Natural Language Processing
Lecture 03 Word Embeddings

Qun Liu, Valentin Malykh
Huawei Noah's Ark Lab

Autumn 2020
A course delivered at MIPT, Moscow

Content

Content

Word representations

- In rule-based approaches, i.e., grammars, automata, etc., words are represented as symbols.
- However, if we want to apply machine learning algorithms, we should represents linguistic units (words, phrases, etc.) as numerical vectors.
- Then the questions is, how can we represent words as numerical vectors?

Representing words by their context

"You shall know a word by the company it keeps"

(J. R. Firth, 1957)

- Distributional hypothesis:
  Linguistic items with similar distributions have similar meanings.

  $\Rightarrow$ **Distributional Semantics**

Distributional semantics

- *Distributional semantics* is a research area that develops and studies theories and methods for quantifying and categorizing semantic similarities between linguistic items based on their distributional properties in large samples of language data. (Wikipedia)
- Idea: Collect distributional information in high-dimensional vectors, and to define distributional/semantic similarity in terms of vector similarity.

when the door opened and	Doctor	Livesey came in on
visit to my father Oh	doctor	we cried what shall
s end ! said the	doctor	No more wounded than
back with the basin the	doctor	had already ripped up
great spirit Prophetic said the	doctor	touching this picture with
him First he recognized the	doctor	with an unmistakable frown

A sample of concordance

Distributional semantic models

- Distributional semantic models differ primarily with respect to the following parameters:
  - Context type (text regions vs. linguistic items)
  - Context window (size, extension, etc.)
  - Frequency weighting (e.g. entropy, pointwise mutual information, etc.)
  - Dimension reduction (e.g. random indexing, singular value decomposition, etc.)
  - Similarity measure (e.g. cosine similarity, Minkowski distance, etc.)

Example: Window based co-occurrence matrix

- Window length 1 (more common: 5–10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Window based co-occurrence matrix

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Problems with simple co-occurrence vectors

Increase in size with vocabulary

Very high dimensional: requires a lot of storage

Subsequent classification models have sparsity issues

→ Models are less robust

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Solution: Low dimensional vectors

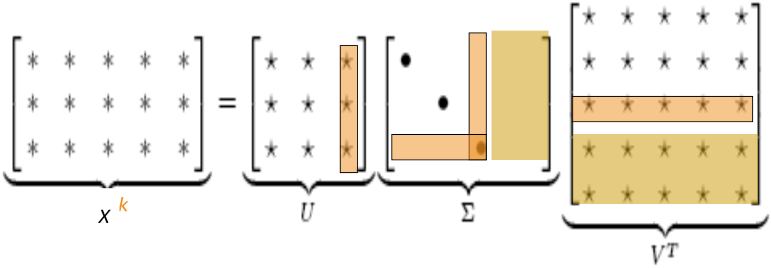
- Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
- Usually 25–1000 dimensions, similar to word2vec
- How to reduce the dimensionality?

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Method: Dimensionality Reduction on X (HW1)

Singular Value Decomposition of co-occurrence matrix X

Factorizes X into $U\Sigma V^T$, where U and V are orthonormal



Retain only k singular values, in order to generalize.
$\hat{X}$ is the best rank k approximation to X, in terms of least squares.
Classic linear algebra result. Expensive to compute for large matrices.

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Simple SVD word vectors in Python

Corpus:
I like deep learning. I like NLP. I enjoy flying.

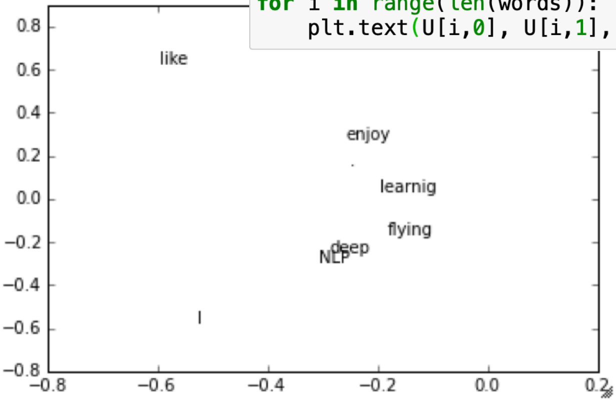
```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Simple SVD word vectors in Python

Corpus: I like deep learning. I like NLP. I enjoy flying.
Printing first two columns of U corresponding to the 2 biggest singular values




```python
for i in range(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

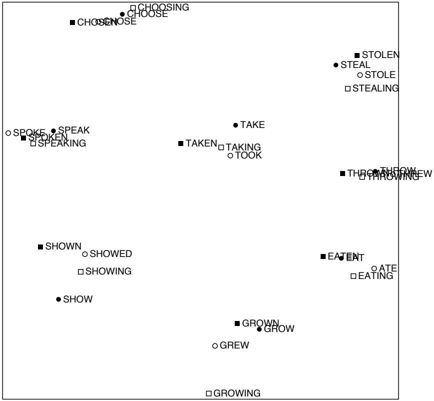
Hacks to X (several used in Rohde et al. 2005)

Scaling the counts in the cells can help ***a lot***

- Problem: function words (*the, he, has*) are too frequent → syntax has too much impact. Some fixes:
  - min(X,t), with t ≈ 100
  - Ignore them all
- Ramped windows that count closer words more
- Use Pearson correlations instead of counts, then set negative values to 0
- Etc.

