Data-analysis and Retrieval
Word Embeddings

Ad Feelders

Universiteit Utrecht

Word Vectors

- Represent words as vectors of real numbers.
- Related words should have similar vectors.
- As measured by their cosine similarity.
- How is this useful?

Application to Home Depot

- Given a query q, return relevant products, based on their descriptions d.
- Quantify how well the query-product pair (q, d) matches.
- Query contains "screw", product description contains "bolt": no match!
- But these words represent strongly related objects.
- Hence, their word vectors v_{screw} and v_{bolt} should be similar.
- Using word vector representations we should be able to compute a better matching score.

Learning word vectors from data

- A word's meaning is determined by the words that frequently appear close-by:

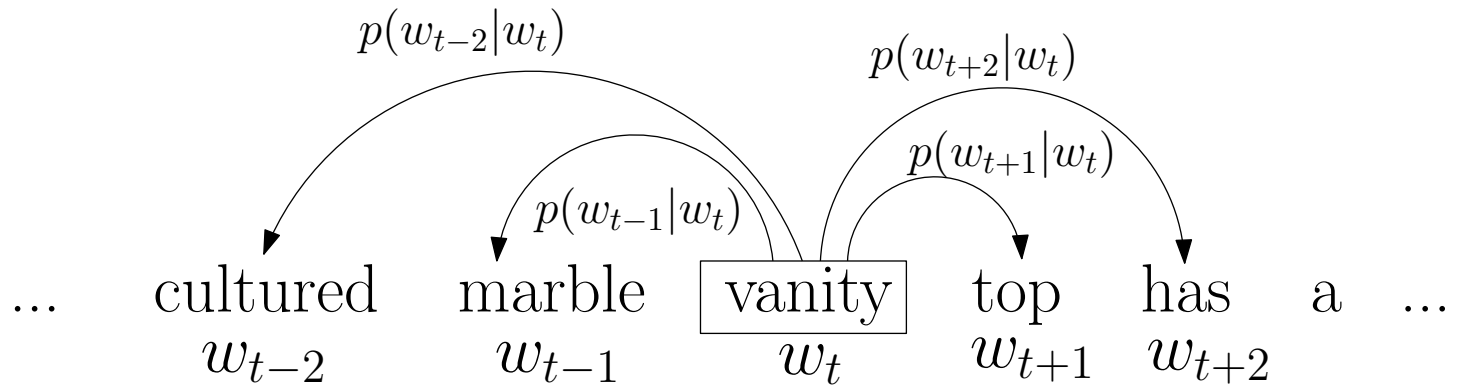
  "You shall know a word by the company it keeps"
  (J. R. Firth).

- When a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
- To learn word vectors from a text corpus, we use the many contexts of w in the corpus to build up a word vector representation of w.

Word2vec skip-gram model

Example sentence from the home depot corpus:

`The cultured marble vanity top has a rectangular integral sink basin to make cleaning easier.`



The center word is boxed. In this example, $m = 2$.

Word2vec skip-gram model

Try to predict words in the context of the center word.

The likelihood-function is:

$$L(\theta) = \prod_{t=1}^{T} \left(\prod_{\substack{-m \leq j \leq m \\ j \neq 0}} p(w_{t+j} \mid w_t; \theta) \right)$$

The negative log-likelihood is given by:

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

Word2vec skip-gram model

In the model there is only one probability for each (outside word, center word) pair o, c:

$$p(o \mid c) = \frac{\exp(u_o^\top v_c)}{\sum_{w=1}^{V} \exp(u_w^\top v_c)} \tag{1}$$

This is the softmax function that we know from multinomial logistic regression.

According to the model, words with similar word vectors (as measured by their dot product) are more likely to appear in each other's context.

Word2vec skip-gram

Training data:

center word (c)	outside word (o)	probability ($p(o \mid c)$)
vanity	cultured	$p(\text{cultured} \mid \text{vanity})$
vanity	marble	$p(\text{marble} \mid \text{vanity})$
vanity	top	$p(\text{top} \mid \text{vanity})$
vanity	has	$p(\text{has} \mid \text{vanity})$
top	marble	$p(\text{marble} \mid \text{top})$
...	...	...

Word2vec skip-gram

- To maximize the likelihood, we should give high probability to events that occur often, and low probability to events that occur seldom.
- So to maximize the likelihood, $p(o \mid c)$ will be given a relatively large value if o often appears in the context of c.
- Likewise, $p(o \mid c)$ will be given a relatively small value if o hardly ever appears in the context of c.
- This is accomplished by making the dot product of their word vectors relatively large, respectively small.

Each word has *two* vector representations; one as an outside word (u), and one as a center word (v).

These are the parameters θ of the model that we have to learn from the data.

Optimization (single variable)

Suppose we want to find the value of x for which the function

$$y = f(x)$$

is minimized (or maximized).

From calculus we know that a necessary condition for a minimum is:

$$\frac{df}{dx} = 0 \tag{2}$$

This condition is not sufficient, since maxima and points of inflection also satisfy equation (2). Together with the second-order condition:

$$\frac{d^2 f}{dx^2} > 0, \tag{3}$$

we have a sufficient condition for a local minimum.

