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:

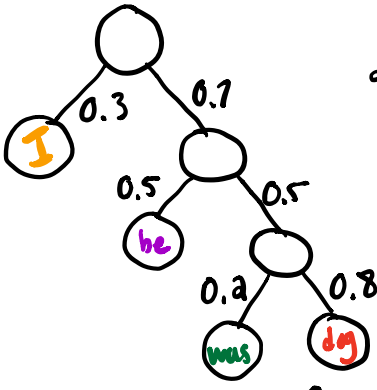
writer ⇒ (writer, write, writ, ...)

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

Uses several optimizations, notably:

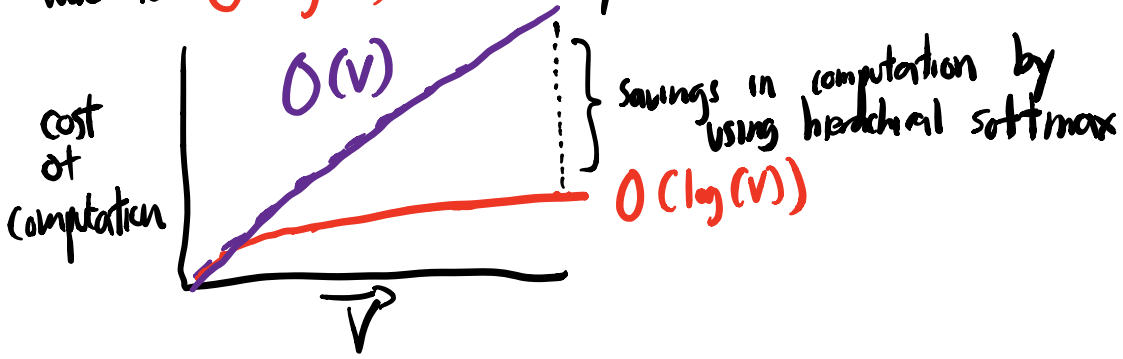
- Hierarchical softmax: V=900,000 for GloVe
  V=3,000,000 for word2vec

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



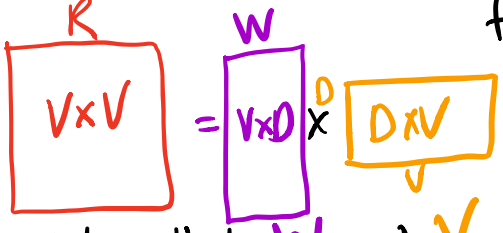
p(I) = 0.3
p(be) = 0.7×0.5 = sum to 1.
p(was) = 0.7×0.5×0.2
p(dog) = 0.7×0.5×0.8

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



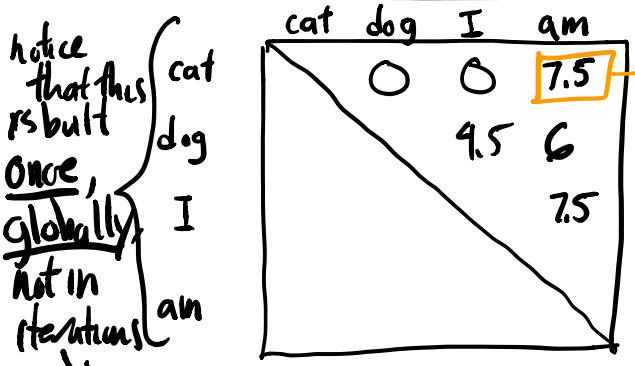
GLOVE (Global Vectors for word representation)

Most matrices (like most scalar numbers), can be factored into smaller subcomponents. For instance, a matrix R of shape V×V can be reconstructed from two smaller matrices W and V. The D specifies the size of your word embeddings; (ie. 100, or 200, or 300)



Notice that W and V are the same shapes as the word and context vectors in word2vec! They will serve similar roles!

① Build a term-term matrix for your corpus:



notice that this is built once, globally, not in iterations

→ hence Global vectors for word representation

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

I am a dog distance of 2 $1 \times \frac{1}{2} = \boxed{\frac{1}{2}}$

Properties of this matrix:

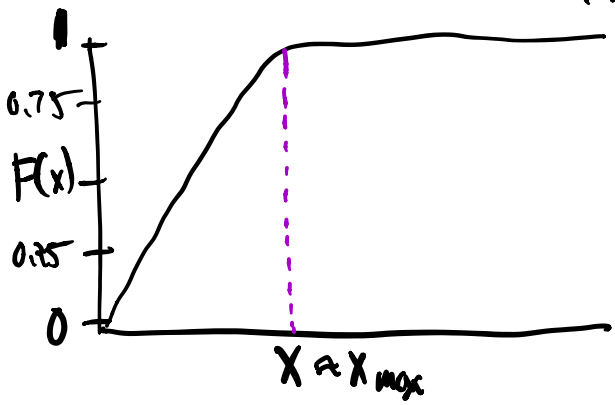
- most values will be zero
- the nonzero values will be extremely high

} ③ solution: take $x \to \log(x+1)$ 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 net effect is that high co-occurence relationships (where X_{ij} is a very large #) will be weighted to most heavily.

The final objective function is the error we are minimizing

$$J(\theta) = \sum_{i}^{V} \sum_{j}^{V} F(X_{ij}) \left(w_i^T v_j - \log(X_{ij}+1)\right)^2$$

for i in words:
  for j in words:
    ...

- $F(X_{ij})$: a weighting between 0 and 1
- w_i: word vector for word i
- v_j: vector for word j
- $\log(X_{ij}+1)$: our grand truth

We then take the derivatives with respect to w and v to get our updates. We try to minimize the difference between $w_i^T v_j$ (our prediction) and $\log(X_{ij}+1)$ (the truth)

Advantages of GloVe vs word2vec:

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

Sentence Vectors

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

$$W_{sentence} = \frac{\sum_{1}^{N} W_i}{N}$$

N = # of words in sentence

W_1 I = [2 0 -1 2 0 -1]
W_2 like = [-1 0 -1 2 1 0]
W_3 + cats = [0 1 0 -1 2 1]
[1 1 -2 3 3 0] / 3 = [1/3 1/3 -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.

$$F(word, sentence) = TF_{word, sentence} \times IDF_{word}$$

$$W_{sentence} = \frac{\sum_{1}^{N} F(word_i, sentence) \times W_i}{\sum_{1}^{N} F(word_i, sentence))}$$

W_i ← word vector for word i

N = # of words in sentence

$F(word_i, sentence)$ = TF IDF score for $word_i$ in sentence.

$\sum_{1}^{N} F(word_i, sentence))$ sum of all the TF-IDF scores for a document (sentence)

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

ate ... he ... lunch

TF [1 ... 1 ... 1] ← everything else here is 0.

ate he lunch

IDF [2 ... 1.5 ... 4]

---

ate he lunch

TF-IDF = [2 1.5 4]

word embedding (pretrained from word2vec):

he = [-1 0 3 4]

ate = [1 0 .5 1]

lunch = [-2 -2 1 0]

$$W_{sentence} = \frac{\sum_{i}^{N} tfidf_i \times w_i}{\sum_{i}^{N} tfidf_i}$$

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

TFIDF — embeddings for "he" — TFIDF — embedding for "ate" — TFIDF — embedding for "lunch"