.21	-0.48	-0.56	-0.37

🟫	animal
🟩	domesticated
🟥	pet
🟦	fluffy

Word Embeddings

- In reality, what each dimension represents is much harder to define
- Words with similar meanings have similar representations
- While tf-idf focuses on word relevance, word embeddings focus on word meaning
- Many different ways to produce word embeddings

Word2Vec

- First major model of word embeddings
    - Introduced by Google Research in 2013
    - Given some text, for each word, take the word and some context
        - One way to do this is the skip-gram model: Given a "center" word, predict the probabilities that it will be surrounded by some context words using a neural network
        - Another way is to train a model to "fill in" the "center" word given some context. e.g. "the cat __ on the mat" → predict high probability for "sat".
        - Embeddings are produced by an intermediate ("hidden") layer in the network

Why is this useful?

- We can do vector arithmetic with words to show things like

  king – man + woman = queen

- We can show relationships between concepts in 2D
- cool visual demo of word2vec

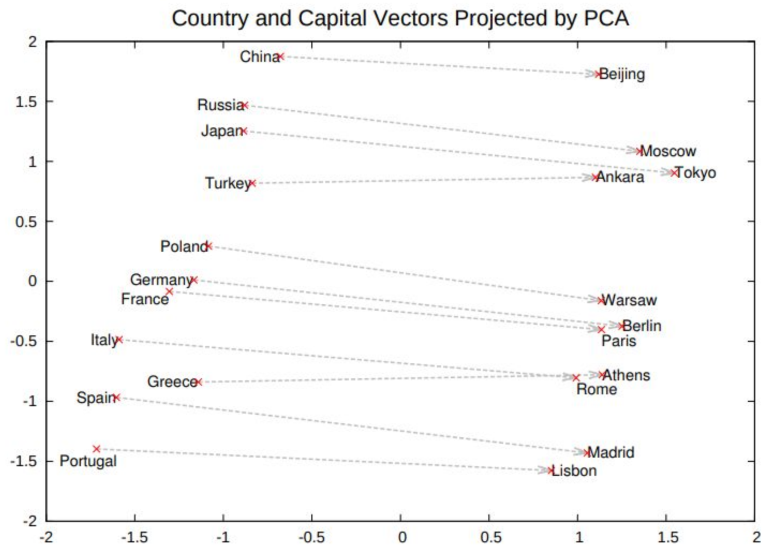
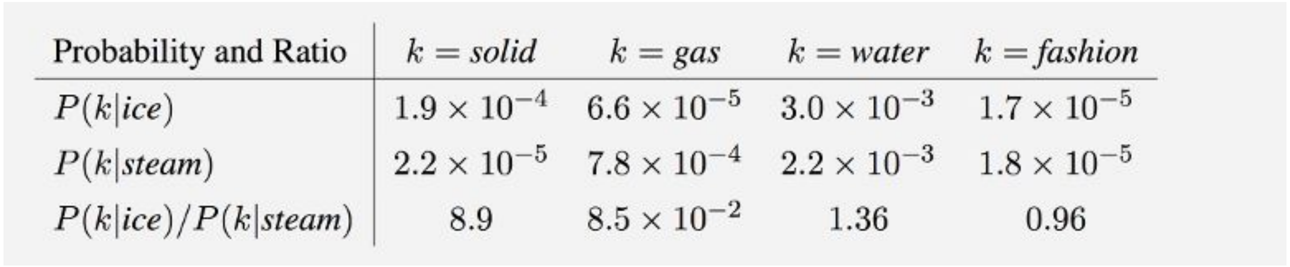


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

GloVe

Global **Ve**ctors for Word Representation

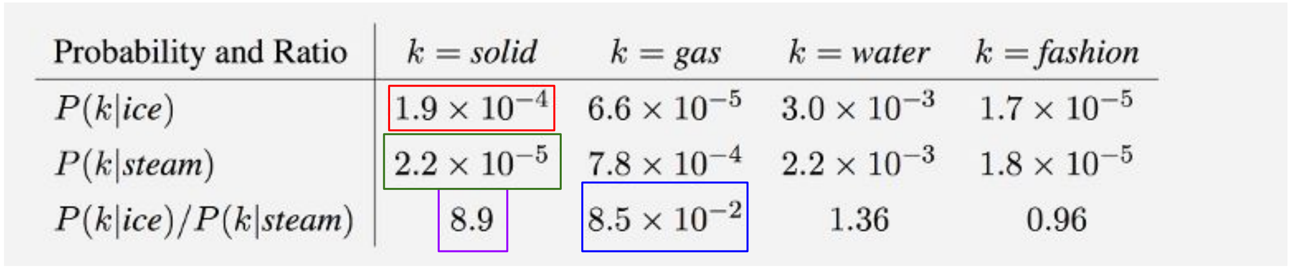
- From Stanford researchers
- Not training a model to predict stuff
- Base on co-occurrence *ratios*



Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

GloVe

- P(k | w) = probability *k* appears in context of *w*
- "solid" is closely related to "ice", but not to "steam"
- *P(solid | ice)* should be high and *P(solid | steam)* should be low
- So the **ratio** of *P(solid | ice) / P(solid | steam)* is large
- "gas" is related to "steam" but not "ice".
- **Ratio** of *P(gas | ice) / P(gas | steam)* is small.
- For a word related to *both* "ice" and "steam", like "water", we expect the ratio to be close to 1.
- Same for words unrelated to both "ice" and "steam", like "fashion".