Optimization (single variable)

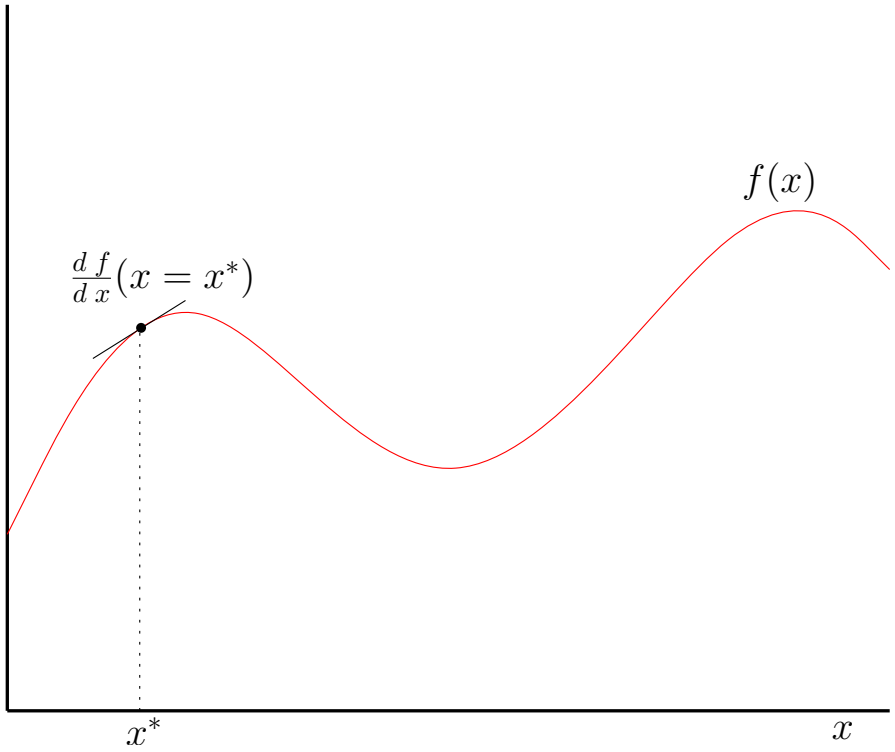
The equation

$$\frac{df}{dx} = 0$$

may not have a closed form solution however.

In such cases we have to resort to iterative numerical procedures such as gradient descent.

Optimization (single variable)



The derivative at $x = x^*$ is positive, so to increase the function value we should increase the value of x, i.e. make a step in the direction of the derivative.

$$\Delta f \approx \frac{df}{dx}(x = x^*)\Delta x$$

Gradient Descent Algorithm (single variable)

The basic *gradient-descent algorithm* is:

1. Set $t = 0$, and *choose* an initial value $x^{(0)}$
2. determine the derivative

$$\frac{df}{dx}(x = x^{(t)})$$

of $f(x)$ at $x^{(t)}$ and *update*

$$x^{(t+1)} = x^{(t)} - \eta \frac{df}{dx}(x = x^{(t)})$$

Set $t = t + 1$.

3. Repeat the previous step until

$$\frac{df}{dx} = 0$$

and *check* if a *(local) minimum* has been reached.

$\eta > 0$ is the *step size* (or *learning rate*).

Optimization (multiple variables)

Suppose we want to find the values of $x_1, \ldots, x_p$ for which the function

$$y = f(x_1, \ldots, x_p)$$

is minimized (or maximized).

Analogous to the single-variable case a necessary condition for a minimum is:

$$\frac{\partial f}{\partial x_j} = 0 \qquad j = 1, \ldots, p \tag{4}$$

Again this condition is not sufficient, since maxima and saddle points also satisfy (4). For the second order condition the Hessian matrix H, should be positive definite.

The Gradient

The gradient ∇f of

$$f(x_1, x_2, \ldots, x_p),$$

is defined to be the vector of partial derivatives

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_p} \end{bmatrix}$$

Gradient of a Linear Function

The gradient of a linear function

$$f(x) = a + \sum_{i=1}^{p} b_i x_i = a + b^\top x$$

is given by

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_p} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix} = b$$

Furthermore, we have:

$$\Delta f = b^\top \Delta x = \nabla f^\top \Delta x$$

In which direction should we move to maximize Δf?

Gradient Descent

We restrict Δx to have unit length. Let's call this vector u (for unit length). So now we have

$$\Delta f = \nabla f^\top u$$

In which direction u does f increase the fastest?

Let α denote the angle between ∇f and u, then (cosine similarity)

$$\cos(\alpha) = \frac{\nabla f^\top u}{\|\nabla f\|\|u\|} = \frac{\nabla f^\top u}{\|\nabla f\|},$$

since $\|u\| = 1$.

