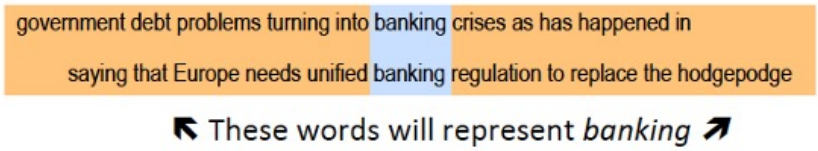
1.4.3 Word Embeddings

The neighborhood of a word expresses a lot about its meaning

- "You shall know a word by the company it keeps" (J. R. Firth 1957)



↖ These words will represent *banking* ↗

With latent semantic indexing we have exploited the idea that words that occur together in a document are likely to have some shared meaning. The idea that words occurring in a common context have a shared meaning is quite old and has been already stated by Firth in 1957.

Different to LSI we will in the following exploit this idea in different way. Instead of considering a complete document as the context of a word, we will consider only the immediate neighborhood of a word, in the spirit of the statement of Firth.

Word Context

government debt problems turning into banking crises as has happened in

saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Context: words in a window of size s, e.g., s = 2

Example:

- Center word: w = banking
- Context 1: $C_1(w)$ = {turning, into, crises, as}
- Context 2: $C_2(w)$ = {needs, unified, regulation, to}

Word-Context occurrence: $(w, c), c \in C_i(w)$

For a given word w we can identify its neighboring words, which will call its context and denote by C(w). The context can be determined by choosing a window of preceding and succeeding words. A typical window size used in practice would by n = 5. We will call the word w in the following center word and a word c a context word.

Since a word can occur multiple times, each occurrence of the word induces a new context. We will denote these contexts by $C_i(w)$ with an index i running over all occurrences of a word in a document collection.

Similarity-Based Representation

One of the most successful ideas in natural language processing

Two words are considered as similar,
when they have similar contexts

- Context captures both syntactic and semantic similarity
  - Syntactic: e.g., king – kings
  - Semantic: e.g., king – queen
- Implicitly used with the term-document matrix M
  - Less localized context
  - No distinction between roles of context and word

The idea of context representing meaning is one of the most successful ideas in natural language processing. The neighborhood captures not only semantic relationships among words (as would co-occurrence in the same document). It, at the same time, also captured syntactic relationships. This is an important difference compared to methods based on document-level co-occurrence like LSI.

A second important difference is that methods based on this idea distinguish between a word occurring as the center word of a context and as a context word. For example, it is very unlikely the word “king” would have in its context a second occurrence of the word “king”. Thus, king as a “context word” needs to be treated differently from king as a center word. This distinction is not made in approaches based exclusively on co-occurrence statistics at the document level.

Idea: Word Embeddings

Model how likely a word and a context occur together

Approach:

- Map words into a low-dimensional space (e.g., d = 200)
- Map context words into the same low-dimensional space
  - A different mapping for the same words
- Interpret the vector distance (product) as a measure for how likely the word and its context occur together

Due to projection in low-dimensional space, semantically and syntactically related words and contexts should be close

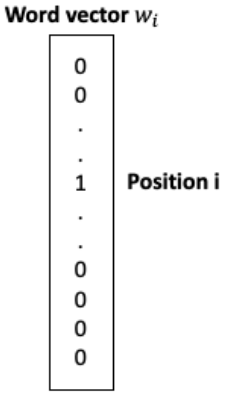
The basic idea for word embeddings is straightforward. Both, center words and context words, are mapped into a low-dimensional space of the same dimension. Their representations in that space should be similar, if the center word and the context word occur frequently together. Thus, we capture the information on word context in a low-dimensional representation. Using dimensionality reduction, the expectation is that words and contexts that play similar roles would also be close to each other and would capture syntactic and semantic similarity.

Dimensionality reduction is helpful, since vocabularies can become very large. This is especially true when not only words, but also short phrases are considered. Thus, data would be very sparse in the original vocabulary space, and it would be difficult to represent similarity.

Mapping to Concept Space

Vocabulary T of size m

Words $w_i \in T$ are encoded as "1-hot vectors" of length m, i.e., the component at position i is 1 all others are 0



We introduce now the representations of center words and context words in an m-dimensional space. The representation where the i-th component of a vector is set to 1 for the i-th word in the vocabulary is called often the 1-hot encoding of the word.

Model Parameters

Dimension of concept space d

The mapping to concept space is represented by matrices $W^{(w)}$ and $W^{(c)}$ of dimension $d \times m$

Mapping a word w_i

$\boldsymbol{w_i} = W^{(w)} w_i$ $\quad$ $\boldsymbol{w_i}$ is the column i in $W^{(w)}$

Mapping a context word c_i

$\boldsymbol{c_i} = W^{(c)} w_i$ $\quad$ $\boldsymbol{c_i}$ is the column i in $W^{(c)}$

