Classical NLP

Machine Learning

Denis Litvinov

November 17, 2022

Table of Contents

Preprocessing

Example: "The cat sits on a larger mat."
Text normalization:

- remove punctuation.
- remove stopwords = remove redundant words like articles or prepositions (*the, on, a*).
- lemmatize = revert the word to the initial form (*sits -> sit*).
- stem = cut the word endings (*larger -> larg*)

Tf-Idf

Tf-Idf implies a *Bag of Words* model, where the order of words (tokens) is not important.
A document $d = w_1...w_t$ is represented as a vector of elements v_i = tf-idf(w_i, d) for each unique w$_k in the sentence$.

$$tf - idf(w, d) = tf(w, d) * idf(w, d)$$

Tf-Idf

Term frequency *tf(w,d)* is a relative frequency of term w within document d.

$$tf(w,d) = \frac{count(w,d)}{\sum_{w' \in d} count(w',d)}$$

Inverse document frequency *idf(w, d)* shows how common is the word across all documents.

$$idf(w,d) = \frac{N}{\sum_{i=1}^{D} [w \in d_i]}$$

Tf-Idf

Normalization: reduces the dependence on the document length.

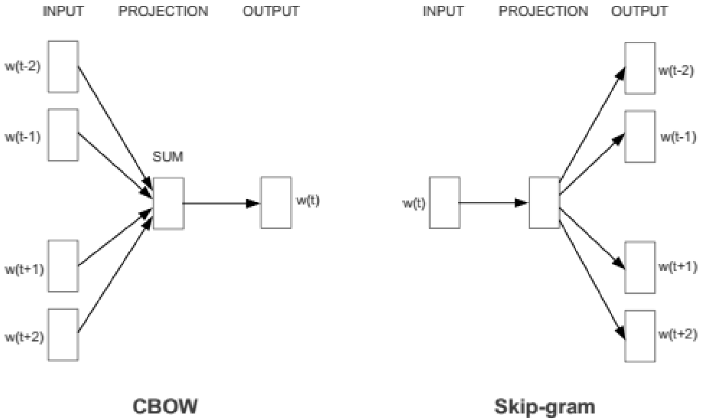
$$v \to \frac{v}{||v||}$$

Hashing trick: How to deal with Out of Vocabulary words? Replace $w \to hash(w)$

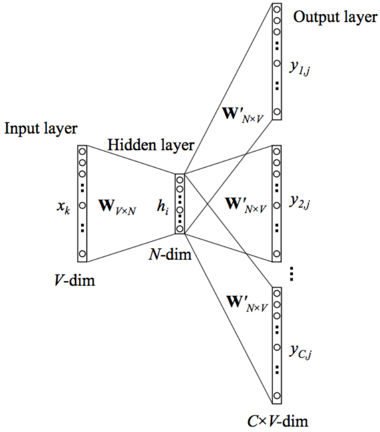
Problem Statement

Given a sequence of tokens (words), build a vector in R^N for each token, which are in some sense representative.

Word2Vec Model



Skip-gram Model



where N - desired embedding dimension
V - vocabulary size
C - context size

Skip-gram Model

For each word t predict surrounding words in a window of size m (context)
Objective is to maximize probability of context words given the current center word:

$$J(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq m; j \neq 0} p(x_{t+j}|x_t; \theta) \rightarrow \max_{\theta}$$

, or

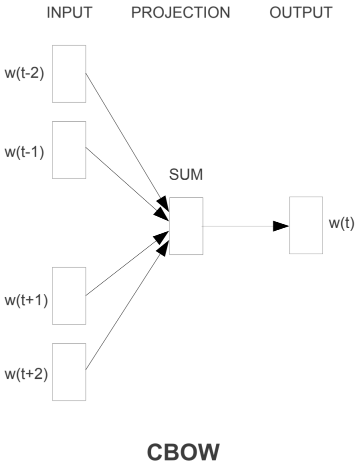
$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m; j \neq 0} \log p(x_{t+j}|x_t; \theta) \rightarrow \min_{\theta}$$

where x_t - center word,
x_{t+j} - word from context,
m - context size.