Gradient Ascent

Multiplying both sides by $\|\nabla f\|$, we get:

$$\nabla f^\top u = \cos(\alpha)\|\nabla f\|$$

So now we have

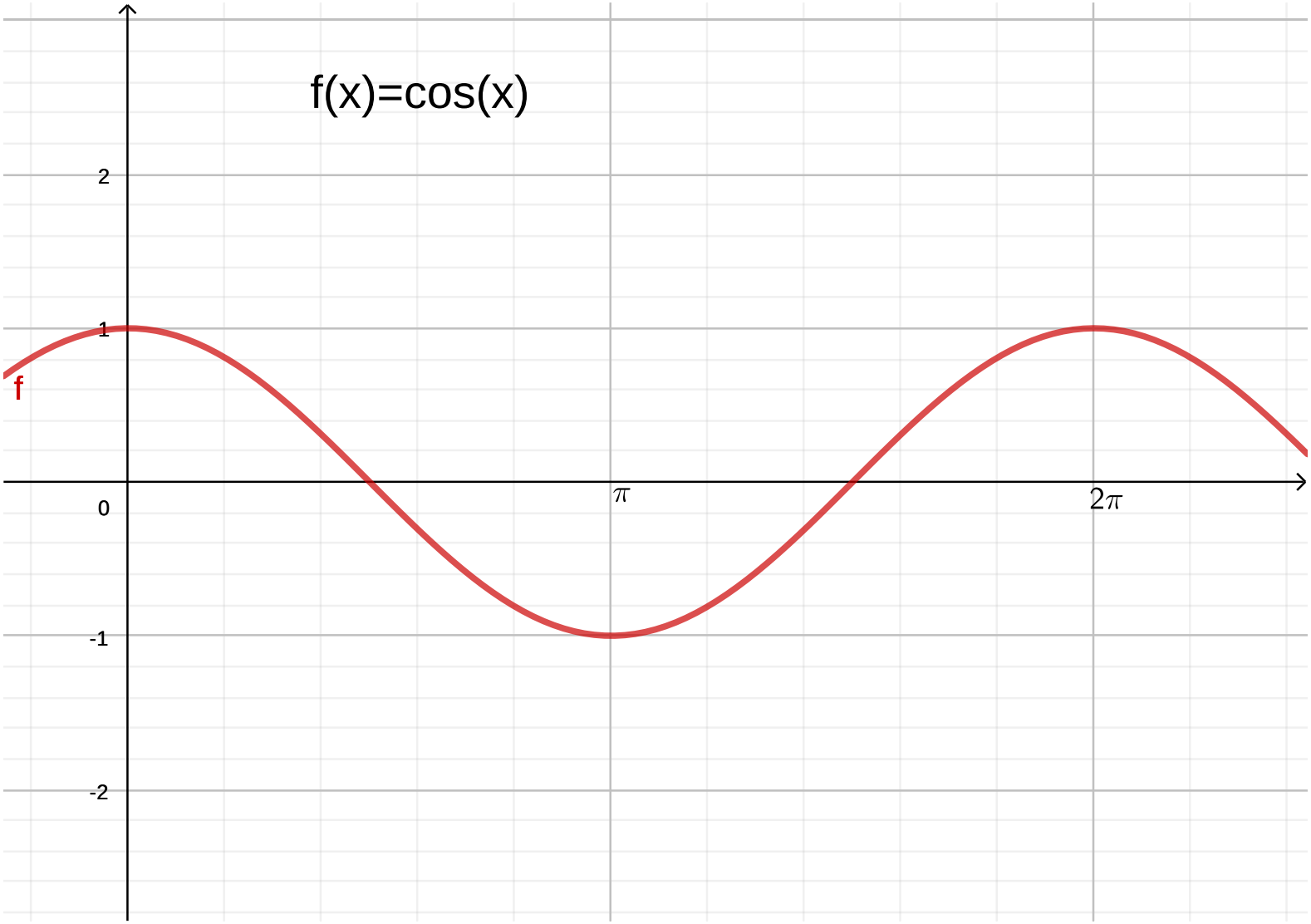
$$\Delta f = \cos(\alpha)\|\nabla f\|$$

$\cos(\alpha)$ achieves its maximum value of $+1$, when the angle α is 0° (0 radians), that is, the two vectors point in the same direction.

Hence, to maximize Δf we should choose u to point in the same direction as the gradient.

The gradient points in the direction of fastest increase of f.

The Cosine Function



Gradient Descent

Likewise, $\cos(\alpha)$ achieves its minimum value of -1, when the angle α is 180° (π radians), that is, the two vectors point in opposite directions.

Hence, to minimize Δf we should choose u to point in the opposite direction of the gradient.

