

Symbolic (one-hot) representations

cat = (0, ..., 1, 0, 0, ..., 0)
dog = (0, ..., 0, 1, 0, ..., 0)
book = (0, ..., 0, 0, 1, ..., 0)
...



Problems with one-hot representations

- No notion of similarity
- Large and sparse vectors

Representations of linguistic units

- The success of NLP methods depend on how we represent the objects of interest, such as
 - words, morphemes
 - sentences, phrases
 - letters, phonemes
 - documents
 - speakers, authors
 - ...
- The way we represent these objects interacts with the ML methods used for the task
- We will mostly talk about word representations
 - They are also applicable any of the above and more

More useful vector representations: embeddings

- The idea is to represent similar words with similar vectors

cat = (0, 3, 1, ..., 4)
dog = (0, 3, 0, ..., 3)
book = (4, 1, 4, ..., 5)
...



- The similarity between the vectors may represent similarities based on
 - syntactic
 - semantic
 - topical
 - ... features useful in a particular task

Where do the vector representations come from?

- The vectors are (almost certainly) learned from data
- Typically using an unsupervised (or self-supervised) method
- The idea goes back to,
 - You shall know a word by the company it keeps. —Firth (1957)*
- In practice, we make use of the contexts (company) of the words to determine their representations
- Words that appear in similar contexts are mapped to similar representations

How do we learn word vectors?

	c_1	c_2	c_3	...	c_m
w_1	0	3	1	...	4
w_2	0	3	0	...	3
w_3	4	1	4	...	5

- Counting/weighting based on context already allows us to learn similarities between words
- But these vectors are large and sparse
- Dense vectors have a number of desirable properties
 - More efficient to process
 - Removed redundancy also means better generalizations
 - Less sensitive to noise

How to calculate dense word vectors?

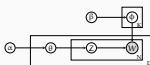
(1) count, factorize, truncate

$$\begin{matrix} w_1 \\ w_2 \\ w_3 \end{matrix} \begin{bmatrix} c_1 & c_2 & c_3 & \dots & c_m \\ 0 & 3 & 1 & \dots & 4 \\ 0 & 3 & 0 & \dots & 3 \\ 4 & 1 & 4 & \dots & 5 \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} =$$

$$\begin{matrix} w_1 \\ w_2 \\ w_3 \end{matrix} \begin{bmatrix} z_1 & z_2 & z_3 & \dots & z_n \\ 1 & 5 & 9 & \dots & 0 \\ 1 & 4 & 1 & \dots & 3 \\ 9 & 1 & 1 & \dots & 5 \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} \theta_1 & \dots & \theta_n \\ 0 & 3 & 1 & \dots & 4 \\ 0 & 3 & 0 & \dots & 3 \\ 0 & \dots & \theta_m \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} c_1 & c_2 & c_3 & \dots & c_m \\ 0 & 3 & 1 & \dots & 4 \\ 0 & 3 & 0 & \dots & 3 \\ 9 & 1 & 4 & \dots & 5 \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix}$$

How to calculate word vectors?

(2) latent variable models (e.g., LDA)

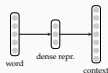


- Assume that the each 'document' is generated based on a mixture of latent variables
- Learn the probability distributions
- Typically used for *topic modeling* (θ)
- Can model words too (ϕ)

How to calculate word vectors?

(3) predict the context from the word, or word from the context

- The task is predicting
 - the context of the word from the word itself
 - or the word from its context
- Task itself is not (necessarily) interesting
- We are interested in the hidden layer representations learned



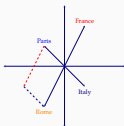
Applications of word vectors: similarity and analogy

- It was shown that the vector space models outperform humans in
 - TOEFL synonym questions
 - Receptors for the sense of smell are located at the **top** of the nasal cavity.
 - A. upper end B. inner edge C. mouth D. division
 - SAT analogy questions
 - Pithy** is to **significance** as **_____** is to **_____**
 - A. redundant : discussion
 - B. austere : landscape
 - C. opulent : wealth
 - D. oblique : familiarity
 - E. banal : originality

Vector arithmetic with embeddings

Word vectors map some syntactic/semantic relations to vector operations

- $\text{Paris} - \text{France} + \text{Italy} = \text{Rome}$
- king - man + woman = queen
- ducks - duck + mouse = mice



Singular Value Decomposition (SVD)

a very short introduction

- Singular value decomposition is a well-known method in linear algebra
- An $n \times m$ (n terms m documents) term-document matrix X can be decomposed as

$$X = U \Sigma V^T$$

U is a $n \times r$ unitary matrix, where r is the rank of X ($r \leq \min(n, m)$). Columns of U are the eigenvectors of XX^T
 Σ is a $r \times r$ diagonal matrix of singular values (square root of eigenvalues of XX^T and $X^T X$)
 V^T is a $r \times m$ unitary matrix. Columns of V are the eigenvectors of $X^T X$

Truncated SVD

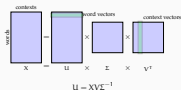
$$X = U \Sigma V^T$$

- Using eigenvectors (from U and V) that correspond to k largest singular values ($k < r$), allows reducing dimensionality of the data with minimum loss
- The approximation,

$$\hat{X} = U_k \Sigma_k V_k^T$$

- results in the best approximation of X , such that $\| \hat{X} - X \|_F$ is minimum
- Note that r and n may easily be millions (of words or contexts), while we choose k much smaller (a few hundreds)

More notes on word vectors from SVD



- Each component of a 'reduced' word vector is a weighted sum of the original word vector
- SVD removes correlations, resulting in less redundancy