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

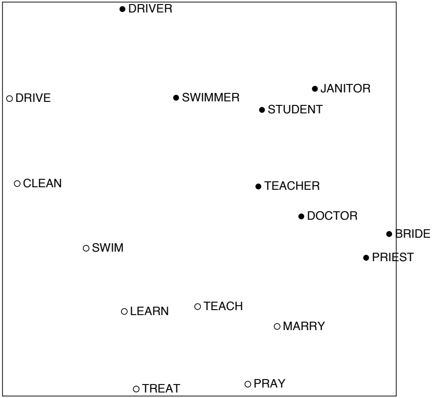
Interesting syntactic patterns emerge in the vectors



COALS model from
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. ms., 2005

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Interesting semantic patterns emerge in the vectors



COALS model from
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. ms., 2005

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

Count-based representations

- The vectors used in distributional semantics are based on frequencies, which are also called count-based representations.
- Count-based representations were successful in many similarity-related tasks, however, their usage was not able to extended to other NLP tasks.
- In order to take full advantages of machine learning / deep learning approaches, it is necessity to represent words solely in vectors. In another word, we must use vectors to replace words completely, and get rid of symbols in computing, except for the output layer.

Prediction-based representations

- If a word can be predicted by the vectors of its content words, then we can expect we can use the vectors to replace the words in any NLP tasks.
- Prediction-based word representations:
  - Randomly assign a vector to each word in the vocabulary;
  - Prepare a corpus;
  - For each of the word (referred as the current word) in the corpus, repeat:
    - Calculate the probability of all content words given the current word (or vice versa);
    - Adjust the word vectors to maximize the above probability.

Word embeddings

- Various predict-based *word representations*, or *word embeddings*, are developed, including:
  - Word2Vec, GloVe, FastText, etc.
- Word embeddings are the first step towards deep learning (neural network) based NLP
- In some cases, the pretrained word embeddings (like Word2Vec) can be directly used to solve NLP problems.
- However, this is not always the case.
- Instead, word embeddings are defined as components of the whole NLP system, whose parameters are tuned together with all other parameters.

Content

Word2Vec

- T Mikolov, I Sutskever, K Chen, GS Corrado, J Dean, Distributed representations of words and phrases and their compositionality, NIPS 2013
  - Word2Vec (include Skip-gram (SG) and Continuous Bag of Word (CBOG))
- Y Goldberg, O Levy. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. arXiv:1402.3722
- A Joulin, É Grave, P Bojanowski, T Mikolov, Bag of Tricks for Efficient Text Classification. EACL 2017.
  - FastText

Tomas Mikolov

Content

3. Word2vec: Overview

Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

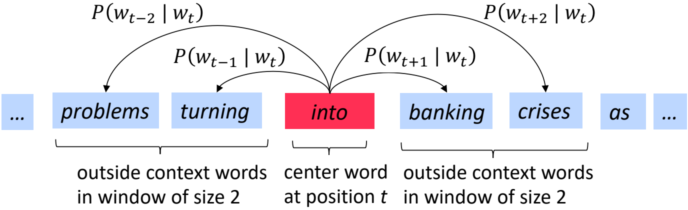
Idea:

- We have a large corpus of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Word2Vec Overview

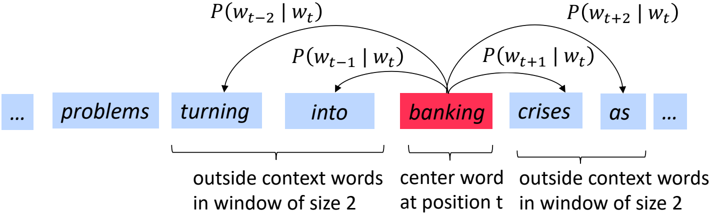
- Example windows and process for computing $P(w_{t+j} \mid w_t)$



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Word2Vec Overview

- Example windows and process for computing $P\left(w_{t+j} \mid w_t\right)$



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

Word2vec: objective function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_j.