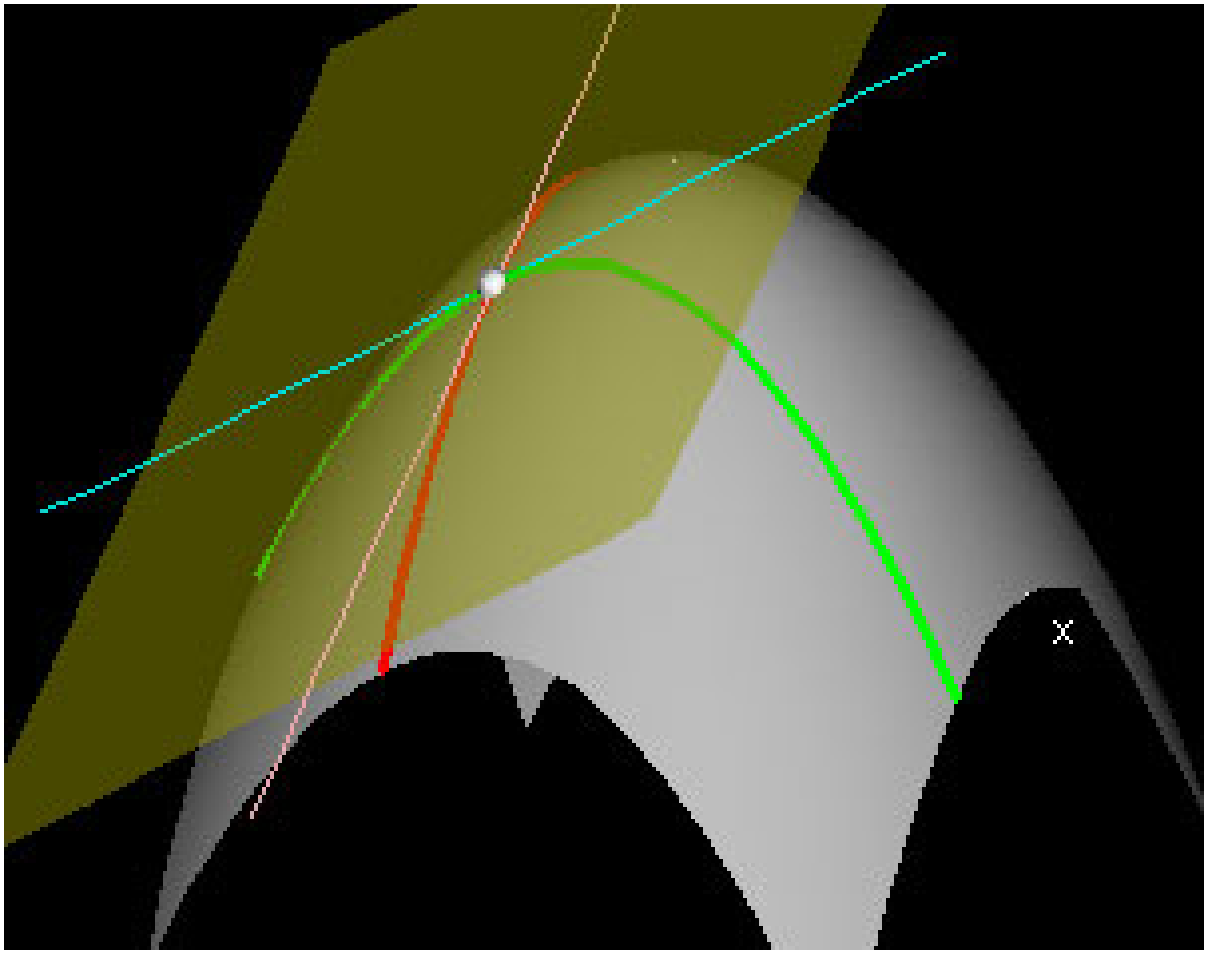
Minus the gradient points in the direction of fastest decrease of f.

Local Linear Approximation

- Thus far we only considered linear functions.
- The functions of interest (e.g. likelihood functions) are non-linear.
- The result can however be applied to arbitrary functions by considering a *local linear approximation* to the function at a point x^*

$$\Delta f \approx \frac{\partial f}{\partial x_1}(x = x^*)\Delta x_1 + \frac{\partial f}{\partial x_2}(x = x^*)\Delta x_2.$$

Local Linear Approximation by Tangent Plane



Gradient Descent Algorithm

The basic *gradient-descent algorithm* is:

1. Set $t = 0$, and *choose* an initial value $x^{(0)}$
2. determine the gradient $\nabla f(x^{(t)})$ of $f(x)$ at $x^{(t)}$ and *update*

$$x^{(t+1)} = x^{(t)} - \eta \nabla f(x^{(t)})$$

   Set $t = t + 1$.
3. Repeat the previous step until

$$\nabla f(x^{(t)}) = 0$$

   and *check* if a *(local) minimum* has been reached.

$\eta > 0$ is the *step size* (or *learning rate*).

Example of gradient descent

Note: b_0 and b_1 are the variables here!

i	x	y	$\hat{y} = b_0 + b_1 x$	$e = y - \hat{y}$
1	0	1	b_0	$1 - b_0$
2	1	3	$b_0 + b_1$	$3 - b_0 - b_1$
3	2	4	$b_0 + 2b_1$	$4 - b_0 - 2b_1$
4	3	3	$b_0 + 3b_1$	$3 - b_0 - 3b_1$
5	4	5	$b_0 + 4b_1$	$5 - b_0 - 4b_1$

$$\begin{aligned} \mathrm{RSS}(b_0, b_1) &= (1 - b_0)^2 + (3 - b_0 - b_1)^2 \\ &\quad + (4 - b_0 - 2b_1)^2 + (3 - b_0 - 3b_1)^2 \\ &\quad + (5 - b_0 - 4b_1)^2 \end{aligned}$$

Example of gradient descent

The gradient is:

$$\nabla \text{RSS} = \begin{bmatrix} \frac{\partial \text{RSS}}{\partial b_0} \\ \frac{\partial \text{RSS}}{\partial b_1} \end{bmatrix} = \begin{bmatrix} -32 + 10b_0 + 20b_1 \\ -80 + 20b_0 + 60b_1 \end{bmatrix}$$