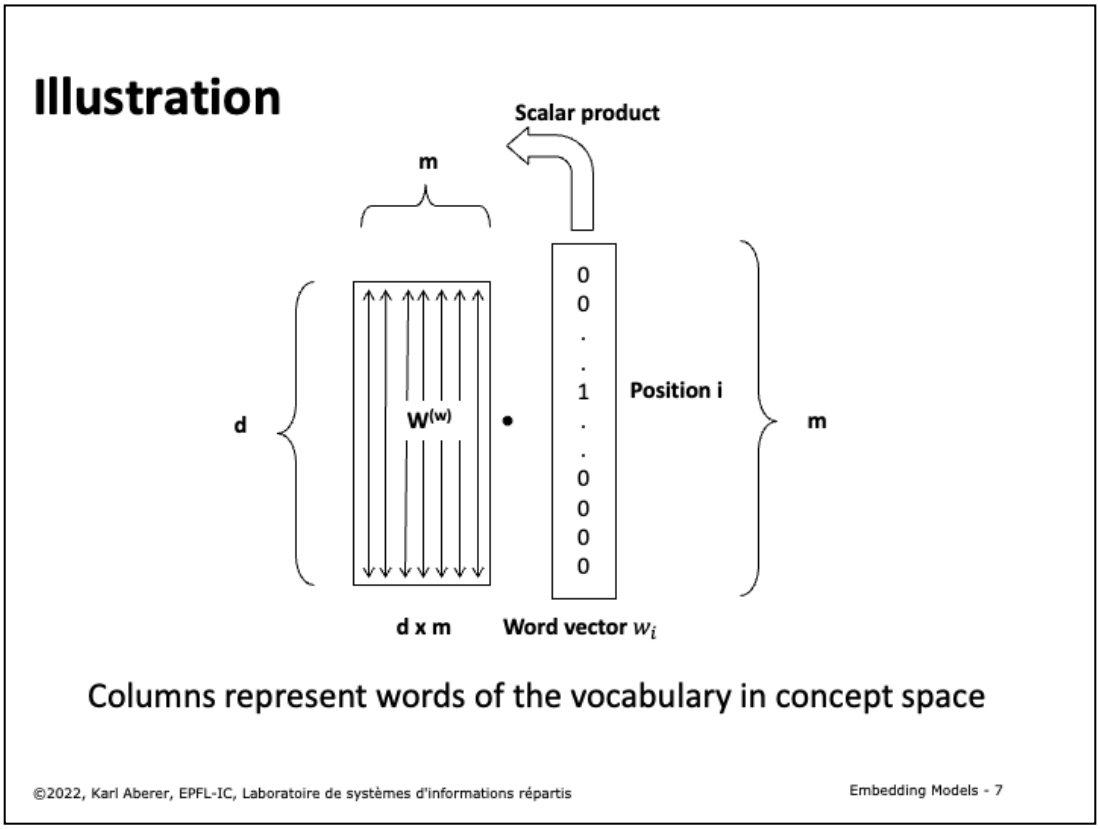
Parameters of model θ : All coefficients of $W^{(w)}$ and $W^{(c)}$

For creating a word embedding we first choose a dimension that is significantly lower than the size of the vocabulary. Typical values of the dimension are in the hundreds, whereas vocabularies can have millions of terms.

In order to map a 1-hot vector from the vocabulary space into the word embedding space, we apply linear mappings represented by a matrices $W^{(w)}$ and $W^{(c)}$. As mentioned before. the mappings for center words and context words are different.

Note that the i-th columns of the matrices correspond exactly to the word embedding representations of the i-th word in the vocabulary.

For notational convenience we will denote in the following the combined set of coefficients of the two matrices also by θ.



Here we illustrate the mapping of words into concept space graphically. Since the word vectors use the 1-hot encoding, we observe that the columns of the matrix $W^{(w)}$ in fact are exactly the representation of the center words in the concept space. The same holds also for the mapping of context words.

Question

A row of matrix $W^{(c)}$ represents

1. How relevant word c is for each dimension
2. How often a context word c co-occurs with all words
3. A representation of word c in concept space

Learning the Model from Data

Given a document collection, how to obtain the parameters θ?

Formulate a classification problem

- Given a word-context pair, predict whether it occurs in the document collection

The problem we face now, is how to determine the model parameters, given a document collection.

We can approach this by introducing learning problem. We want to be able to predict whether a given word-context pair occurs in the document collection. By solving this problem, we will obtain the embedding parameters.

Statistical Learning

We aim to learn a function

$$f: S \rightarrow R$$

We have training data:

samples $f(s_1), f(s_2), \ldots, f(s_t)$

We define a parametrized function (model)

$$f_\theta: S \rightarrow R$$

Learning is obtaining good parameters θ

Before we introduce in detail, how to obtain the parameters for the word embedding, let's first recall the principles of the approach we are going to use, statistical learning.

The general setting of statistical learning is to obtain an approximation of a given function f, from some sample values of the function that are known. The function f is approximated by a function f_θ that has a given form and contains a set of parameters θ. By optimizing the parameters θ we obtain the function approximation. The optimization step is what is generally called learning.

A standard example of statistical learning is linear regression.

Word Context Learning Problem

Domain S	$S = \{(w,c) \mid w,c) \in T \times T\}$ w, c are word-context pairs
$f: S \rightarrow R$	$f(w,c) = P(w,c)$ the probability for a given word-context pair to occur, thus $R = [0,1]$
$f(s_1), f(s_2), \ldots, f(s_t)$	$f(w,c) = 1$ iff (w,c) occurs in text, otherwise $f(w,c) = 0$
$f_\theta(w,c) = P_\theta(w,c)$	approximation of $P(w,c)$

Given that we want to predict whether a word-context pair occurs in a document collection, the statistical learning problem we formulate is for a function that takes as input word-context pairs and returns as output a probability for that this pair occurs in the documents. The samples from which we learn are derived from the document collection. For word-context pairs that occur in the collection we set the function value to 1, and for any pair that does not occur, we can set it to 0. The problem is then to find a function f_θ that approximates well the original probability distribution P. We have now to determine what form the function f_θ can take, and how the quality of approximation is measured.

Representation Learning

A learning algorithm that creates for some input data a new representation with desirable properties

- Most often the representation is in a vector space
- For example, represent words of a vocabulary as vectors

Idea: create the word embeddings using representation learning

So far, the idea of using statistical learning does not explain how we would obtain word embedding representations. To this end, we introduce now another concept, which is called representation learning. Representation learning aims at creating a new representation of input data, into a space of interest and with useful properties. Most often, the space for representation is a vector space.

In order to obtain the representations as a by-product of solving a statistical learning problem, the “trick” is to create a function f_θ that is based on the desired representations of the input data, in our case on the vector representations of the words. As a result, the “real objective” of solving the previous learning problem is no longer doing good predictions for the function f, but obtaining the vector representations.

Word Embedding Learning Problem

Specify the form of the function f_θ

$$f_\theta(w, c) = \boldsymbol{f}(\boldsymbol{w}, \boldsymbol{c}) = P_\theta(w, c)$$

$\boldsymbol{w}, \boldsymbol{c}$ are the embedding vectors of w, c
$\boldsymbol{w} = W^{(w)}w, \boldsymbol{c} = W^{(c)}c$

$\boldsymbol{f}$ is some function that takes as input the embedding vectors

The function f_θ we learn has a specific form: it first maps the words into a vector space, their representation, and then applies another function $\boldsymbol{f}$ for approximating the probabilities.

Note that the function$\boldsymbol{f}$ is using as input the vector representations of the words. The matrices $W^{(w)}$ and $W^{(c)}$ are therefore part of the parameters of the function f_θ.

