Embeddings

Stephen Scott

(Adapted from Haluk Dogan)

sscott2@unl.edu

Introduction

- To apply recurrent and transformer architectures to text (e.g., NLM), need numeric representation of words
  - The "Embedding lookup" block
  - Want words with similar meanings to cluster together
- Where does the embedding come from?
  - Could train it along with the rest of the network
  - Or, could use "off-the-shelf" embedding
    - E.g., word2vec or GloVe
- Embeddings not limited to words: E.g., biological sequences, graphs, ...
  - Graphs: node2vec
- The xxxx2vec approach focuses on training embeddings based on **context** of what words often appear near it
  - Different "context" than, e.g., BERT, which has separate embedded vector for each usage

Outline

- Vector semantics
- Representing words as vectors
- word2vec
  - Architectures
  - Training
  - Semantics of embedding
- node2vec
- Summary

Vector Semantics

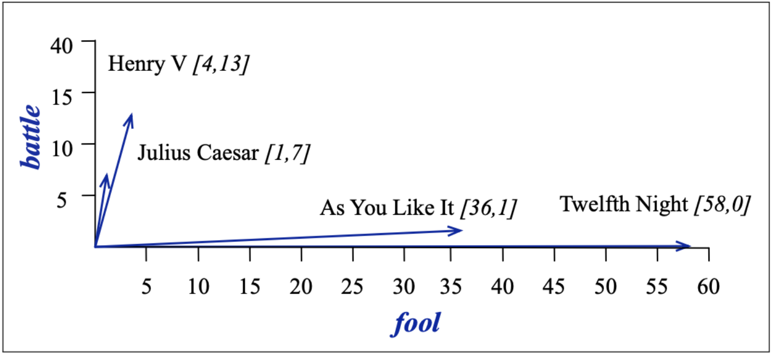
- Want to numerically represent words such that words sharing lexical semantics tend to cluster together
  - A trained model that predicts positive when the word "vanish" appears ought to respond similarly to the word "disappear"
- Will define representation that groups words together when they appear in similar **contexts**

Two-dimensional (t-SNE) projection of 60-dimensional embedding

- Let D be set of documents based on vocabulary V
- A **term-document matrix** is a $|V| \times |D|$ matrix where column d is document d's bag-of-words representation (word counts)

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Two-dimensional projection



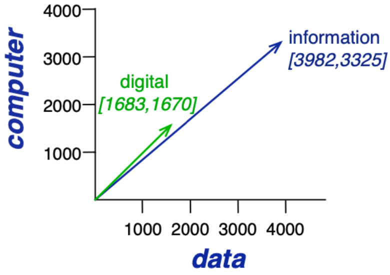
Comedies cluster, as do non-comedies

- Can use the rows of the T-D matrix to represent words
- Context for a word is the set of documents that a word appears in, and how often
- Expect similar words to have similar representations
- Can also use a $|V| \times |V|$ **word-word** (or **term-context**) matrix
  - Cell (i, j) holds the number of times words i and j appear in the same **context** (same document, same window, etc.)

Words and Vectors

Representing Words as Vectors

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	
strawberry	0	...	0	0	1	60	19	
digital	0	...	1670	1683	85	5	4	
information	0	...	3325	3982	378	5	13	

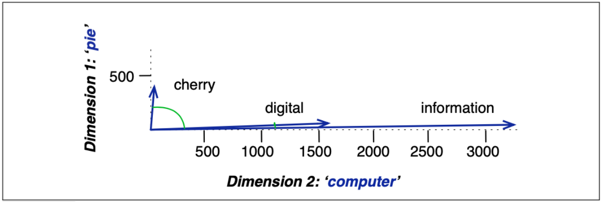


Cosine Similarity

- Can compute similarity of words (or documents)
- Common approach: The **cosine** of the angle betwen vectors v and w:

$$\frac{v \cdot w}{\|v\|\|w\|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

- I.e., **normalize** each vector by dividing by its length, then take the dot product of unit-length vectors
- Similarity $= 1$ if vectors identical, 0 if orthogonal (never < 0 since all vectors nonnegative)



Term Frequency-Inverse Document Frequency

- Want to be careful about how we define context
- Common words such as “the” appear frequently in any corpus
- Can artificially inflate similarity measures
- **TF-IDF** weights terms not only by frequency, but also **reduces** a weight if a term appears in many documents

Term Frequency-Inverse Document Frequency

Definition

Embeddings
Stephen Scott
Introduction
Vector semantics
Words and Vectors
Term-Document Matrix
Words as Vectors
Cosine Similarity
TF-IDF
word2vec
node2vec
Summary