Let $b^{(0)} = (0, 0)$. Then the gradient evaluated in the point $b^{(0)}$ is:

$$\nabla \text{RSS}(b^{(0)}) = \begin{bmatrix} -32 + 10 \times 0 + 20 \times 0 \\ -80 + 20 \times 0 + 60 \times 0 \end{bmatrix} = \begin{bmatrix} -32 \\ -80 \end{bmatrix}$$

Local linear approximation:

$$\Delta RSS \approx -32\Delta b_0 - 80\Delta b_1$$

Example of gradient descent

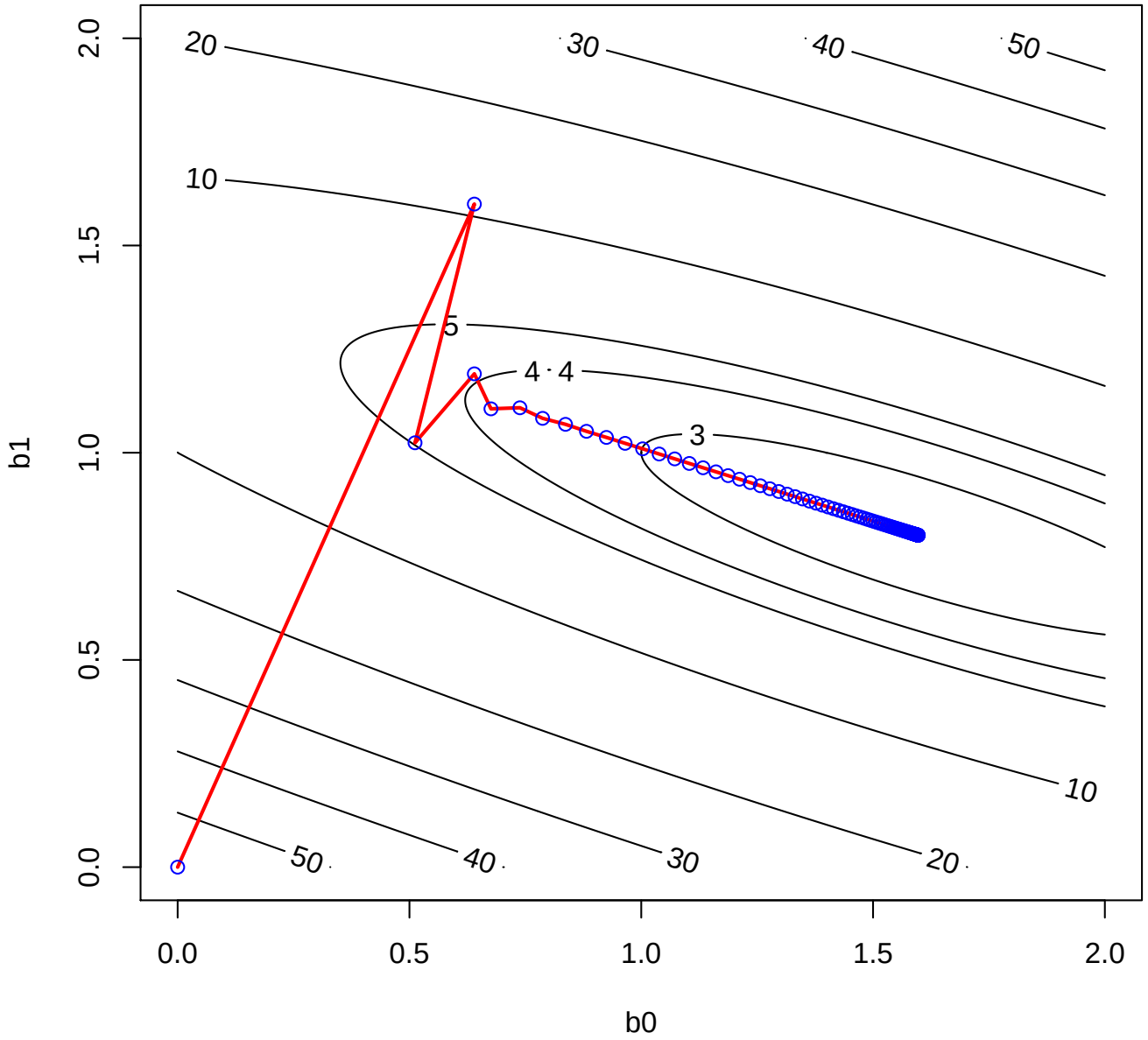
Let $\eta = \frac{1}{50}$. Then we get the following update:

$$b_0^{(1)} = b_0^{(0)} - \eta \frac{\partial \text{RSS}}{\partial b_0} = 0 - \frac{1}{50} \times -32 = 0.64$$
$$b_1^{(1)} = b_1^{(0)} - \eta \frac{\partial \text{RSS}}{\partial b_1} = 0 - \frac{1}{50} \times -80 = 1.6$$

Or all at once:

$$b^{(1)} = b^{(0)} - \eta \frac{\partial \text{RSS}}{\partial b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \frac{1}{50} \begin{bmatrix} -32 \\ -80 \end{bmatrix} = \begin{bmatrix} 0.64 \\ 1.6 \end{bmatrix}$$

Gradient Descent with step size $\eta = 0.02$



GloVe

The GloVe error function is:

$$E(\theta) = \frac{1}{2} \sum_{i,\, j=1}^{V} f(X_{ij})(u_i^\top v_j - \log X_{ij})^2,$$

where X_{ij} denotes the number of times word j occurs in the context of word i, and

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise.} \end{cases}$$

If word i often appears nearby word j, then $\log X_{ij}$ is big, and training will make the dot product between u_i and v_j big as well, in order to match their co-occurrence count.