SVD based vectors: practical concerns

- In practice, instead of raw counts of terms within contexts, the term-document matrices typically contain
 - pointwise mutual information
 - tf-idf
- If the aim is finding latent semantic/topical dimensions, frequent/syntactic words (*stopwords*) are often removed
- Depending on the measure used, it may also be important to normalize for the document length

Predictive models



- For this lecture, we are interested in *static* embeddings. We will discuss *contextual* representations later

word2vec

- word2vec** is a popular algorithm and open source application for training word vectors
- It has two modes of operation
 - CBOW**: continuous bag of words: predict the word using a window around the word
 - Skip-gram**: does the reverse, it predicts the words in the context of the target word using the target word as the predictor

word2vec

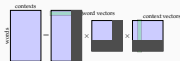
a bit more in detail

- For each word w , the algorithm learns two sets of embeddings
 - v_w for words
 - c_w for contexts
- Objective of the learning is to maximize (skip-gram)

$$P(c | w) = \frac{e^{v_w \cdot c_w}}{\sum_{c' \in \mathcal{C}} e^{v_w \cdot c'}}$$

- Note that the above is simply *softmax* – the learning method is equivalent to logistic regression, but we have additional parameters (c) to estimate
- Now, we can use *gradient-based* approaches to find word and context vectors that maximize this objective

Truncated SVD: with a picture



- Step 1 Get word-context associations
- Step 2 Decompose
- Step 3 Truncate

SVD: LSI/LSA

SVD applied to term-document matrices are called

- Latent semantic analysis* (LSA) if the aim is constructing *term* vectors
 - Semantically similar words are closer to each other in the vector space
- Latent semantic indexing* (LSI) if the aim is constructing *document* vectors
 - Topically related documents are closer to each other in the vector space

Matrix factorization: summary

- Matrix factorization methods were around for a long time: they are well studied and well known
- These methods are effective: guaranteed optimality / convergence
- The methods do not scale well for large data sets
- The mappings are linear

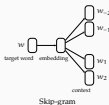
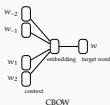
Predictive models

common approaches

- Instead of dimensionality reduction through SVD, we try to predict
 - either the target word from the context
 - or the context given the target word
- In practice 'shallow' methods are shown to be effective
- Typically,
 - We assign each word to a fixed-size random vector
 - We use a standard ML model and try to reduce the prediction error with a method like gradient descent
 - During learning, the algorithm optimizes the vectors as well as the model parameters

word2vec

CBOW and skip-gram modes – conceptually



Issues with softmax

$$P(c | w) = \frac{e^{v_w \cdot c_w}}{\sum_{c' \in \mathcal{C}} e^{v_w \cdot c'}}$$

- A particular problem with models with a softmax output is high computational cost:
 - For each instance in the training data denominator has to be calculated over the whole vocabulary (can easily be millions)
- Two workarounds exist:
 - Negative sampling: a limited number of negative examples (sampled from the corpus) are used to calculate the denominator
 - Hierarchical softmax: turn output layer to a binary tree, where probability of a word equals to the probability of the path followed to find the word
- Both methods are applicable during training, during prediction, we still need to compute the full softmax

word2vec: some notes

- Note that word2vec is not 'deep'
- word2vec performs well, and it is much faster than earlier (more complex) ANN architectures developed for this task
- The resulting vectors used by many (deep) ANN models, but they can also be used by other 'traditional' methods
- word2vec treats the context as a BoW, hence vectors capture (mainly) semantic relationships
- We need to keep the vocabulary (relatively) small, the method does not help with out-of-vocabulary words

Using vector representations

- Dense vector representations are useful for many ML methods
- They are particularly suitable as input to neural network models
- The embeddings alone can be used in many applications that require measuring similarities between words
- Dense vector representations are not specific to words, they can be obtained and used for any (linguistic) object of interest

Evaluating vector representations

- Like other unsupervised methods, there are no 'correct' labels
- Evaluation can be
 - Intrinsic: based on success on finding analogy/synonymy
 - Extrinsic: based on whether they improve a particular task (e.g., parsing, sentiment analysis)
 - Correlation with human judgments

Summary

- (Dense) vector representations of linguistic units allow calculating similarity/difference between the units
- General purpose embeddings can be 'trained' using counting (SVD), or predicting (word2vec, GloVe)
- They are particularly suitable for ANNs as low-dimensional inputs
- Although these general purpose embeddings are useful,
 - they typically do not distinguish some important properties (e.g., they assign similar vectors to antonyms)
 - they do not handle polysemy, meaning in context
- Embeddings can also be trained on a particular task
- Also works for other linguistic objects (e.g., letters, sentences)
- Reading: Jurafsky and Martin (2025, Chapter 6)

Next:

- Gradient descent, Reading: Jurafsky and Martin (2025, Section 5.6)

Other predictive methods for building vector representations

- There are a few other popular methods for building 'general purpose' vector representations
 - GloVe tries to combine local information (similar to word2vec) with global information (similar to SVD)
 - FastText makes use of characters (n-grams) within the word as well as their context
- One can also train embeddings for a particular task/application, by plugging an 'embedding layer' to any neural network

Context matters

In SVD (and other) vector representations, the choice of context matters

- Larger contexts tend to find semantic/topical relationships
- Smaller (also order-sensitive) contexts tend to find syntactic generalizations

Differences of the methods

...or the lack thereof

- It is often claimed, after excitement created by word2vec, that prediction-based models work better
- Careful analyses suggest, however, that word2vec can be seen as an approximation to a special case of SVD
- Performance differences seem to boil down to how well the hyperparameters are optimized
- In practice, the computational requirements are probably the biggest difference

Some sources of information

- Jurafsky and Martin (Chapter 6, 2025)

 Jurafsky, Daniel and James H. Martin (2025). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released January 12, 2025. url: <https://web.stanford.edu/~jurafsky/slp3/>.