- **Term frequency** counts frequency of term t in document d, with a 1 psuedocount to avoid log of zero:

$$\text{tf}_{t,d} = \log_{10}(\text{count}(t, d) + 1)$$

- Log is used since, e.g., 100 occurences is not much different than 200
- **Inverse document frequency** decreases as the number of documents containing t increases:

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

N = total number of documents, df_t = number of documents containing t

- Weight of t, d is then $w_{t,d} = (\text{tf}_{t,d})(\text{idf}_t)$

- Now, when we compute word-word matrix, weight counts by tf-idf
- Then extract vector for term t as usual
- Can then normalize and compute cosine similarity

Word2vec (Mikolov et al.)

Dense Vector Representations

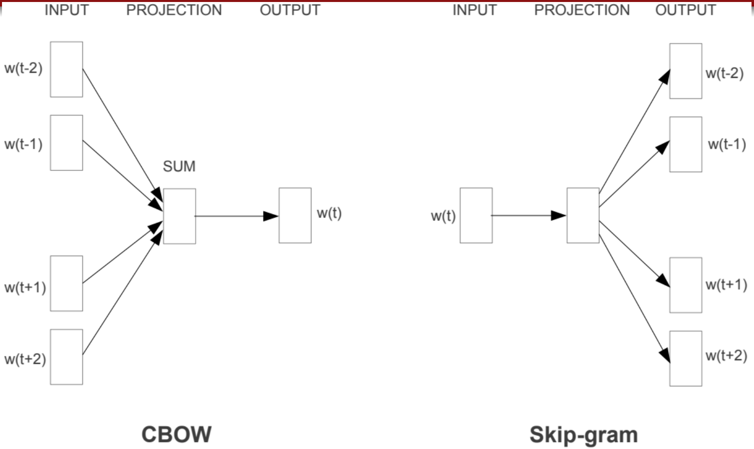
- Vectors based on tf-idf are typically long ($|V|$ order of 10^4–10^5) and sparse (mostly 0 entries)
- Alternative: Vectors that are short (say, 100 dimensions) and dense
  - Fewer model parameters to learn
  - Capture synonymy: In a sparse representation, "car" and "automobile" are distinct dimensions, so vectors might be distant
  - Work better in practice
- Examples
  - Word2vec (`https://code.google.com/archive/p/word2vec/`)
  - Fasttext (`http://www.fasttext.cc/`)
  - GloVe (`http://nlp.stanford.edu/projects/glove/`)

Word2vec (Mikolov et al.)

Intuition

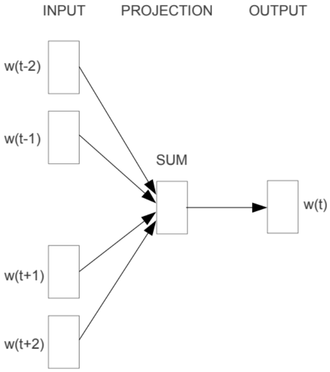
- Given a word w
  - Rather than **count** how often w appears near word "apricot"
  - **Predict** how likely w is to appear near "apricot"
- **Equivalent view:** Train as a variation of autoencoding
  - But rather than mapping a word to itself, learn to map between a word and its **context**
  - Context-to-word: **Continuous bag-of-words** (CBOW)
  - Word-to-context: **Skip-gram**
- Can train this with a neural network
  - Can use supervised training, since any word appearing near "apricot" in corpus is a positive example
  - Will also use **negative sampling** in training
- Model's weights can be used as word embeddings

Word2vec (Mikolov et al.)
Architectures



- CBOW: Predict current word $w(t)$ based on context
- Skip-gram: Predict context based on $w(t)$
- One-hot input, hidden linear activation, softmax output

Word2vec (Mikolov et al.)
CBOW



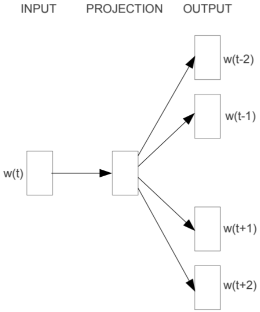
CBOW

- N = vocabulary size, d = embedding dimension
- $N \times d$ matrix W is shared weights from input to hidden
- $d \times N$ matrix W' is weights from hidden to output
- When one-hot context vectors $\boldsymbol{x}_{t-2}, \boldsymbol{x}_{t-1}, \ldots, \boldsymbol{x}_{t+2}$ input, corresponding rows from W are summed to $\hat{\boldsymbol{v}}$
- Then get **score vector** $\boldsymbol{v}'$ and softmax it
- Train with cross-entropy

