CS 263: Lecture 2

Word2vec: Lexical Semantic

How can we represent the similarity between two words computationally?

BoW Approach

Problem with the bag of vectors approach:

* Everything has a one hot vector encoding
* Doesn’t capture the frequency of the words
* Doesn’t account for the words having different meanings and new words
* Dimensionality is large, vector is sparse
* There is no similarity(usually you take a dot product to find the similarity, however dot product of two one hot encoded vectors is always 0)

Lexical Semantics: Word meanings that can help decide -:

* Word similarity: Distribution of vector model of meaning
* Word Relations/Context
* Word Sense Disambiguation
* Semantic Rules

Theoretical Foundation of distributional semantics

* “If A and B have almost identical environments, they are called synonyms”
* We figure this out by the frequency of what words does it occur with, etc.

Two classes of vector representation

Sparse vector representations

1. Mutual information weighted word co-occurrence matrices

Model the meaning of a word by “embedding” in the vector space

**Document Matrix:** Each cell has a count term t in a document d. You have a matrix which has dimensionality D\*V, where D is the number of documents, and V is the number of the words.

This does capture the similarity of words. Row vector represents a word vector, column vector refers to the document vector.

You can then measure the similarity of two vector through vector embeddings using a dot product, where basically you multiply two vectors and divide it by the length of the vector.(cosine)

Problems:

* Low frequency words would be underrepresented. And articles would be overrepresented that may not be related to the words but have appearances.
* Dimension count of the vector is small as there are a limited amount of documents

**Word-Word Matrix/ Word-Context Matrix**: Break document into words or paragraphs, and then make frequency matrices out of those new datasets. Word is now defined by a vector over counts of context words. Instead of each vector being of length D, it would be of dimension |V|. Dimensions: |V|\*|V|. However when you use context instead of word, this now becomes a sparse matrix and you can reduce the dimensionality of the matrix.

Facts about this:

* Real Matrix is 50,000\*50,000 (Approx)
* Small windows of 1-3 words would be very syntactic, and larger windows of 4-10 are more semantic. 10+ would give topical contexts.

Problems:

* Wasting a lot of space as most of the matrix is just sparse, computationally inefficient
* Number of basis concepts is large(high dimensional)
* Basis is not orthogonal, and not linearly independent.
* Also same issue with frequency of conjunctions, pronouns, and articles.

Latent Semantic Analysis:

We can apply SVD to the matrix to find latent components. Uncovers relationships not explicit in the corpora. Term vectors projected to k-dim latent space. Over here k<< n -> |V|. They take the U part of the matrix(rows) for the word representations.

Dense Vector Representations-> This is where we use Word2Vec embeddings.

LSA: A compact /low dimensional representation of co-occurrence matrix. Prediction-based models is another way to get dense vectors.

Skip Gram/CBOW

Train a neural network to predict neighboring words. In this way, we learn dense embeddings for the words. This is fast and east to train, and there is a package available online that would help with this (Word2Vec)

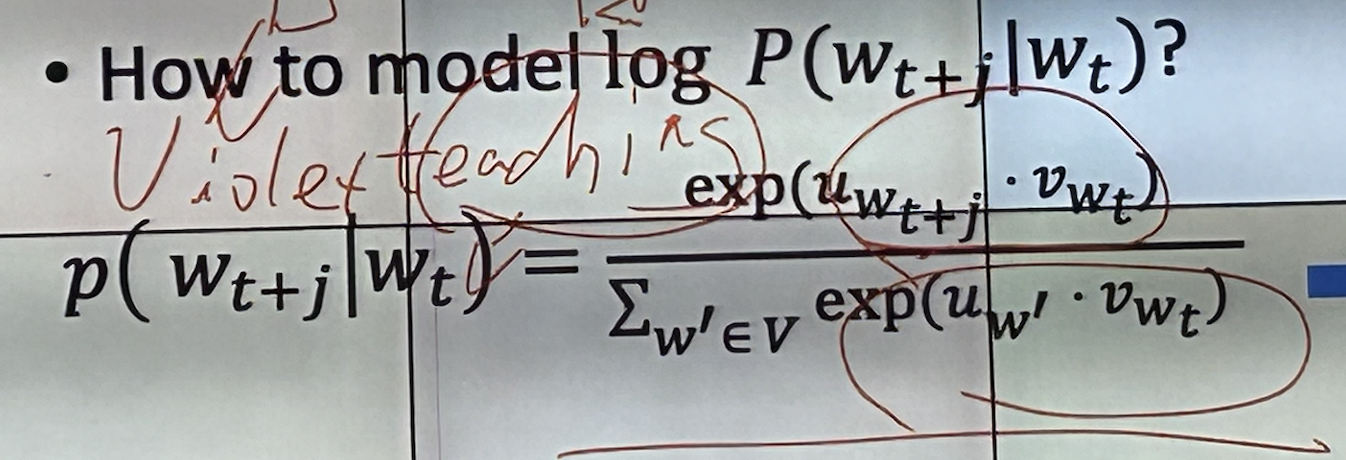
CBOW: Using neighboring words to predict the center word

Skip-Gram-> Reverse of CBOW, you predict the neighboring words using the center word.

Loss function is based on the missing words(Word2Vec)

Model log P(wt+j|wt). You use the softmax function to model the score where j suggest the range or the window of the data.

You do a dot product in the following manner:



Every word has two vectors: uw+t is the output vector embedding, and vwt is the input vector embedding

This is how it is done: Skip Gram:

X is the one hot encoding of the words

Winput is a |V\*N| dimensional matrix which are the model parameters(“Look up table”) h is N hidden dimensions, (Hidden Projection Later for center word passes). Then you have a WToutput |N\*V| matrix. This would lead to a 1\*V vector. This gives the score, which we can feed to a softmax function to calculate the probabilities.

This would give you the most likely word, which you put into the loss function to optimize for performance.