CIS5300 - Speech and Language Processing - Chapter 6 Notes

Jonathon Delemos - Chris Callison Burch

July 13, 2025

0.1 Abstract - Vector Semantics and Embeddings

In this chapter, we will introduce **vector semantics**, which instantiates this linguistic hypothesis by learning representations of the meaning of the words, called **embeddings**, directly from their distribution in texts. These representations are the first examples of **representation learning**, automatically learning useful representations of the input text.

0.2 6.1 - Lexical Semnatics

Let's start by looking at how one word might be defined in a dictionary. *Mouse* might be defined as:

- 1. Any of numerous small rodents
- 2. a hand-operated device that controls a cursor

Here, the form mouse is the **lemma**, also called the **citation form**. Similarly, sing is the lemma for sing, sang, and sung. The different definitions for each word can be called **word sense disambiguation**. Similarity is very useful in large semantic tasks. For example, vanish and disappear might have s similarity score of 9.8/10. **Semantic fields** are a common way to relate words to each other: *CPU, disk, memory, I/O*

0.3 6.2 - Vector Semantics

Vector Semantics is the standard wy to represent word meaning in NLP, helping us model many of the aspects of word meaning. For example, we might measure the affective meaning of a word by using *three metrics.* This would now be a vector with three variables. The big idea behind vecor semantics is to represent a word as a point in a multidimensional space. We can infer some words just by looking at the words that might be adjacent.

Information Retrieval is the task of finding document d and from the D documents in some colletion that best matches the query q. For IR, we'll therefore also represent a query by a vector, also of length $\mid V \mid$.

Instead of looking at the document as a vector, we can also look at the word. Each word might be a vector that contains the number of times it was used across a variety of documents. If we take every occurance of each word, and count the context words around it, we get a word-word co-occurance matrix. Since *cherry* and *strawberry* were both used a silimar number of times around sugar and pie, we can infer they have similar meanings in that specific context. Note that $\mid V \mid$ is generally the size of the vocabulary.

0.4 6.4 - Cosine for Measuring Similarity

To find the difference between the two vectors, we can use the dot product.

$$v \cdot w = \sum_{i=1}^{N} v_i \cdot w_i = v_1 w_1 + v_2 w_2 ....$$

$$cosine(\theta) = \frac{a \cdot b}{\mid a \mid\mid b \mid}$$

This will give you cosine of theta. For some odd reason, the book stops short of suggesting you take the arc cosine. I might find this more valuable because then you have access to the plain angle. If we want to know what kinds of contexts are shared by cherry and strawberry but not by digital and information, we're not going to get good discrimination from words like the, it, or they, which occur frequently with all sorts of words and aren't informative about any particular word.
It's a bit of a paradox-words that occur too frequently have less value, but words that don't appear enough can't be used reliably.

0.5 6.5 - TF-IDF: Weighing terms in the vector

TF-IDF weighting is the product of two terms-each term is capturing one of two intuitions: The first is the **term frequency**:

$$tf_{t,d} = count(t, d)$$

Here it's common to use the log 10 frequency. So:

$$1 + log_{10}(10) = 2, 100$$

The second factor is meant to weigh which documents the term was used in. Terms that are limited to a few documents are useful for discriminating those documents form the rest of the colleciton. I.E, they have more weight in those documents. If they are used frequently across a variety of documents, they carry less weight.

0.6 TF-IDF Example: Very Helpful

THE WORD ROMEO APPEARS IN ONE SHAKESPEARE DOCUMENT

Consider in the collection of Shakespeare's 37 plays the two words Romeo and action. The words have identical collection frequencies (they both occur 113 times in all the plays) but very different document frequencies, since Romeo only occurs in a single play. If our goal is to find documents about the romantic tribulations of Romeo, the word Romeo should be highly weighted, but not action.

If we want to analyze *Romeo*'s feelings, we would heavily weight the word Romeo, as it only appears in one document.

0.7 6.5 - IDF Explained

If we want to find discriminating words, we call the **inverse document frequency** or *idf.*

$$idf_t = \log_{10}(\frac{N}{df_t})$$

$$w_{t,d} = tf_{t,d} X idf_t$$

0.8 6.6 - Pointwise Mutual Information

Another way to measure the relative importance of a word is **positive pointwise mutual information**. This is a technique that involves determining the association between two words by measuring their relative position in previous documents. This is a technique to determine the association between two words given proximity in the document collection.

$$PMI(w_1, w_2) = log_2(\frac{0.02}{0.10X0.05}) = log_2(\frac{0.02}{0.005}) = log_2(4) = 2$$

These models are often used for document functions like deciding if the two documents are similar. We represent a document by taking the vectors of all the words in the document, computing the cetroid of these vectors.
Centroid:

$$d = \frac{w_1 + w_2 ....}{k}$$

Given k word vectors w_1, w_2 ...

Given this information, we can compute document vectors! It's also useful for document similarity, plaigiarism detection, new recommender systems, music recommendation, etc.

0.9 6.8 - Word2vec

We now introduce embeddings: a short dense vector that contains 50-1000 elements. Believe it or not, these vectors are actually more effective than the vectors with $\mid V \mid$ elements.
In this chpater we will introduce **skip-gram with negative sampling** which is one of the two algorithms that employee a software package called **word2vec**. This system learns contant embeddings which means the vector *doesn't* change with context. I.e. cold and cold mean the same thing regardless of context. n summary, skip-gram trains a probabilistic classifier that, given a test target word w and its context window of L words c1:L, assigns a probability based on how similar this context window is to the target word.
The basis for skip-gram is:

- Treat the target word and a nieghboring word as positive examples
- Randomly sample other words to get negative samples
- Use Logistic Regression to train a classifer to distinguish between the two classes
- Use the learned weights as embeddings

Using logisitc regression, the model learns two separate embeddings for each word *i*, the target embedding, and the context embedding.

0.10 Analogy/Relational Similarility

By expressing these combinations of words as vectors, we can then predict the next word by searching for other words that might be in the same region or extend out from the same vector. The most common metric for testing their similarity is to use similarity word scores. For example *plane and car* had an average score of 5.77.

0.11 Summary

- In vector semantics, a word is modeled as a vector—a point in high-dimensional space, also called an embedding.
- In this chapter we focus on static embeddings, where each word is mapped to a fixed embedding. Vector semantic models fall into two classes: sparse and dense.
- In sparse models each dimension corresponds to a word in the vocabulary V and cells are functions of co-occurrence counts.
- The term-document matrix has a row for each word (term) in the vocabulary and a column for each document.
- The word-context or term-term matrix has a row for each (target) word in the vocabulary and a column for each context term in the vocabulary.

- Two sparse weightings are common: the tf-idf weighting which weights each cell by its term frequency and inverse document frequency, and PPMI (point- wise positive mutual information), which is most common for word-context matrices. Dense vector models have dimensionality 50–1000.
- Word2vec algorithms like skip-gram are a popular way to compute dense embeddings. Skip gram trains a logistic regression classifier to compute the probability that two words are 'likely to occur nearby in text'.
- This probability is computed from the dot product between the embeddings for the two words.
- Skip-gram uses stochastic gradient descent to train the classifier, by learning embeddings that have a high dot product with embeddings of words that occur nearby and a low dot product with noise words.
- Other important embedding algorithms include GloVe, a method based on ratios of word co-occurrence probabilities.
- Whether using sparse or dense vectors, word and document similarities are computed by some function of the dot product between vectors.
- The cosine of two vectors a normalized dot product is the most popular such metric.

0.12 Questions