Properties of Embeddings

The function we want to learn, is the probability $P_\theta(w, c)$ for the word w to appear with the context word c

Derive an approximation of this probability from the representations

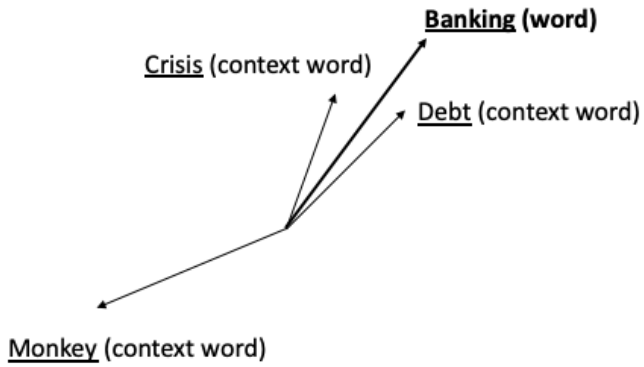
Idea for $\boldsymbol{f}$

- require that if w and c are likely to co-occur their representations $\boldsymbol{w}$ and $\boldsymbol{c}$ should be similar, i.e., the scalar product $\boldsymbol{w} \cdot \boldsymbol{c}$ should be large

So far, we have not imposed any specific constraint on the function $\boldsymbol{f}$ that is applied to the embedding vectors. Now we come back to the requirement that low-dimensional representations of words should be similar, if they have a close semantic or syntactic relationship. If consider co-occurrence as a word-context pair as an indicator for such a relationship, we could also state the related requirement that the low-dimensional representations of center words and context words should be similar. This is the approach we will consider in the following.

For determining similarity of vector representations, we can rely on a well-known approach, the use of the scalar product.

Illustration



Banking • Crisis = +3
Banking • Debt= +2
Banking • Monkey = -1

Here we illustrate the embedding of words and context words in the same low-dimensional space. The idea is that the context words that typically co-occur together with a word, have similar embedding vectors, whereas others (like monkey, which is rarely occurring together with bank) have very different ones.

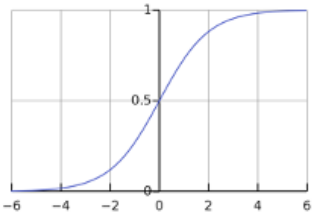
Deriving Probabilities

Normalized scalar product?

- Produces values in [-1,1]

Using the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

$$f_\theta(w,c) = P_\theta(w,c) = \boldsymbol{f}(\boldsymbol{w},\boldsymbol{c}) = \sigma(\boldsymbol{c}\cdot\boldsymbol{w}) = \frac{1}{1+e^{-\boldsymbol{c}\cdot\boldsymbol{w}}}$$

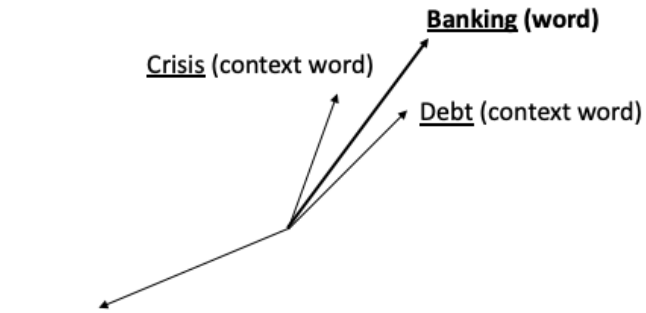


Since we have formulated the learning problem as a prediction of probabilities, the range of values of the learnt function f_θ should fall into the interval [0,1]. Using the scalar product, even if we use normalize by using cosine similarity, does not achieve this.

Therefore, a common function to map real values into the interval [0,1] is used, the sigmoid function. The sigmoid function has other useful properties, in particular, it is differentiable.

This results in the final structure of the function f_θ we learn. It is fully specified by its parameters θ.

Illustration



Banking • Crisis = +3
Banking • Debt= +2
Banking • Monkey = -1

P(Banking, Crisis) = 0.95
P(Banking, Debt= 0.88
P(Banking, Monkey) = 0.27

```
import numpy as np
def sigmoid(z):
    return 1/(1 + np.exp(-z))
[sigmoid(3),sigmoid(2),sigmoid(-1)]
```

```
[0.9525741268224334, 0.8807970779778823, 0.2689414213699951]
```


Here we illustrate how by applying the softmax function to the scalar product of the embedding vectors results in probabilities predicting the occurrence of word-context pairs.

Question

From which data samples the embeddings are learnt?

1. Known embeddings for (w,c) pairs
2. Frequency of occurrences of (w,c) pairs in the document collection
3. Approximate probabilities of occurrences of (w,c) pairs
4. Presence or absence of (w,c) pairs in the document collection

Question

Which of the following functions is not equal to the three others?

1. $f(w, c)$
2. $f_\theta(w, c)$
3. $\boldsymbol{f(w, c)}$
4. $\sigma(\boldsymbol{c} \cdot \boldsymbol{w})$

Learning the Parameters

Assume we have positive examples D for (w, c), as well as negative examples $\widetilde{D}$ for (w, c) that are not occurring in the document collection

Maximizing the overall probabilities

$$\theta = \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P_\theta(w,c) \prod_{(w,c)\in \widetilde{D}} \big(1 - P_\theta(w,c)\big) =$$
$$\underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \sigma(\boldsymbol{c}\cdot\boldsymbol{w}) + \sum_{(w,c)\in \widetilde{D}} \log \sigma(-\boldsymbol{c}\cdot\boldsymbol{w})$$

Now that we have defined the structure of the function we want to learn, we must perform the learning. To that end, we formulate an optimization problem which states that learning consists of finding the optimal parameters θ for achieving the highest predicted probability for the available data. The available data consists of samples of word-context pairs. There exist two types of such samples, first, word-context pairs that occur in the text collection and thus must have a high probability to occur, and word-context pairs that do not occur in the document collection and thus must have a low probability. We cannot assume that we achieve perfect prediction of those probabilities, i.e., values of 0 and 1, under the specific choice of the function P_θ we decided to learn. Therefore, we try to maximize the product of the probabilities of the positive samples and the complement of the probabilities of the negative samples.

The optimization problem can be reformulated by taking the logarithm of the expression to be optimized. This is a standard operation to make the optimization problem easier to treat from an algebraic and numerical perspective.