- Use ith column of W' as embedding

Word2vec (Mikolov et al.)
Skip-gram

- Symmetric to CBOW: use ith row of W as embedding
- Goal is to maximize $P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} \mid w_t)$
- Same as minimizing $-\log P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} \mid w_t)$
- Assume words are independent given w_t: $P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} \mid w_t) = \prod_{j \in \{-2,-1,1,2\}} P(w_{t+j} \mid w_t)$



Skip-gram

Word2vec (Mikolov et al.)
Skip-Gram with Negative Sampling (SGNS)

Given target word t

1. Treat t and a nearby **context word** c as a positive example, e.g.,
   ... lemon, a tablespoon of apricot jam, a pinch ...
   c_1 c_2 t c_3 c_4
2. Randomly sample other words for negative examples
   - E.g., $t =$ “apricot”, $c =$ “aardvark”
   - Probability of selecting word w as a negative is

   $$\frac{count(w)^{\alpha}}{\sum_{w}' count(w')^{\alpha}}$$

   for parameter α (e.g., 0.75)
3. Train with logistic output activation, starting with random weights
4. Use final weights as embeddings

- Given t and c, goal is to predict probability that they are near each other ("$+$" label)
- Represent t and c as their embedded representations and compute similarity via dot product: $t \cdot c$
- Use logistic function to map to a probability in interval $[0, 1]$:

$$P(+ \mid t, c) = \frac{1}{1 + \exp(-t \cdot c)}$$

$$P(- \mid t, c) = 1 - P(+ \mid t, c) = \frac{\exp(-t \cdot c)}{1 + \exp(-t \cdot c)}$$

- For k context words, log of total probability is

$$\log P(+ \mid t, c_1, \ldots, c_k) = \sum_{i=1}^{k} \log \frac{1}{1 + \exp(-t \cdot c_i)}$$

- Given positive pairs $(t, c) \in S_+$ and k negative pairs $(t, n) \in S_-$, want to maximize

$$\sum_{(t,c)\in S_+} \log P(+ \mid t, c) + \sum_{(t,n)\in S_-} \log P(- \mid t, n)$$

$$= \sum_{(t,c)\in S_+} \log \sigma(c \cdot t) + \sum_{(t,n)\in S_-} \log \sigma(-n \cdot t)$$

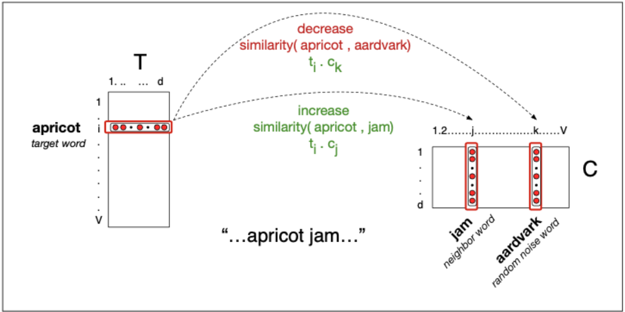
$$= \sum_{(t,c)\in S_+} \log \frac{1}{1 + \exp(-c \cdot t)} + \sum_{(t,n)\in S_-} \log \frac{1}{1 + \exp(n \cdot t)}$$

- I.e., optimizing involves finding an embedding maximizing $c \cdot t$ and minimizing $n \cdot t$ for all t

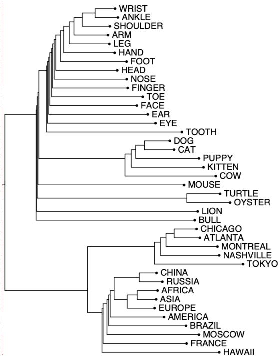
Word2vec (Mikolov et al.)

Skip-Gram with Negative Sampling (SGNS)

- Because word w can be used both as a target word and a context word, training results in two matrices: **target embedding** T and **context embedding** C
- T and C form parameters θ
  - Row w of T is d-dimensional representation of word w
  - Column w of C is d-dim representation of word w
- Often will just use T as final embedding, but could concatenate w's vectors into a $2d$-dimensional vector