Hence: words that often appear close to each other will get similar word vectors.

Weighting function

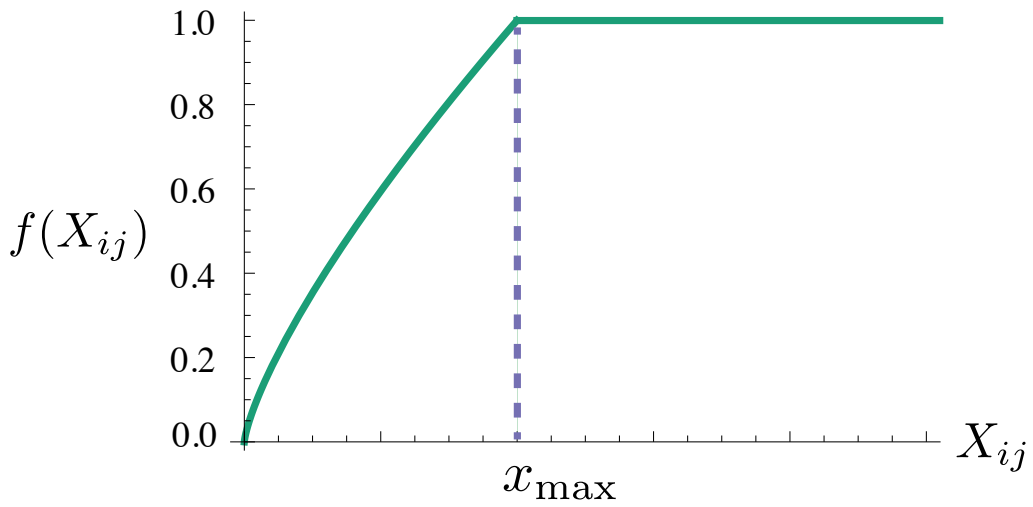


Figure 1: Weighting function f with $\alpha = 3/4$.

GloVe: gradient descent

Let

$$E_{ij} = \frac{1}{2} f(X_{ij})(u_i^\top v_j - \log X_{ij})^2,$$

denote the error corresponding to cell (i, j) of the co-occurrence matrix. The partial derivatives are:

$$\frac{\partial E_{ij}}{\partial u_i} = f(X_{ij})(u_i^\top v_j - \log X_{ij}) v_j$$
$$\frac{\partial E_{ij}}{\partial v_j} = f(X_{ij})(u_i^\top v_j - \log X_{ij}) u_i$$

Gradient descent update step:

$$u_i^{(t+1)} = u_i^{(t)} - \eta \times f(X_{ij})(u_i^{\top(t)} v_j^{(t)} - \log X_{ij}) v_j^{(t)}$$
$$v_j^{(t+1)} = v_j^{(t)} - \eta \times f(X_{ij})(u_i^{\top(t)} v_j^{(t)} - \log X_{ij}) u_i^{(t)}$$

Word analogy task

Answer questions like: a is to b as c is to . . .?

1. Semantic: "Athens is to Greece as Berlin is to . . .?"
2. Syntactic: "Dance is to dancing as fly is to . . .?"

One would desire that

$$v_{\text{Greece}} - v_{\text{Athens}} \approx v_{\text{Germany}} - v_{\text{Berlin}}$$

Find the word d whose word vector v_d is the closest to

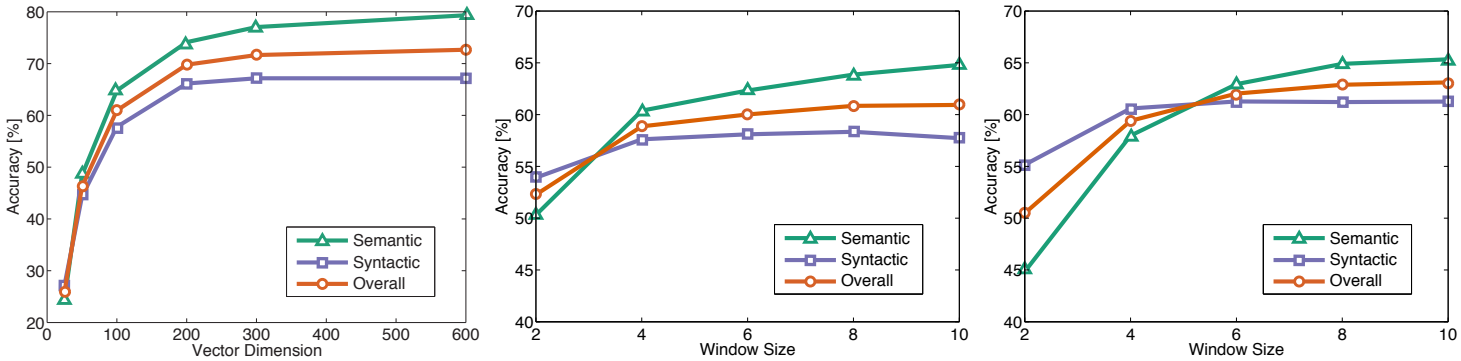
$$v_{\text{Greece}} - v_{\text{Athens}} + v_{\text{Berlin}},$$

according to cosine similarity. Only the exact correspondence $d =$ Germany counts as a correct match.