Positive and Negative Samples

Let D be the set of all word-context occurrences

$$(w_t, c_t) \in D, t = 1, .., s, |D| = s$$

For each (w_t, c_t) define a set $P_n(w_t)$ of K negative samples, i.e., word-context pairs not occurring in the collection

Here we describe the approach for obtaining the positive and negative samples. We consider the set D to be the set of all word-context occurrences in the document collection. For each element of D we then choose a set of negative samples of a given size s. This set consists of word-context pairs that do not occur in the collection. We will describe a concrete approach for selecting such negative samples later.

Obtaining Negative Samples

Negative samples are taken from $P_n(w) = V \setminus C(w)$

Empirical approach

- If p_c is the probability of a candidate context word c in collection, choose the word with probability ${p_c}^b$ with $b < 1$, e.g., $b = 0.75$
- Less frequent words are sampled more often
- Practically: approximate the probability by sampling a few non-context words

Example

- If $p_c = 0.9$, then ${p_c}^{0.75} = 0.92$
- If $p_c = 0.01$, then ${p_c}^{0.75} = 0.032$, boosted by factor 3

A concrete method to obtain the negative samples is described now. For a given word w from the vocabulary V, the context words for negative samples are chosen from all words that do not occur as context word of w. This could be done, for example, uniformly at random. However, with such choice high frequency words would have too much influence on the learning process, whereas low- frequency words would not be sufficiently considered. Therefore, based on empirical evaluation, it turns out that it is of advantage to favor low frequency words. The probability to choose a context word is thus biased towards low frequency words, by attenuating higher probabilities with an exponent $b < 1$. In this way, less frequent words will be considered more frequently.

In order to apply this approach, the word frequencies in the collection of words not occurring in the context need to be known. Since computing this statistics for every word separately would be expensive, in practice it is obtained by sampling.

Loss Function

Maximizing the overall probabilities is equivalent to minimizing the following function

$$J(\theta) = \frac{1}{s}\sum_{t=1}^{s} J_t(\theta)$$

$$J_t(\theta) = -\log \sigma(\boldsymbol{c_t} \cdot \boldsymbol{w_t}) - \sum_{(w_t, c_k) \in P_n(w_t)} \log \sigma(-\boldsymbol{c_k} \cdot \boldsymbol{w_t})$$

$J(\theta)$ is a **loss function**

- A function that measures how well the model fits the samples (training data)
- Learning is applying an algorithm that minimizes this loss function

Having defined the set of positive and negative samples, we can further reformulate the optimization problem for learning the parameters θ. We convert the maximization problem to a minimization problem, which is the conventional formulation of learning problems, and replace the specific choice of samples. This allows to decompose the function to be optimized into a sum of functions $J_t(\theta)$, corresponding to the positive samples (w_t, c_t). This decomposition will be important in the following for the method used to solve the optimization problem.

According to the standard terminology in machine learning, $J(\theta)$ is called the loss function, and is a function that needs to be minimized in order to solve the optimization resp. learning problem.

Skipgram Model with Negative Sampling

The approach to derive word embeddings described is called the skipgram model

- $\boldsymbol{w} = W^{(w)}w, \boldsymbol{c} = W^{(c)}c$, embedding vectors
- $P_\theta(w, c)$, structure of function to represent probabilities of word-context occurrences
- $J(\theta)$, definition of loss function
- $P_n(w)$, choice of negative samples

The different elements for defining an optimization problem to learn the embedding vectors that we have introduced correspond to one specific word embedding model which is called the skipgram model with negative sampling. Note the several choices that are made, in particular on the structure of the probability function, the specific loss function and the choice of negative samples.

Observations

1. The function P_θ can be learnt from a document corpus
2. We can apply optimization to learn the function
3. As we learn the parameters θ, implicitly by learning the function we learn a representation for words (the word embedding vectors)

We have not set up a formulation of an optimization problem that can be solved for any given document collection. By solving the optimization problem, we will obtain as a side effect the embedding vectors. This, in fact, has been the initial objective of defining the learning problem. What remains to be seen is of how concretely the optimization problem can be solved.

Question

With negative sampling a set of negative samples is created for

1. For each word of the vocabulary
2. For each word-context pair
3. For each occurrence of a word in the text
4. For each occurrence of a word-context pair in the text

Question

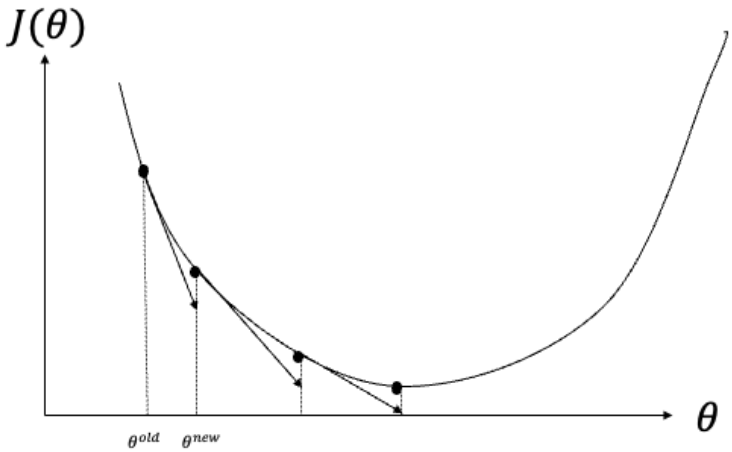
The loss function is minimized

1. By modifying the word embedding vectors
2. By changing the sampling strategy for negative samples
3. By carefully choosing the positive samples
4. By sampling non-frequent word-context pairs more frequently

Solving the Learning Problem

Gradient Descent: compute using all samples

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

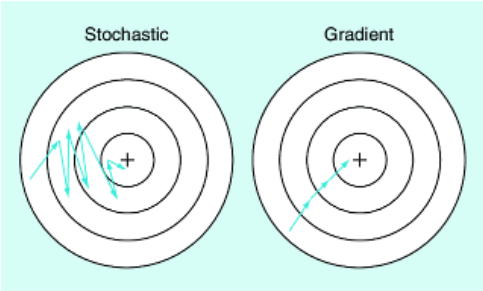


Given the loss function, finding the optimal parameters θ can then be achieved by applying standard methods from machine learning. Concretely, θ can be determined using gradient descent, a standard method for searching the minima of a function using derivatives. Gradient descent is an iterative approach for locally improving a current minimum. By following the local gradient, the optimal value is incrementally updated till it converges to a (local) minimum.