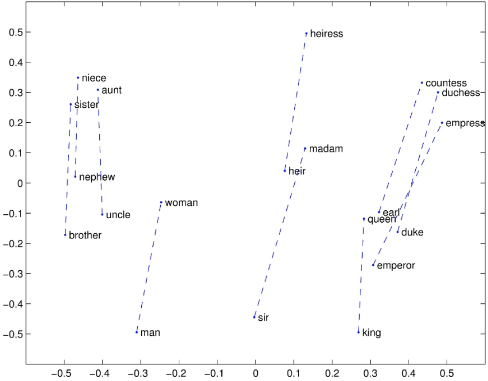


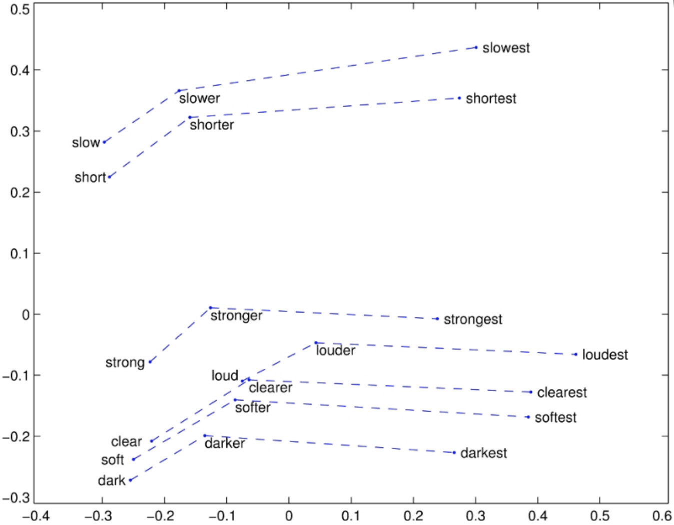
- High-dim representations make visualizing embeddings difficult
- One visualization technique: **Hierarchically cluster** words into a **dendogram**
- Another common approach: Project to 2-dimensional space using **t-SNE**



Word2vec (Mikolov et al.)
Semantics

- Embeddings can allow **semantic arithmetic**
- E.g., take embedded representation of “king”, subtract “man” and add “woman”, and result is near that of embedded representation of “queen”
- (“queen” minus “king” $\approx$ “woman” minus “man”)

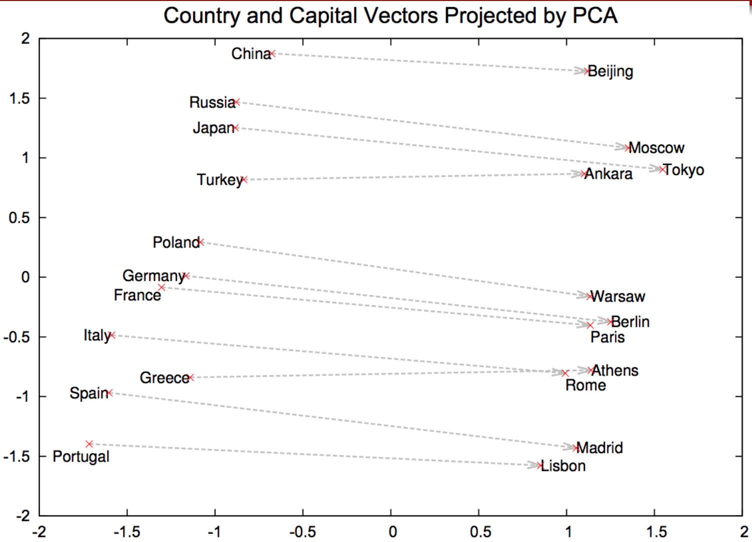




Adjective → comparative → superlative

Word2vec (Mikolov et al.)

Semantics



Distances between countries and capitals similar

- Semantics can also reveal biases in corpus
  - E.g., “man” minus “computer programmer” plus “woman” yields “homemaker”
  - Racial biases also revealed
- Application of word2vec to **materials science paper abstracts**
  - Embedded vectors of elements appearing in abstracts clustered according to the periodic table
  - Was able to predict properties of materials before discovered by scientists

Node2vec (Grover and Leskovec, 2016)

- Word2vec's approach generalizes beyond text
- All we need to do is represent the context of an instance to embed together instances with similar contexts
  - E.g., biological sequences, nodes in a graph
- Node2vec defines its context for a node based on its local neighborhood, role in the graph, etc.