Accuracy on word analogy task

Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	67.5	54.3	60.3
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	64.8	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	80.8	61.5	70.3
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	67.4	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	**81.9**	**69.3**	**75.0**

Influence of vector dimension and window size



(a) Symmetric context (b) Symmetric context (c) Asymmetric context

Figure 2: Accuracy on the analogy task as function of vector size and window size/type. All models are trained on the 6 billion token corpus. In (a), the window size is 10. In (b) and (c), the vector size is 100.

Influence of corpus size

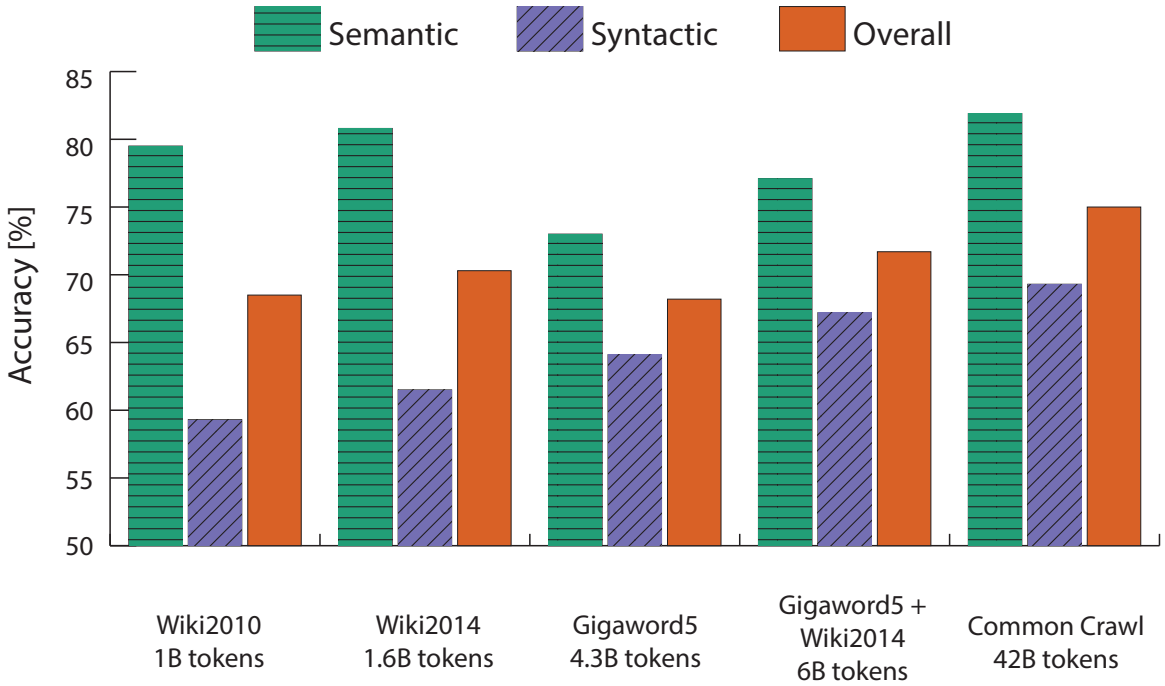


Figure 3: Accuracy on the analogy task for 300-dimensional vectors trained on different corpora.

GloVe: Toy Example

"I like to drink orange juice.
I don't like to drink apple juice.
I like to drink orange juice.
I don't like to drink apple juice.
I like to drink orange juice.
I don't like to drink apple juice.
I like to drink orange juice.
I don't like to drink apple juice.
I like to drink orange juice.
I don't like to drink apple juice.
I like to drink orange juice.
I don't like to drink apple juice.
I like to drink orange juice.
I don't like to drink apple juice.
I like to drink orange juice.
I don't like to drink apple juice."

GloVe: Toy Example

```
> library(text2vec)
> library(tm)
> txt <- scan("C:/DAR 2019/toy.txt",what="character")
> txt <- removePunctuation(txt)
> it = itoken(txt, tolower, word_tokenizer, n_chunks = 10)
> vocab = create_vocabulary(it)

> vocab
Number of docs: 1
0 stopwords:  ...
ngram_min = 1; ngram_max = 1
Vocabulary:
     term term_count doc_count
1:  apple          8         1
2:   dont          8         1
3: orange          8         1
4:  drink         16         1
5:      i         16         1
6:  juice         16         1
7:   like         16         1
8:     to         16         1
```