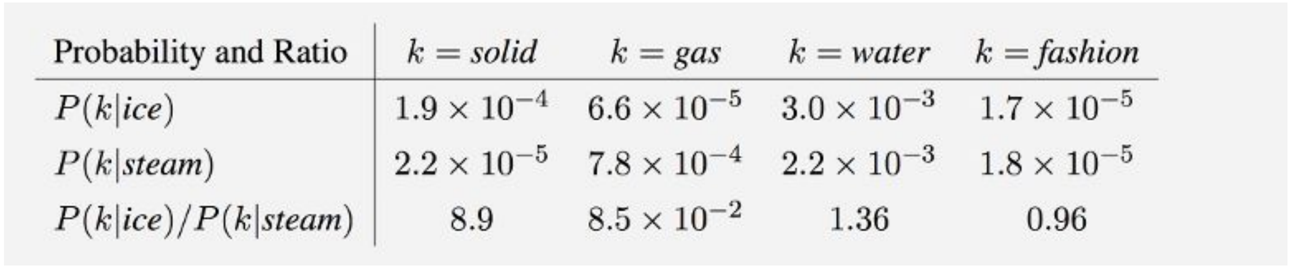


Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	$\boxed{1.9 \times 10^{-4}}$	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	$\boxed{2.2 \times 10^{-5}}$	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	$\boxed{8.9}$	$\boxed{8.5 \times 10^{-2}}$	1.36	0.96

More on GloVe

Recall from math class that log(A/B) = log(A) - log(B)

- GloVe learns word vectors such that the dot product of two vectors equals the logarithm of the words' probability of co-occurrence.
- This basically associates differences between these vectors with co-occurrence ratios (last row of the table), giving these vectors similar properties as those produced by word2vec



Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

FastText

Basically an extension of word2vec

- Uses character ngrams, e.g. "apple" -> "ap", "app", "appl", "apple", "ppl", "pple", etc
- Benefits:
  - Generate word embeddings for words it hasn't seen before, as long as it has the right character ngrams to produce it
  - Word2Vec and GloVe cannot do this
  - Supposedly better performance than other models when used for some downstream tasks
- Drawbacks
  - More hyperparameters to tweak (ngram sizes)
  - Longer to generate embeddings
  - More memory intensive

More notes on word embeddings

- Word embeddings (usually pretrained) are the standard features to start with for basically any NLP research task
  - Image captioning
  - Question Answering
  - Machine Translation
- Steps for text classification with word embeddings:
  - Tokenize text
  - Map each token to its word embedding to create 2D matrix of features
  - Add more features if you want
  - Train classifier (usually a CNN or RNN/LSTM; these are neural network models we haven't seen)
- CNNs and RNNs can handle variable-length input. Important when using word embeddings to really learn as much as possible
- For fixed-length input models that we've used, we can simply take our word embeddings and average them to create a single feature vector (losing lots of information here)

More resources

- More in-depth explanation of different types of word embeddings
- Word2Vec paper
- GloVe paper
- FastText paper
- Giant repo of word embeddings

Reviewing the main types of Machine Learning

- Supervised learning: input → expected output label
  - Classification: Logistic regression, naive bayes, svm, knn, etc etc
  - Regression: Linear regression
- Unsupervised learning: input → ???
  - Start with unlabeled data
  - Learning word embeddings
  - Dimensionality reduction (e.g. PCA)
  - **Clustering**
- Semi-supervised learning:
  - Mix of labeled and unlabeled data
  - Usually end up using some combination of supervised + unsupervised learning
  - Complicated

K-means Clustering

- Want to produce **K** clusters/groups from your data
- Steps:
  - Initialize **K** "centroids" (centers of clusters) randomly.
  - For each data point, calculate distance to each centroid and assign it to the closest one
  - Recalculate the centroids to be the averages of all the points assigned to it
  - Repeat until centroids aren't changing much anymore
- Visualization of k-means
- Clustering word embeddings demo

Why use clustering?

- Find structure from unstructured data
- Poke around, find some new insights from clusters
- Cluster customer data
  - Help with ad targeting?
- Find/remove outliers
- Visualization
- https://en.wikipedia.org/wiki/Cluster_analysis#Applications

Try out this stuff yourself!

- On CRC, copy my notebook to your own directory and run it in CRC Hub
  - **cp /ihome/rhwa/daz53/ling1340/embeddings.ipynb .**
  - Open in Jupyter Hub (http://hub.crc.pitt.edu)