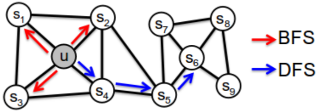
Node2vec (Grover and Leskovec, 2016)
Notation

- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- $\mathcal{A}$ is a $|\mathcal{V}| \times |\mathcal{V}|$ adjacency matrix
- $f : \mathcal{V} \rightarrow \mathbb{R}^d$ is a mapping function from individual nodes to feature representations
  - $|\mathcal{V}| \times d$ matrix
- $N_S(u) \subset \mathcal{V}$ denotes a neighborhood of node u generated through a **neighborhood sampling strategy** S
- **Objective:** Preserve local neighborhoods of nodes

Node2vec (Grover and Leskovec, 2016)

Organization of nodes is based on:

- **Homophily:** Nodes that are highly interconnected and cluster together should embed near each other



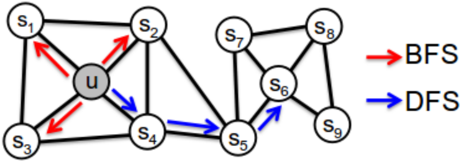
- **Structural roles:** Nodes with similar roles in the graph (e.g., hubs) should embed near each other
- u and s_1 belong to the same community of nodes
- u and s_6 in two distinct communities share same structural role of a hub node

Goal

- Embed nodes from the same network community closely together
- Nodes that share similar roles have similar embeddings

Key Contribution: Defining a flexible notion of a node's network neighborhood.

1. **BFS:** role of the vertex
   - Far apart from each other but share similar kind of vertices
2. **DFS:** community
   - Reachability/closeness of the two nodes
   - My friend's friend's friend has a higher chance to belong to the same community as me



Objective function

$$\max_f \sum_{u \in \mathcal{V}} \log P\left(N_S(u) \mid f(u)\right)$$

Assumptions:

- Conditional independence:
  $$P\left(N_S(u) \mid f(u)\right) = \prod_{n_i \in N_S(u)} P(n_i \mid f(u))$$
- Symmetry in feature space:
  $$P\left(n_i \mid f(u)\right) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in \mathcal{V}} \exp(f(v) \cdot f(u))}$$

Objective function simplifies to:

$$\max_f \sum_{u \in \mathcal{V}} \left[-\log Z_u + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u) \right]$$

Given a source node u, we simulate a random walk of fixed length ℓ:

$$P\left(c_i = x \mid c_{i-1} = v\right) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

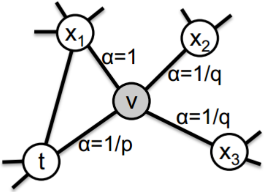
- $c_0 = u$
- π_{vx} is the unnormalized transition probability
- Z is the normalization constant.
- 2^{nd} order Markovian

Node2vec (Grover and Leskovec, 2016)

Neighborhood Sampling

Search bias α: $\pi_{vx} = \alpha_{pq}(t,x) w_{vx}$ where

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$



Return parameter p:

- Controls the likelihood of immediately revisiting a node in the walk
- If $p > \max(q, 1)$
  - less likely to sample an already visited node
  - avoids 2-hop redundancy in sampling
- If $p < \min(q, 1)$
  - backtrack a step
  - keep the walk local

In-out parameter q:

- If $q > 1$ inward exploration
  - Local view
  - BFS behavior
- If $q < 1$ outward exploration
  - Global view
  - DFS behavior

Node2vec (Grover and Leskovec, 2016)
Algorithm

Algorithm 1 The *node2vec* algorithm.

```
LearnFeatures (Graph G = (V, E, W), Dimensions d, Walks per
    node r, Walk length l, Context size k, Return p, In-out q)
    π = PreprocessModifiedWeights(G, p, q)
    G' = (V, E, π)
    Initialize walks to Empty
    for iter = 1 to r do
        for all nodes u ∈ V do
            walk = node2vecWalk(G', u, l)
            Append walk to walks
    f = StochasticGradientDescent(k, d, walks)
    return f

node2vecWalk (Graph G' = (V, E, π), Start node u, Length l)
    Inititalize walk to [u]
    for walk_iter = 1 to l do
        curr = walk[−1]
        V_curr = GetNeighbors(curr, G')
        s = AliasSample(V_curr, π)
        Append s to walk
    return walk
```

- Implicit bias due to choice of the start node u
    - Simulating r random walks of fixed length ℓ starting from every node

Phases:

1. Preprocessing to compute transition probabilities
2. Random walks
3. Optimization using SGD

Each phase is parallelizable and executed asynchronously

Summary

- Numeric representations of words called **embeddings** are an important part of NLP
- One type of **sparse** embedding is based on **co-occurence** counts, often weighted by **tf-idf**
- A popular **dense** embedding is **word2vec**, which is built via training a logistic classifier
- Other popular embeddings are **GloVe** and **fasttext**
- **Dot products** of embedded vectors (which equals cosine when vectors are unit length) are good ways of capturing word similarity
- **Semantic arithmetic** is possible with some embeddings, including adding and subtracting vectors
- The word2vec approach is applicable to other types of data with contexts, such as **nodes in a graph**