Stochastic Gradient Descent (SGD)

For every $(w_t, c_t) \in D$, update θ separately

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



In basic gradient descent, the parameters are updated based on the complete data (set of samples). In stochastic gradient descent, the parameters are updated for a randomly selected subset of the data. Therefore, the decomposition of the loss function into component functions corresponding to each element of the data was useful. In our case we update the parameters for each word-context pair separately. With stochastic gradient we reduce the cost of each iteration, but on the other hand slow down the convergence rate. In practice, using SGD is the standard method of learning with complex loss functions.

Computing the Derivatives

$$J_t(\theta) = -\log\big(\sigma(\boldsymbol{c}_t \cdot \boldsymbol{w}_t)\big) - \sum_{(w_t, c_k) \in P_n(w_t)} \log \sigma(-\boldsymbol{c}_k \cdot \boldsymbol{w}_t)$$

with $\theta = \boldsymbol{w}_1, \ldots, \boldsymbol{w}_m, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_m$

$$\nabla_\theta J_t(\theta) = \left(\frac{\partial}{\partial \boldsymbol{w}_1} J_t, \ldots \frac{\partial}{\partial \boldsymbol{w}_m} J_t, \frac{\partial}{\partial \boldsymbol{c}_1} J_t, \ldots \frac{\partial}{\partial \boldsymbol{c}_m} J_t\right)$$

then $\frac{\partial}{\partial \boldsymbol{w}_i} J_t = 0$ and $\frac{\partial}{\partial \boldsymbol{c}_i} J_t = 0$, if $\boldsymbol{w}_i \neq \boldsymbol{w}_t$ and $\boldsymbol{c}_i \neq \boldsymbol{c}_t, \boldsymbol{c}_k$

When stochastic gradient descent is used, only columns of the matrices θ that contain the word, or a context word related to the current word-context pair and its negative samples need to be updated. This implies that the data must be organized that these rows can be efficiently accessed (e.g., using hashing).

Updating the Model Parameters

For J_t updates only affect columns in $W^{(w)}$ and $W^{(c)}$ that correspond to $\boldsymbol{w}_t$, $\boldsymbol{c}_t$ and $\boldsymbol{c}_k$,

And the updates are according to the corresponding partial derivative

e.g.,

$$\boldsymbol{w}_t^{new} = \boldsymbol{w}_t^{old} - \alpha \frac{\partial}{\partial \boldsymbol{w}_t} J_t(\boldsymbol{w}_t^{old}, \boldsymbol{c}_t^{old}, \boldsymbol{c}_1^{old}, \dots, \boldsymbol{c}_K^{old})$$

The embedding vectors of a word are affected only by the derivatives of the loss function with respect to the word vector.

Computing the Derivative (Backpropagation)

Some notation

$$x = \mathbf{c} \cdot \boldsymbol{w}, \mathrm{p} = \sigma(x), z_k = -\mathbf{c}_k \cdot \boldsymbol{w}, q_k = \sigma(z_k), |P_n(w)| = K$$

Then $$J_t(\boldsymbol{w}, \mathbf{c}, \mathbf{c}_1, \ldots, \mathbf{c}_k) = -\log(p) - \sum_{k=1}^{K} \log q_k$$

$$\frac{\partial J_t}{\partial \boldsymbol{w}} = \frac{\partial J_t}{\partial p}\frac{\partial p}{\partial x}\frac{\partial x}{\partial \boldsymbol{w}} + \sum_{k=1}^{K} \frac{\partial J_t}{\partial q_k}\frac{\partial q_k}{\partial z_k}\frac{\partial z_k}{\partial \boldsymbol{w}}$$

$$\frac{\partial J_t}{\partial p} = -\frac{1}{p}, \frac{\partial p}{\partial x} = \mathrm{p}(1-\mathrm{p}), \frac{\partial x}{\partial \boldsymbol{w}} = \mathbf{c}$$

Finally $$\frac{\partial J_t}{\partial \boldsymbol{w}} = -(1-\mathrm{p})\mathbf{c} + \sum_{k=1}^{K}(1-q_k)\,\mathbf{c}_k$$

Expensive; cause it is on vector

Computing the derivative of the loss function, for updating the model parameters, requires evaluating derivatives along the operators that compose the loss function. This step is part of what is in machine learning commonly called backpropagation. We illustrate the essential steps of this computation for computing the derivative for a given word $\frac{\partial J_t}{\partial \boldsymbol{w}}$. To that end, it is helpful to identify the different components that constitute the loss function and compose then the derivatives of those components using the chain rule. This reveals that the updates to the parameters result in simple operations. As an exercise one can complete the computation for the other derivatives.

.

Regularization

Regularization: limit model complexity

- In order to avoid overfitting
- In order the model to generalize well to new data

Possible approaches

- Limit the complexity of the model (e.g., linear)
- Add a regularization term to the loss function

Example:

$$J(\theta) = \sum_{t=1}^{s} \left(f(s_t) - f_\theta(s_t)\right)^2 + \gamma|\theta|_2$$

One key problem in statistical learning (or machine learning) is overfitting. If we provide a large number of parameters θ we can easily learn a good, or even perfect approximation of the function applied to the training data, but the function will perform poorly on new data. Therefore, the idea is to “limit” the amount of information that can be stored in the model parameters, by minimizing some measure on them by including it into the loss function. This is called regularization. In the method we have introduced for deriving word embeddings no regularization is applied. The reason is that a linear model is nto considered to be a complex model and therefore the risk of overfitting is limited.

Question

A word embedding for given corpus …

1. depends only on the dimension d
2. depends on the dimension d and number of iterations in gradient descent
3. depends on the dimension d, number of iterations and chosen negative samples
4. there are further factors on which it depends

CBOW Model

Consider a pair $(w, \{c_1, \dots, c_n\})$, does it origin from the data?