GloVe: Toy Example

```
> vectorizer = vocab_vectorizer(vocab)
# compute term co-occurrence matrix (tcm) with window size = 2
> tcm = create_tcm(it, vectorizer, skip_grams_window = 2L)
> tcm
8 x 8 sparse Matrix of class "dgTMatrix"
       apple dont orange drink   i juice like to
apple      .    .      .     8 3.5     8  .    4
dont       .    .      .     . 8.0     4  8.0  4
orange     .    .      .     8 4.0     8  .    4
drink      .    .      .     . .       8  8.0 16
i          .    .      .     . .      15 12.0  4
juice      .    .      .     . .       .  3.5  .
like       .    .      .     . .       .  .   16
to         .    .      .     . .       .  .    .
```

GloVe: Toy Example

```
# train the word vectors
> glove = GlobalVectors$new(rank = 3, x_max = 10)
> wv_main = glove$fit_transform(tcm, n_iter = 100,
                    convergence_tol = 0.01, learning_rate=0.001)

> wv_context = glove$components
# add center vectors and context vectors
> word_vectors = wv_main + t(wv_context)
> word_vectors
               [,1]       [,2]       [,3]
apple    0.21094299 -0.9846557  0.5689361
dont    -0.04181674  1.3585625  0.7532957
orange   0.40341986 -0.5874701  0.6021801
drink    0.19814930 -0.8773270  0.3185843
i        0.22016927  0.5890721  1.8129052
juice    0.36769405 -1.2219923  1.8221069
like    -0.05956698  0.4269785 -0.1197119
to       0.16924874 -0.8205703 -0.2006241
```

GloVe: Toy Example

```
# compute cosine similarity between "apple" and other words
> apple = word_vectors["apple", , drop = FALSE]
> cos_sim = sim2(x = word_vectors, y = apple, method = "cosine", norm = "l2")
> cos_sim
            apple
apple   1.0000000
dont   -0.5107285
orange  0.9323933
drink   0.9848749
i       0.2243131
juice   0.9007214
like   -0.9683208
to      0.7321403

> sort(cos_sim[,1], decreasing = TRUE)
     apple      drink     orange      juice         to          i       dont       like
 1.0000000  0.9848749  0.9323933  0.9007214  0.7321403  0.2243131 -0.5107285 -0.9683208
```

Note that "orange" is very similar to "apple", even though they never appear in each other's context!

Application to Home Depot data

```
# load libraries
> library(text2vec)
> library(tm)
# read product descriptions
> product.dat <- read.csv("C:/DAR 2019/
   product_descriptions.csv", stringsAsFactors=FALSE)
# select product descriptions
> txt <- product.dat[,2]

# remove punctuation
> txt <- removePunctuation(txt)
# convert letters to lower case
> txt <- tolower(txt)
```

Application to Home Depot data

```
# txt is a vector of type chr (= string)
> str(txt)
 chr [1:124428] "not only do angles make joints stronger
                 they also provide ..."
> length(txt)
[1] 124428
> txt[1]
[1] "not only do angles make joints stronger ..."

> it = itoken(txt, tolower, word_tokenizer, n_chunks = 10)

# extract vocabulary from product descriptions
> vocab = create_vocabulary(it)
# how many "words"?
> dim(vocab)
[1] 364758      3
```

Application to Home Depot data

```
# reduce the size of the vocabulary
> vocab = prune_vocabulary(vocab, term_count_min = 10,
    doc_proportion_max = 0.8,doc_proportion_min = 0.001,
    vocab_term_max = 20000)

# how many words are left?
> dim(vocab)
[1] 6563    3

# show the first three

> vocab[1:3,]
Number of docs: 124428
0 stopwords:  ...
ngram_min = 1; ngram_max = 1
Vocabulary:
   term term_count doc_count
1: with     250258     98674
2:   of     229820     91535
3:   is     192208     87733
```

Application to Home Depot data

```
> vectorizer = vocab_vectorizer(vocab)
# create term co-occurrence matrix with window-size 5
> tcm = create_tcm(it, vectorizer, skip_grams_window = 5L)
# train the word vectors
> glove = GlobalVectors$new(rank = 50, x_max = 10)
> wv_main = glove$fit_transform(tcm, n_iter = 50,
    convergence_tol = 0.01)

# add center word vectors and context word vectors (u and v)
> wv_context = glove$components
> word_vectors = wv_main + t(wv_context)
```

Application to Home Depot data

```
# show the first 5 components of the first 3 word vectors
# each word vector has 50 components (as specified in training)

> word_vectors[1:3,1:5]
           [,1]       [,2]        [,3]       [,4]        [,5]
with  0.1572956 -0.4986556 -0.30388024 -0.5208205 -0.49814761
of    0.5462567 -0.3089656 -0.40487972  0.3400864 -0.16796595
is   -0.3926085 -0.9640168  0.08781935  0.3989814 -0.08923554

# select the word vector of "screw"
> screw = word_vectors["screw", , drop = FALSE]
# compute its cosine similarity with all other word vectors
> cos_sim = sim2(x = word_vectors, y = screw, method = "cosine",
                 norm = "l2")
# show the 5 closest words
> head(sort(cos_sim[,1], decreasing = TRUE), 5)
    screw    screws      head       hex     bolts
1.0000000 0.7952427 0.6618135 0.6532096 0.6130304
```

Literature

Required:

- Chapter 6 of the book Speech and Language Processing by Jurafsky en Martin.

Additional:

- Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. `https://nlp.stanford.edu/projects/glove/`
- Text2vec R package: `http://text2vec.org/`