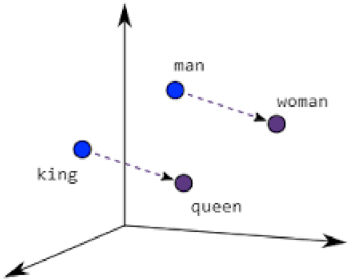
$$p(x_{t+j}|x_t) = p(out|center) = \frac{e^{u_{out}^T v_{center}}}{\sum_{i=1}^{V} e^{u_i^T v_{center}}}$$

Continuous Bag of Words Model

= Predict center word from surrounding context

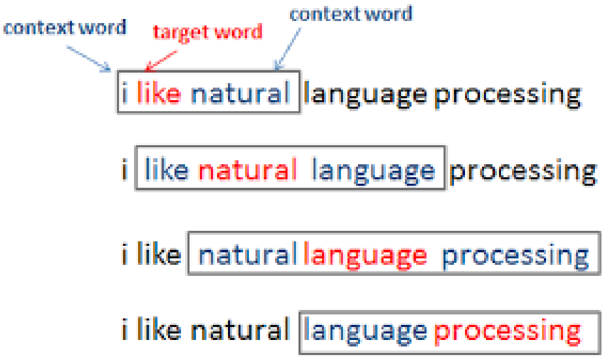


Why Embeddings?



- What are other ways to construct a vector in R^N for each word?
- Embeddings allow to apply simple algebra on words
- Embeddings can describe entities (words, documents) that are absent in the dataset.

Distributional hypothesis



Word embedding is defined by it’s context.

Toy example on Machine Translation

Let $E_1, E_2 \in R^{NxD}$ be matrices of word embeddings for source and target language respectively.
Then we can train a MT model with loss:

$$||E_1 - E_2 U||_F \rightarrow \min_U$$

We also can add condition $U^T U = I$

Problem statement

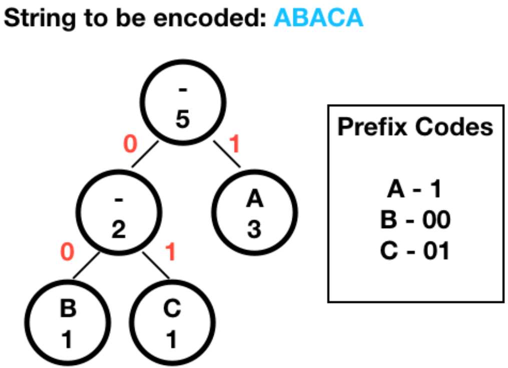
What computational problems do you see in the objective function?

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m; j\neq 0} \log p(x_{t+j}|x_t;\theta) \to \min_{\theta}$$

$$p(x_{t+j}|x_t) = p(out|center) = \frac{e^{u_{out}^T v_{center}}}{\sum_{i=1}^{V} e^{u_i^T v_{center}}}$$

Huffman tree

How to build a binary prefix tree?



Huffman tree

Complexity $O(V) \to O(\log_2 V)$
$x = v_{n(x,j)}^T v_x$,
where $n(x,j)$ is the j-th node on the path from the root to token x.
$p(n, left) = \sigma(v_n^T v_x)$ - probability to go left.
$p(n, right) = \sigma(-v_n^T v_x)$ - probability to go right.
Then,
$p(x_j|x) = \prod_{j=1}^{L(x)-1} \sigma(I[n(x, j+1) == child(n(x,j))] v_n^T v_x)$,
where $L(x)$ - depth of the tree,
$child(x)$ - child of node n.

Huffman tree

Using negative sampling with k samples:

$$\log p(x_{t+j}|x_t;\theta) = \log \sigma(u_{outer}^T v_{center}) + \sum_{i=1}^{k} E_{j\sim P(x)}[\log \sigma(-u_j^T v_{center})]$$

Co-occurrence matrix

= Word embeddings through decomposition of co-occurrence matrix

1. I enjoy flying.
2. I like NLP.
3. I like deep learning.

The resulting counts matrix will then be:

$$X = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array} \begin{array}{c} \begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\ \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

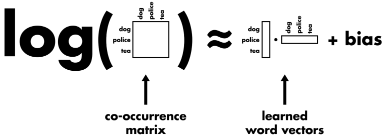
Glove

P_{ij} - occurrence of i-th word along with j-th in the window of size m
Cons:

1. Very high-dimensional, not used in practice
2. Hard to add new words and docs

Trivial solution: use some dimension-reduction method, usually SVD

Glove



$J(\theta) = \frac{1}{2}\sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})$

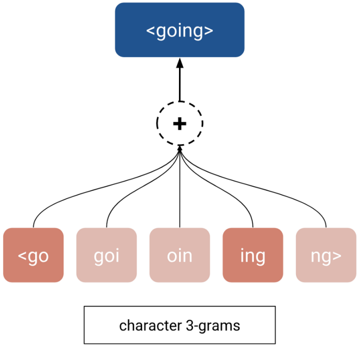
$f(x)$ - some weight functions that obeys following properties:

- $f(0) = 0$
- non-decreasing, so rare co-occurrences won't overweight
- relatively small for large x, to compensate frequent co-occurrences

The authors have chosen

$$f(x) = \begin{cases} (\frac{x}{x_{max}})^{\alpha} & x \leq x_{max} \\ 1 & else \end{cases}$$

FastText



FastText

Subword embeddings.
Introduce scoring function (instead of scalar product as in w2v):

$$s(w, c) = \sum_{g \in G_w} z_g^T v_c$$

where G_w - set of 3-grams appearing in word w
z_g - embedding of 3-gram g
v_c - context vector

FastText

Objective function for skip-gram case:

$$\sum_{t=1}^{T}[\sum_{c\in C_t} log(1+e^{-s(w_t,w_c)}) + \sum_{n\in N_{t,c}} \log(1+e^{s(w_t,n)})] \rightarrow \min$$

where c - chosen context position
C_t set of context position dependent on current word t
T - total number of words
$N_{t,c}$ - set of negative samples dependent on chosen word and context
Inference: Embedding of word w from 3-grams G_w:

$$v_w = \sum_{g\in G_w} z_g$$

FastText

Tweaks in Negative sampling: sampling with probability

$$p(w) = \frac{\sqrt{U(w)}}{Z}$$

where $Z = \sum_w \sqrt{U(w)}$
and $U(w)$ - the count of a particular word w
Probability of token w to be discarded during training:

$$P(w) = \sqrt{\frac{t}{f(w)}} + \frac{t}{f(w)}$$

where $f(w) = \frac{U(w)}{Z}$ - frequency of token w

Problem Statement

Probability of a sequence of tokens w_i:

$$P(w_1, .., w_T) = P(w_1)P(w_2|w_1)...P(w_T|w_1, ..., w_{T-1})$$

Perplexity

Common quality metric language modeling is perplexity (smaller is the better):

$$Q(w_1, .., w_T) = P(w_1, .., w_T)^{-\frac{1}{T}}$$

Perplexity of a corpus of text:
As

$$C = s_1, .., s_m$$

then

$$P(c) = \prod_{i=1}^{m} P(s_i)$$

$$Q(C) = (\prod_{i=1}^{m} P(s_i))^{-\frac{1}{m}} = 2^{\log_2(\prod_{i=1}^{m} P(s_i))^{-\frac{1}{m}}} =$$

$$= 2^{-\frac{1}{m}\log_2(\prod_{i=1}^{m} P(s_i))} = 2^{-\frac{1}{m}\sum_{i=1}^{m} log_2 P(s_i)}$$

N-gram Language Models

N-gram language model = (finite horizon):

$$P(w_t|w_1,...,w_{t-1}) = P(w_t|w_{t-n+1},...,w_{t-1})$$

$$P(w_1,..,w_T) = \prod_{t=1}^{T} P(w_t|w_{t-n+1},...,w_{t-1})$$

How to estimate $P(w_t|w_{t-n+1},...,w_{t-1})$?

N-gram Language Models

$$P(w_t|w_{t-n+1},...,w_{t-1}) = \frac{\#(w_{t-n+1},...,w_{t-1},w_t)}{\sum_{\hat{w}}\#(w_{t-n+1},...,w_{t-1},\hat{w})}$$

OR

$$P(w_t|w_{t-n+1},...,w_{t-1}) = \frac{\#(w_{t-n+1},...,w_{t-1},w_t)^{\frac{1}{\tau}}}{\sum_{\hat{w}}\#(w_{t-n+1},...,w_{t-1},\hat{w})^{\frac{1}{\tau}}}$$

where τ is called temperature. Higher temperature means more flat probability distribution.