Compute $\bar{\boldsymbol{c}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{c_i}$

$$P_\theta(w, \{c_1, \dots, c_n\}) = \frac{1}{1 + e^{-\bar{\boldsymbol{c}} \cdot \boldsymbol{w}}} = \sigma(\bar{\boldsymbol{c}} \cdot \boldsymbol{w})$$

Observation: CBOW produces better results for frequent terms, skipgram for rare terms

With the skipgram model, we have considered the probability of word-context word pairs as the objective for learning. An alternative approach is to consider the complete context of a word and predict whether a set of context words is likely to occur with a given word. In order to represent a set of words in the embedding model we can choose the mean of the embedding vectors of the context words and require that this vector is similar to the embedding vector of center word, if this specific selection of context words appears with the center word in the document collection. The remaining aspects of the method remain the same. This alternative model is called CBOW. Empirical evaluations show that skipgram gives better representations for rare terms.

Result

Matrices $W^{(w)}$ and $W^{(c)}$ that capture information on word similarity

- Words appearing in similar contexts generate similar contexts and vice versa
- Hence, mapped to similar representations in lower dimensional space
- Use $W = W^{(w)} + W^{(c)}$ as the low-dimensional representation

By using skipgram or CBOW we obtain two models mapping words into a low dimensional space corresponding to the two roles words can take, center words or context words. In practice, a final model is derived as the sum of the two matrices.

Alternative Formulation of the Problem

Consider as prediction problem

- Given a word, predict whether another word is a context word (skipgram)
- Given a context word, predict whether a word relates to this context (CBOW)

$$\theta = \underset{\theta}{\operatorname{argmax}} \prod_{w \epsilon V} \prod_{c \in C(w)} P_\theta(w|c) =$$

$$\underset{\theta}{\operatorname{argmax}} \sum_{w \epsilon V} \sum_{c \in C(w)} \log P_\theta(w|c)$$

The original formulation of the skipgram and CBOW models used a different approach. Instead of predicting the independent probabilities of occurrences of word-context pairs, it models the conditional probability of having a context word given a word (skipgram), respectively of having a word given a context word (CBOW). Using these conditional probabilities, a different optimization objective can be formulated, optimizing the observed conditional probabilities of word-context word occurrences.

Deriving Probabilities

Normalized scalar product?

- Produces values in [-1,1]
- $\sum_{i=1}^{m} P_\theta(w_i|c)$ does not add up to 1

Standard approach to convert a set of values into probabilities: **softmax function**

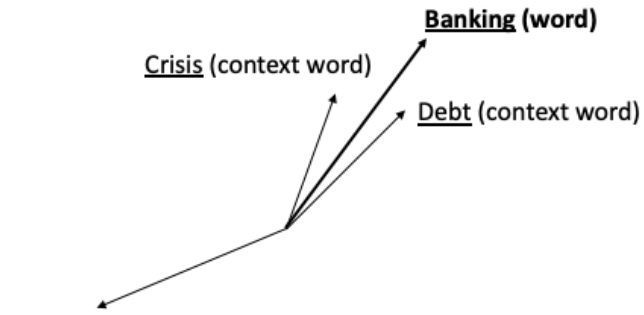
Values: $\boldsymbol{w_i} \cdot \boldsymbol{c}$ for $i = 1, \ldots, m$

$$P_\theta(w_i|c) = \frac{e^{\boldsymbol{w_i} \cdot \boldsymbol{c}}}{\sum_{j=1}^{m} e^{\boldsymbol{w_j} \cdot \boldsymbol{c}}}$$

For introducing a function that derives those conditional probabilities from the vector representation of words while using the scalar product, in addition to the problem that the values of the scalar product do not fall into the interval [-1, 1], we find the additional problem that the conditional probabilities for a given context word (CBOW) do not add up to 1.

To address this issue, there exists a standard approach to convert an (arbitrary) set of values into a probability distribution. First, applying the exponential function to the set of values (in our case the scalar products), produces positive values only, and second normalizing by the sum of the exponentials lets the values fall into the interval [0,1].

Illustration



Banking • Crisis = +3
Banking • Debt= +2
Banking • Monkey = -1

P(Banking, Crisis) = 0.72
P(Banking, Debt= 0.27
P(Banking, Monkey) = 0.01

```
from scipy.special import softmax
arr = [3,2,-1]
softmax(arr, axis=0)
```

```
array([0.72139918, 0.26538793, 0.01321289])
```


Applying the softmax function to the scalar products, produces now a probability distribution for a context word, co-occurring with some word of the vocabulary.

Hierarchical Softmax

Issue: computing $P_\theta(w|c)$ is expensive

- In each iteration of the gradient descent, for computing the sum the algorithm has to iterate over the complete vocabulary

Solution: Hierarchical softmax

- Compute approximate softmax function using a tree over the vocabulary

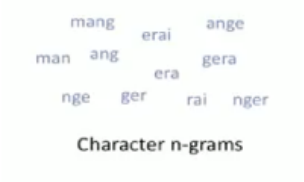
The issue with this variant of the loss function is that in the gradient descent the sums of the conditional probabilities $\sum_{i=1}^{m} P_\theta(w_i|c)$ have to be computed repeatedly which is prohibitively expensive. The solution to this problem is an approach called hierarchical softmax which performs an approximate computation of those sums using a tree-structure to organize the vocabulary.

1.4.4 Fasttext

Fasttext introduces the idea of using word n-grams (phrases) and **subword embeddings**

- Build embeddings for character n-grams
- Enable processing of unseen words

Example (in French): mangerai



Character n-grams

The word embedding models we have presented where based on the assumption that the vocabulary consists of the words found in the text corpus. Fasttext is a variant of word embeddings that has been developed by Facebook research. It incorporates two additional ideas:

- The use of word n-grams, respective phrases: often phrases carry a different meaning than their constituent words. Considering phrases (that are sufficiently frequent) can help to produce better representations of text and capture meaning that cannot be assigned to the constituents of the phrase. As an extreme example one might consider the phrase “The Who” which designates a rock band, whereas the constituent words have no meaning that would be related to music in any way.
- The use of character n-grams: many languages with a writing system based on an alphabet, e.g., Latin languages, have word variations with similar meanings resulting from grammatical rules. Often some variations might not occur in a training corpus. By using so-called subword embeddings the meaning identified for seen words, can so be transferred also to unseen works. This also works across languages, as often languages inherit expressions from foreign languages, or words are same or similar across different languages.

