

fasttext:

- Developed by Facebook AI Labs about 5 years ago. In word2vec and GloVe, the word is the smallest unit.

Instead, consider each word to be ngrams of individual characters

characters: computationally expensive, more things to train
 $\text{writer} \Rightarrow (\text{writer}, \text{write}, \text{writ}, \dots)$

advantage: certain rare words are almost never found, but its n-gram subwords are.

Uses several optimizations, notably:

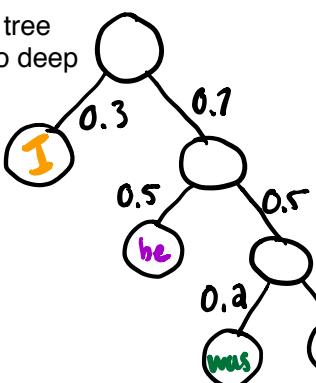
How to optimize word2vec, to make algorithm faster -> Hierarchical softmax

$V \approx 900,000$ for GloVe

$V \approx 3,000,000$ for word2vec

1. computation is extremely expensive.
2. negatively impacts accuracy

don't have to do exponential function all over again like the last layer of word2vec



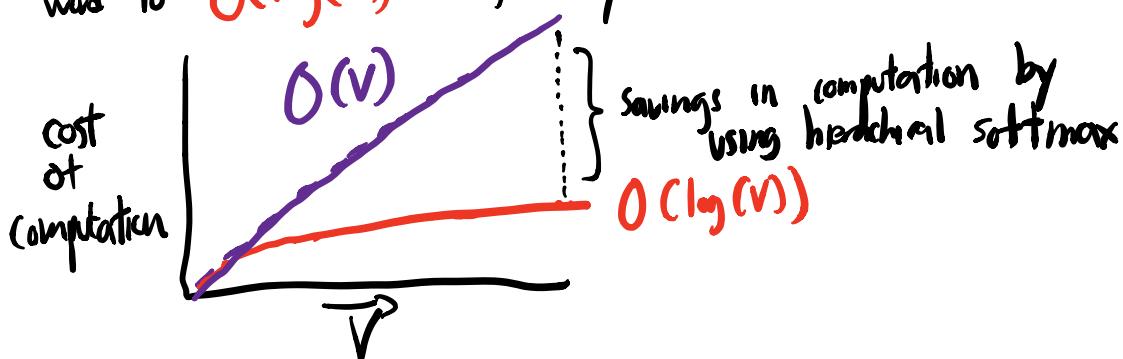
$$p(I) = 0.3$$

$$p(\text{he}) = 0.7 \times 0.5 = \text{sum to } 1.$$

$$p(\text{was}) = 0.7 \times 0.5 \times 0.2$$

$$p(dy) = 0.7 \times 0.5 \times 0.8$$

This reduces the number of operations from $O(V)$ → every single word to $O(\log(V))$ → a significantly lower number.



Larger D \rightarrow more robust model, and more dimensions to evaluate
but takes more time to train, increases dimensionality for downstream model

GLOVE (Global Vectors for word representation)

Most matrices (like most scalar numbers), can be factored into smaller subcomponents. For instance, a matrix R of shape $V \times V$ can be reconstructed

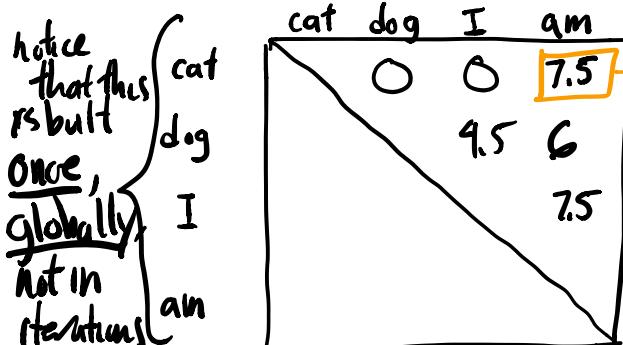
ballpark for sample to dimension ratio:
square root of sample number
but it depends; pick the highest model
you can that does not overfit
and does not take too much time to train
model

$$R_{V \times V} = W_{V \times D} \times D_{D \times V}$$

from two smaller matrices W and D . The D specifies the size of your word embeddings, (i.e. 100, or 200, or 300)

Notice that W and D are the same shapes as the word and context vectors in wordvec! They will serve similar roles!