Problems with naive approach?

Laplace smoothing

$$P(w_t|w_{t-n+1},...,w_{t-1}) = \frac{\#(w_{t-n+1},...,w_{t-1},w_t)+\delta}{\sum_{\hat{w}}[\#(w_{t-n+1},...,w_{t-1},\hat{w})+\delta]}$$

where δ accounts for missing n-grams

Linear Interpolation

$$P(w_3|w_1, w_2) = \lambda_3 \frac{\#(w_1, w_2, w_3)}{\sum_{\hat{w}} \#(w_1, w_2, \hat{w})} + \lambda_2 \frac{\#(w_2, w_3)}{\sum_{\hat{w}} \#(w_2, \hat{w})} + \lambda_1 \frac{\#(w_3)}{\sum_{\hat{w}} \#(\hat{w})}$$

General case, where λ is some hyperparameter, $\sum_i \lambda_i = 1$

Kneser-Ney

$$P(w_t|w_{t-1}) = \frac{\max(0, \#(w_{t-1}, w_t) - \delta)}{\sum_{\hat{w}} \#(w_{t-1}, \hat{w})} + \lambda_{w_{t-1}} P(w_t)$$

where

$$P(w_t) = \frac{|\{w' : \#(w', w_t) > 0\}|}{|\{(w', w'') : \#(w', w'') > 0\}|}$$

$$\lambda_{w_{t-1}} = \delta \frac{|\{\hat{w} : \#(w_{t-1}, w_t) > 0\}|}{\sum_{\hat{w}} \#(w_{t-1}, \hat{w})}$$

Problem Statement

Inference for Language Models: we would like to generate a new sequence of tokens from conditioned on some input.
And usually we want the generated sequence to be the most probable for the input, and at the same time diverse enough not to collapse into the same mode.

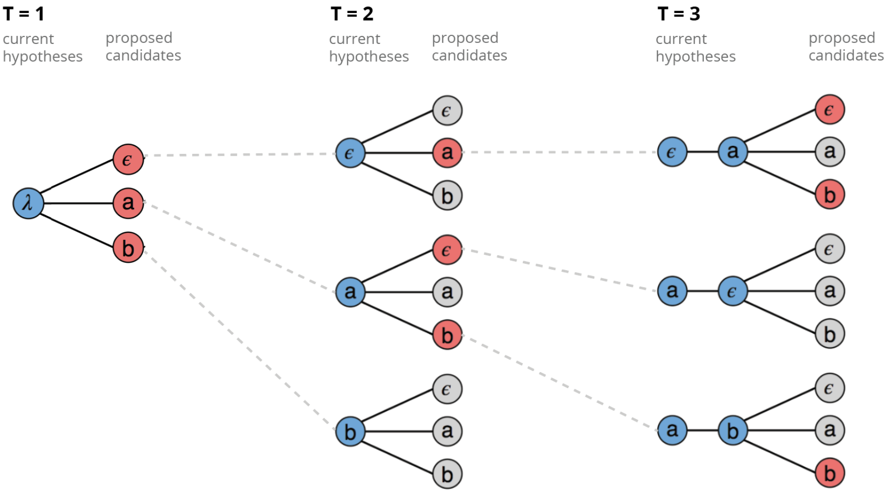
```
What's the time? -> I don't know.
How are you? -> I don't know.
...
```

Argmax

$$w_t = argmaxP(w_t | w_{t-n+1}, ..., w_{t-1})$$

However, greedy selecting the tokens not necessarily mean producing the most probable sequence.

Beam search: greedy, but smarter



Standard beam search algorithm with an output

Sampling

Sample token from distribution

$$w_t \sim P(w_t | w_{t-n+1}, ..., w_{t-1})$$

With temperature, controlling "the randomness" of generated sequence

$$w_t \sim P(w_t | w_{t-n+1}, ..., w_{t-1}; \tau)$$

using softmax with temperature

$$p(w_i) = \frac{e^{z_i/\tau}}{\sum_j e^{z_j/\tau}}$$