Subword Embeddings

For a given word create representations for all of its subwords S_w, including the word itself

- Add special characters at the start and end of the word
- Create all n-grams

Example: S_w for word "mangerai", n=3

- S_w = {_mangerai_, _ma, man, ang, nge, ger, era, rai, ai_}

Representation of w: $\boldsymbol{w} = \sum_{s \in S_w} \boldsymbol{s}$

When using subword embeddings, the representation of a word is aggregated from the representation of its constituent n-grams, by adding up the n-gram representations.

Byte Pair Encoding (BPE)

Subword embeddings potentially generate large numbers of possible tokens, many of which are not useful

- Large vocabulary implies training the model becomes expensive

BPE allows to create smaller vocabularies, while capturing essential subword information

- Statistically analyze text for frequent character sequences
- Replace frequent sequences by tokens

When using subword embeddings with character n-grams one faces the problem that the number of tokens, i.e., the size of the vocabulary, may become very large when all character n–grams occurring in the text are used. This in turn makes training the models very expensive. In addition, many of the tokens are rare and potentially not very useful. In order to address this problem, a method called byte pair encoding is used in order to obtain smaller vocabularies and select only frequent subwords. For this method, first the text needs to be analyzed to determine the frequent tokens.

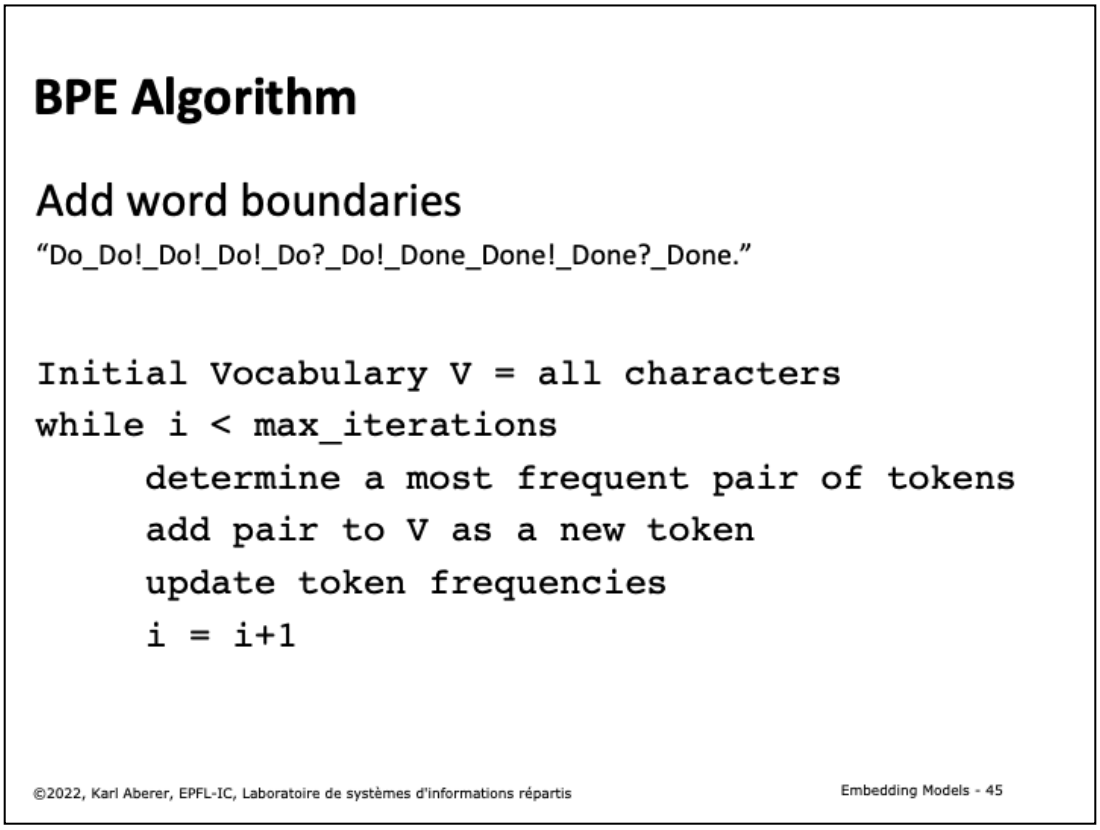
Example

"Do Do! Do! Do! Do? Do! Done Done! Done? Done."

What are good tokens?

- All subwords of size n=2,3,4,5
  {Do, o!, o?, o., on, ne, e!, e?, e., Do!, Do?, …}
- Better solution
  {Do, Do!, Done, !, ?, .}
  Tokenized text:
  [Do, Do!, Do!, Do!, Do, ?, Do!, ., Done, Done, !, Done, ?, Done, .]

This example illustrates the idea of BPE. Instead of blindly selecting all character n-grams, a carefully selected set of frequent n-grams is chosen and used to tokenize the text.



For running the BPW algorithm first a special token for word boundaries is added (in this example _).

The BPE algorithm proceeds iteratively. It determines in every iteration a most frequent pair of tokens and creates from this pair a new token. Then token frequencies are updated with the new set of tokens. The algorithm terminates after a predetermined number of tokens has been created.

In order to determine token frequencies without repeatedly scanning the text, the algorithm can initially scan the document collection to determine all n-gram frequencies (up to a given size).

BPE Example

Do_Do!_Do!_Do!_Do?_Do!_Done_Done!_Done?_Done.
V = {_, D, o, !, ?, n, e, .}

i = 1, most frequent pair: Do with frequency 9
V = {_, D, o, !, ?, n, e, ., v_0 = Do}
New text: v_0 _v_0!_v_0!_v_0!_v_0?_v_0 ne_v_0 ne!_v_0 ne?_v_0 ne.

i = 2: most frequent pair: !_ with frequency 5
V = {_, D, o, !, ?, n, e, ., v_0 = Do, v_1 = !_}
New text: v_0 _v_0 v_1 v_0 v_1 v_0 v_1 v_0?_v_0 ne_v_0 ne v_1 v_0 ne?_v_0 ne.

i = 3: most frequent pair: v_1 v_0 with frequency 4 (or ne with frequency 4)

Here we illustrate the first two steps of the algorithm. Having tokens that combine a character n-gram with a word boundary is important to identify character sequences that would typically occur at the end (or beginning) of a word and to distinguish them from character sequences that occur in the middle of a word.

