Vector semantics and embeddings unit

- Lexical semantics, words as vectors (last week Monday)
- Word2Vec (**Today**)
- Finish Word2Vec (Friday)
- Begin machine translation unit (next week Monday)

Today:

- Purpose of embeddings
- The premise of Word2Vec
- Training Word2Vec
- Observing results

Vector semantics *is the standard way to represent word meaning in NLP ... The roots of the model lie in the 1950s when two big ideas converged: [using] a point in three-dimensional space to represent the connotation of a word, and the proposal ... to define the meaning of a word by its distribution in language use, meaning its neighboring words or grammatical environments.*

The idea of vector semantics is to represent a word as a point in a multi-dimensional semantic space that is derived from the distributions of word neighbors. Vectors for representing words are called **embeddings***.*

Jurafsky and Martin, Chapter 5, pg 5 & 6

Goal: Find word embeddings, vectors that represent words in a semantic space.

Word2vec premise: Train a classifier on a “fake” task and use that classifier’s weights/parameters as word embeddings.

Word2vec algorithm outline: Given an corpus,

- Find the vocabulary of the corpus
- Collect training data from the corpus
- Train a classifier on that data
- Return the classifier’s weights

The classification task: Given *target word* w and potential *context word* c, is c likely (or, *how likely is* c) to be used near w? That is, is c a true context word for w?

Alice	looked	all	around	her	at	the	flowers
	c_0	c_1	w	c_2	c_3		

The training data: For every token w in the corpus, pair it with the tokens found L positions before it and L positions after it (positive examples). For every positive example, find k randomly chosen negative examples.

w	c_{pos}	c_{neg_0}	c_{neg_1}	c_{neg_2}
around	looked	pineapple	earnestly	asleep
around	all	...		
around	her	...		
around	at	...		

Algorithm parameters:
L, window size ($L = 2$ above)
k, number of negative samples ($k = 3$ above)

Choosing negative samples:

For training a binary classifier, we also need negative examples. ...[For each training instance,] we'll create k negative samples, each consisting of a target w plus a "noise word" c_{neg}. A noise word is a random word from the lexicon, constrained not to be the target word w. ...
The noise words are chosen according to their weighted unigram frequency $p_\alpha(w)$, where α is a weight. ...

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

[Weighting p, for example at $\alpha = .75$] gives better performance because it gives rare noise words slightly higher probability: for rare words, $P_\alpha(w) > P(w)$.

Jurafsky and Martin, 5.5.2, pg 13–14

The classifier: Logistic regression

$$P(+ \mid w, c) = \sigma(\boldsymbol{w} \cdot \boldsymbol{c})$$

where

- $P(+ \mid w, c)$ is the probability c is a context word for w.
- $\boldsymbol{w}$ is a vector for w as a target word.
- $\boldsymbol{c}$ is a vector for c as a context word.
- $\boldsymbol{w} \cdot \boldsymbol{c}$ is the dot product
- $\sigma(x) = \frac{1}{1+exp(-x)}$ is the logistic (sigmoid) function.

Parameters to the classifier: **W** and **C**, each $V \times D$ matrices.

Parameter to the algorithm: D, length of embedding

The loss function: The cross-entropy (negative log likelihood) loss. For a given data point $(w, c_{pos}, c_{neg_0}, \ldots c_{neg_{k-1}})$, the loss is

$$-\left(\log \sigma(\boldsymbol{c_{pos}} \cdot \boldsymbol{w}) + \sum_{i=0}^{k-1} \log \sigma(-\boldsymbol{c_{neg_i}} \cdot \boldsymbol{w})\right)$$

The training algorithm: Stochastic gradient descent. For each data point $(w, c_{pos}, c_{neg_0}, \ldots c_{neg_{k-1}})$,

$$\begin{aligned}
\boldsymbol{c_{pos}} &-= \eta(\sigma(\boldsymbol{c_{pos}} \cdot \boldsymbol{w}) - 1)\boldsymbol{w} \\
\boldsymbol{c_{neg_i}} &-= \eta(\sigma(\boldsymbol{c_{neg_i}} \cdot \boldsymbol{w}))\boldsymbol{w} \\
\boldsymbol{w} &-= \eta\left((\sigma(\boldsymbol{c_{pos}} \cdot \boldsymbol{w}) - 1)\boldsymbol{c_{pos}} + \textstyle\sum_{i=0}^{k-1}(\sigma(\boldsymbol{c_{neg_i}} \cdot \boldsymbol{w}))\boldsymbol{c_{neg_i}}\right)
\end{aligned}$$

Parameter to the algorithm: η, the learning rate

Recall that the skip-gram model learns two separate embeddings for each word i: the target embedding $\boldsymbol{w_i}$ and the context embedding $\boldsymbol{c_i}$, stored in two matrices, the target matrix $\mathbf{W}$ and the context matrix $\mathbf{C}$. It's common to just add them together, representing word i with the vector $\boldsymbol{w_i} + \boldsymbol{c_i}$. Alternatively we can throw away the $\mathbf{C}$ matrix and just represent each word i by the vector $\boldsymbol{w_i}$.

Jurafsky and Martin, Chapter 5, pg 15

Coming up:

- Work on stylo project (Fri, Dec 12)
- Take Word2Vec quiz (Fri, Nov 21)
- Do Word2Vec programming assignment (Fri, Dec 5)
- Read J&M Chapter 8 (Mon, Nov 24—class time