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

sometimes called *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\Longleftrightarrow$ Maximizing predictive accuracy

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Cross-entropy loss function

- Negative Log Likelihood Loss:

$$J(\theta) = -\frac{1}{T}\log L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m\\ j\neq 0}} \log(w_{t+j}|w_t;\theta)$$

  is also called Cross-Entropy Loss.
- Cross-entropy is a measure of the difference between two probability distributions for a given random variable or set of events.

Cross-entropy loss function

- Consider two probability distributions p and q defined on a set of events $X = \{x_1, x_2, ..., x_n\}$, then cross-entropy between p and q is:

$$H(q, p) = -\sum_{x \in X} q(x) \log p(x)$$

- Assume X is the vocabulary, $p(x)$ is the model generated probabilities over the vocabulary, $q(x)$ is the actual distribution of the content word at $t + j$:

$$q(x) = \begin{cases} 1, & \text{for } x = w_{t+j} \\ 0, & otherwise \end{cases}$$

then:

$$H(q, p) = -\log p(w_{t+j})$$

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m\\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

- <u>Question:</u> How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?
- <u>Answer:</u> We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word
- Then for a center word *c* and a context word *o*:

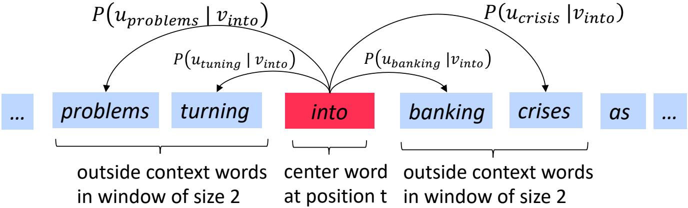
$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

22

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Word2Vec Overview with Vectors

- Example windows and process for computing $P(w_{t+j} \mid w_t)$
- $P(u_{problems} \mid v_{into})$ short for $\mathrm{P}(problems \mid into \,; u_{problems}, v_{into}, \theta)$



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

Word2vec: prediction function

② Exponentiation makes anything positive

① Dot product compares similarity of *o* and *c*.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

$$P(o|c) = \frac{\exp(\boxed{u_o^T v_c})}{\boxed{\sum_{w \in V} \exp(u_w^T v_c)}}$$

③ Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \to (0,1)^n$

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

Open region

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Softmax

- In mathematics, the *softmax* function, also known as *softargmax* or *normalized exponential function*, is a function that takes as input a vector of K real numbers, and normalizes it. We could interpret this as a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers.
- The standard (unit) softmax function $\sigma : \mathbb{R}^K \to \mathbb{R}^K$ is defined by the formula:

$$y_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

Softmax

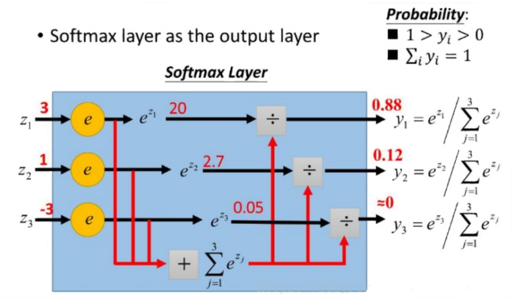
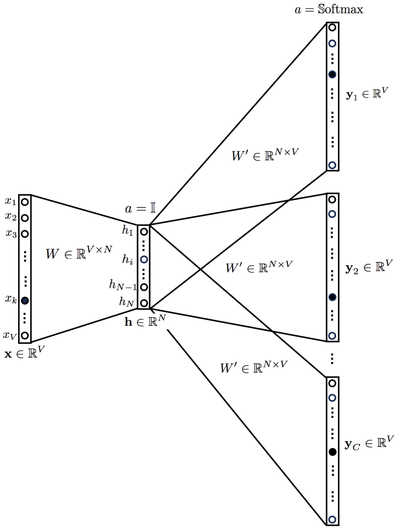


Figure source:https://blog.csdn.net/xg123321123/article/details/80781611

- In Word2Vec, because the softmax function is calculated over all words in the vocabulary, it is quite expensive computationally.

Content

Word2Vec: Skip-gram Model



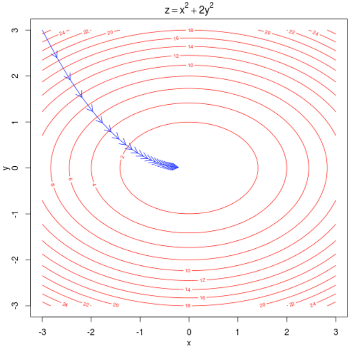
Content

Training a model by optimizing parameters

To train a model, we adjust parameters to minimize a loss

E.g., below, for a simple convex function over two parameters

Contour lines show levels of objective function



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

To train the model: Compute all vector gradients!