① Build a term-term matrix for your corpus:



every time am, cat are found together in the same context window, add $1 \times \frac{1}{\text{context distance}}$.

$$\text{I am a dog} \quad 1 \times \frac{1}{2} = \frac{1}{2}$$

Properties of this matrix:

- most values will be zero
- the non-zero values will be extremely high

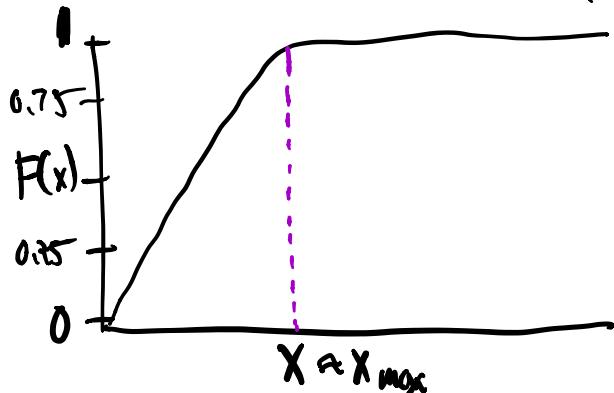
solution: take $x \rightarrow \log(x+1)$ to smooth out the numbers

③ Apply a ceiling function to this matrix to weight every entry in the matrix.

- find X_{\max} (the largest value in the matrix)
- apply the function $F(x)$

$$\begin{cases} x = X_{\max} & 1 \\ x < X_{\max} & \left(\frac{x}{X_{\max}}\right)^{\alpha} \end{cases}$$

α is a hyperparameter selected to smooth the values



The final objective function is the error we are minimizing

$$J(\theta) = \sum_i^V \sum_j^V F(X_{ij}) (w_i^T v_j - \log(X_{ij} + 1))^2$$

for i in words:
for j in words:
... a weighting between 0 and 1

word vector for word i
vector for word j
our ground truth

want w^*v updated closer to ground truth
we care about the updated weight matrix!
they represent the word we care about

We then take the derivatives with respect to w and v to get our updates. We try to minimize the difference

Advantages of GloVe vs word2vec:

- Takes into consideration global context.
- less complicated architecture and can be computationally less costly.

between $w_i^T v_j$ (our prediction) and $\log(X_{ij} + 1)$ (the truth)

Sentence Vectors

- ① Take the average of the word vectors in the sentence:

$$w_{\text{sentence}} = \frac{\sum_i^N w_i}{N}$$

$N = \# \text{ of words in sentence}$

$w_1 = I = [2 \ 0 \ -1 \ 2 \ 0 \ -1]$

$w_2 = \text{like} = [-1 \ 0 \ -1 \ 2 \ 1 \ 0]$

$w_3 + \text{cats} = [0 \ 1 \ 0 \ -1 \ 2 \ 1]$

$[1 \ 1 \ -2 \ 3 \ 3 \ 0] / 3 = [\frac{1}{3} \ \frac{1}{3} \ -\frac{2}{3} \ 1 \ 1 \ 0]$

Assumption: all words in the sentence are weighted equally (sometimes OK, sometimes not valid)

- ② Use a weighting scheme (TF-IDF) to weight the relative importance of each word within a sentence.

$$f(\text{word}, \text{sentence}) = \text{TF}_{\text{word}, \text{sentence}} \times \text{IDF}_{\text{word}}$$

$$w_{\text{sentence}} = \frac{\sum_i^N f(\text{word}_i, \text{sentence}) \times w_i}{\sum_i^N f(\text{word}_i, \text{sentence})}$$

w_i ↗ word vector for word i

Weighted average based on TFIDF score

$N = \# \text{ of words in sentence}$

$$f(\text{word}_i, \text{sentence}) = \text{sum of all the TF-IDF scores for a document(sentence)}$$

$\text{TF IDF score for word}_i$
in sentence.

Example: Get the document embedding for the sentence "He ate lunch".

TF [ate ... he ... lunch]
 everything else here is 0.

$$\text{IDF} [2 \dots 1.5 \dots 4]$$

$$\text{TF-IDF} = [\frac{\text{ate}}{2}, \frac{\text{he}}{1.5}, \frac{\text{lunch}}{4}] \text{ typo}$$

word embedding (pretrained from word2vec):

$$\text{he} = [-1 \ 0 \ 3 \ 4]$$

$$\text{ate} = [1 \ 0 \ .5 \ 1]$$

$$\text{lunch} = [-2 \ -2 \ 1 \ 0]$$

$$W_{\text{sentence}} = \frac{\sum_i^N \text{tfidf}_i \times w_i}{\sum_i^N \text{tfidf}_i}$$

$$= (1.5) \times [-1, 0, 3, 4] + (2) \times [1, 0, .5, 1] + 3.5 [-2, -2, 1, 0]$$

$$\uparrow \text{TFIDF}$$

embeddings
for
"he"

$$\uparrow$$

$$\uparrow \text{TFIDF}$$

embedding
for
"ate"

$$\uparrow \text{TFIDF}$$

embedding
for
"lunch"