- Recall: θ represents all model parameters, in one long vector
- In our case with *d*-dimensional vectors and *V*-many words:

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

- Remember: every word has two vectors
- We optimize these parameters by walking down the gradient

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

Derivation of gradients for Word2Vec model

$$
\begin{aligned}
J(\theta) &= -\frac{1}{T} \log L(\theta) \\
&= -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log p(w_{t+j} | w_t; \theta) \\
&= -\frac{1}{T} \sum_{o \in context(c)} \sum_{c \in corpus} \log p(o|c; u, v) \\
\frac{\partial}{\partial \theta} J(\theta) &= -\frac{1}{T} \sum_{o \in context(c)} \sum_{c \in corpus} \frac{\partial}{\partial \theta} \log p(o|c; u, v)
\end{aligned}
$$

Derivation of gradients for Word2Vec model

$$
\begin{aligned}
\frac{\partial}{\partial v_c} \log p(o|c; u, v) &= \frac{\partial}{\partial v_c} \log \frac{\exp({u_o}^T v_c)}{\sum_{w=1}^{V} \exp({u_w}^T v_c)} \\
&= \frac{\partial}{\partial v_c} ({u_o}^T v_c) - \frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp({u_w}^T v_c) \\
&= u_o - \frac{1}{\sum_{w=1}^{V} \exp({u_w}^T v_c)} \frac{\partial}{\partial v_c} \sum_{w=1}^{V} \exp({u_w}^T v_c) \\
&= u_o - \frac{1}{\sum_{w=1}^{V} \exp({u_w}^T v_c)} \sum_{w=1}^{V} \frac{\partial}{\partial v_c} \exp({u_w}^T v_c) \\
&= u_o - \frac{1}{\sum_{w=1}^{V} \exp({u_w}^T v_c)} \sum_{w=1}^{V} \exp({u_w}^T v_c) u_w \\
&= u_o - \sum_{x=1}^{V} \frac{\exp({u_x}^T v_c)}{\sum_{w=1}^{V} \exp({u_w}^T v_c)} u_x \\
&= u_o - \sum_{x=1}^{V} p(x|c) u_x
\end{aligned}
$$

Derivation of gradients for Word2Vec model

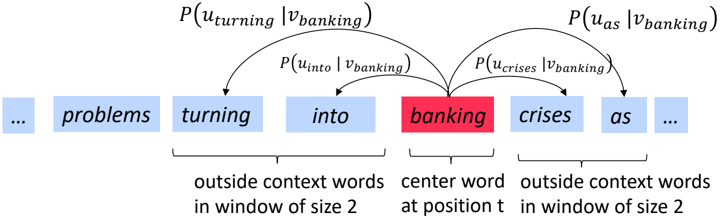
$$\frac{\partial}{\partial\theta}J(\theta) = -\frac{1}{T}\sum_{o\in context(c)}\sum_{c\in corpus}\frac{\partial}{\partial\theta}\log p(o|c;u,v)$$

$$\frac{\partial}{\partial v_c}J(\theta) = -\frac{1}{T}\sum_{o\in context(c)}\sum_{c\in corpus}\left[u_o - \sum_{x=1}^{V} p(x|c)u_x\right]$$

$$\frac{\partial}{\partial u_o}J(\theta) = -\frac{1}{T}\sum_{o\in context(c)}\sum_{c\in corpus}\left[v_c - \sum_{x=1}^{V} p(o|x)v_x\right]$$

Calculating all gradients!

- We went through gradient for each center vector *v* in a window
- We also need gradients for outside vectors *u*
  - Derive at home!
- Generally in each window we will compute updates for all parameters that are being used in that window. For example:

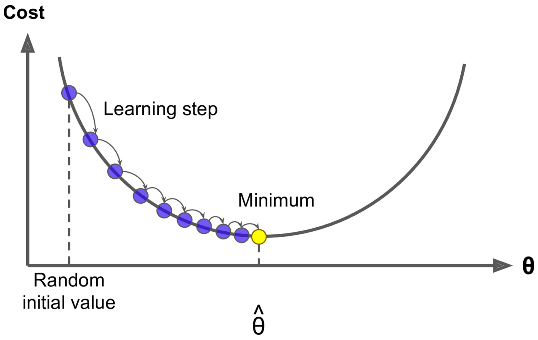


Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Content

5. Optimization: Gradient Descent

- We have a cost function $J(\theta)$ we want to minimize
- **Gradient Descent** is an algorithm to minimize $J(\theta)$
- Idea: for current value of θ, calculate gradient of $J(\theta)$, then take small step in direction of negative gradient. Repeat.



Note: Our objectives may not be convex like this :(

32

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

Gradient Descent

- Update equation (in matrix notation):

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

α = *step size* or *learning rate*

- Update equation (for single parameter):

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

- Algorithm:

```
while True:
    theta_grad = evaluate_gradient(J,corpus,theta)
    theta = theta - alpha * theta_grad
```

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n