For example, the sequence “ed” at the end of of an English word (like “worked”) implies a different meaning of this character sequences at the beginning of a word (like “education”) or in the middle of a word (like “medicine”)

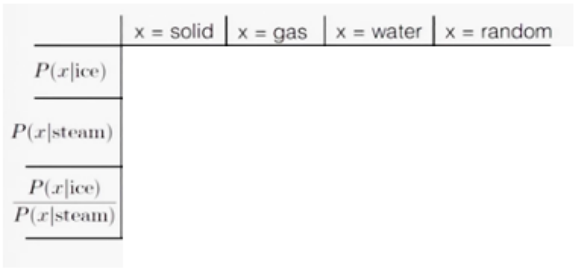
Question

Fasttext speeds up learning by

1. Considering subwords of words
2. By selecting the most frequent phrases in the text as tokens
3. By selecting the most frequent subwords in the text as tokens
4. By pre-computing frequencies of n-grams

1.4.5 Glove

GLOVE is based on the following analysis



1. Consider two terms, with similar, but related meaning: ice, steam
2. Determine the conditional probability $P(x|ice)$, $P(x|steam)$ of other terms x to co-occur within a window
3. Determine the ratio of these two conditional probabilities

The skipgram and CBOW models are one variant among several similar models that have been proposed recently to create word representations that capture word semantics.

One popular model is GLOVE, which is based on the observation that ratios of probabilities of co-occurrence can be used to capture more accurately semantic relationships among terms.

1.4.5 Glove

Observation: ratio captures additional information

	x = solid	x = gas	x = water	x = random
$P(x\|\text{ice})$	large	small	large	small
$P(x\|\text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~1	~1

"solid" is related to "ice" but unrelated to "steam":
we expect a larger ratio of co-occurrence probabilities

"gas" is related to "steam" but unrelated to "ice":
we expect a smaller ratio of co-occurrence probabilities

"water" is related to both "ice" and "steam":
we expect a ratio of co-occurrence probabilities that is close to 1 (both large)

A random word that is unrelated to both "ice" and "steam":
we expect a ratio of co-occurrence probabilities that is close to 1 (both small)

In this example, the term solid is much more likely to co-occur with ice, than with water, and similarly steam co-occurs much more likely with gas than with ice. On the other hand, water co-occurs frequently with both. In the Glove paper it is stated: "Compared to the raw probabilities, the ratio is better able to distinguish relevant words (solid and gas) from irrelevant words (water and fashion) and it is also better able to discriminate between the two relevant words".

Global Co-occurrence Count

Word embeddings with global vectors (GloVe)

- Denote by x_{ik} the global co-occurrence count of words w_i and w_k in the vocabulary, where w_i is the center word and w_k is the context word
- Note: $x_{ik} = x_{ki}$
- Denote by x_i the total number of occurrences of word w_i

Then

$$P(w_k|w_i) = \frac{x_{ik}}{x_i}$$

In order to consider the property illustrated before, Glove uses the global co-occurrence counts among words within a window. In this way, it incorporates into the model a global statistics on the co-occurrence of words, different to skipgram/CBOW which iterate over the different occurrences while learning the parameters.

Using the global co-occurrence counts we can determine a co-occurrence probability.

Modeling Ratios

Find a function f such that for three words w_i, w_j, w_k

$$f_\theta(\boldsymbol{w}_i, \boldsymbol{w}_j, \boldsymbol{w}_k) = \frac{P_\theta(w_k|w_i)}{P_\theta(w_k|w_j)}$$

If we choose

$$f_\theta(\boldsymbol{w}_i, \boldsymbol{w}_j, \boldsymbol{w}_k) = e^{(\boldsymbol{w}_i - \boldsymbol{w}_j)\cdot \boldsymbol{w}_k} = \frac{e^{\boldsymbol{w}_i \cdot \boldsymbol{w}_k}}{e^{\boldsymbol{w}_j \cdot \boldsymbol{w}_k}}$$

we get

$$e^{\boldsymbol{w}_i \cdot \boldsymbol{w}_k} = P_\theta(w_k|w_i) = \frac{x_{ik}}{x_i}$$

In order to formalize the observation that we have described, we need to design a function that models ratios between co-occurrence probabilities derived from their vector representations. In Glove this function is chosen to be of the form of an exponential of the scalar product of the difference of the representation of two (context) words with the representation of a given word. With this function, it follows that the conditional probability of co-occurrence is modelled as the exponential of the scalar product of the representations of the two words.

Glove Loss Function

With word w_i and context words w_k

Assume

$$e^{\boldsymbol{w}_i \cdot \boldsymbol{w}_k} = P_\theta(w_k|w_i) = \frac{x_{ik}}{x_i}$$

taking the logarithm

$$\boldsymbol{w}_i \cdot \boldsymbol{w}_k = \log x_{ik} - \log x_i$$

Adding additional bias terms for w_i and w_k

$$\boldsymbol{w}_i \cdot \boldsymbol{w}_k + \boldsymbol{b}_i + \boldsymbol{b}_k \approx \log x_{ik}$$

Based on the function we have chosen to represent the conditional probability of word cooccurence, we can derive a loss function. By taking the logarithm we obtain a linear relationship between the scalar product and the co-occurrence statistics. By adding so-called bias terms, which correspond to additional vector representations of the words, we obtain a relationship between the word embedding vectors and the co-occurrence probabilities. Note that by adding a bias term $\boldsymbol{b}_i$ the term $\log x_i$ is considered in this relationship.

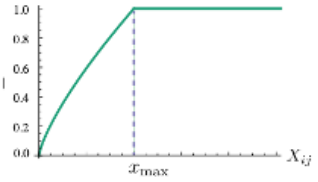
Glove Loss Function

Squared error loss with weights

$$J(\theta) = \sum_{i=1}^{m} \sum_{j=1}^{m} h(x_{ij})(\boldsymbol{w}_i \cdot \boldsymbol{w}_j + \boldsymbol{b}_i + \boldsymbol{b}_j - \log x_{ij})^2$$

Weight function

$$h(x) = \min\left(\left(\frac{x}{x_{max}}\right)^{\beta}, 1\right)$$



With $\beta = 0.75$ and $x_{max} = 100$

Based on the relationship introduced, a loss function is derived. In the case of Glove, the squared error is used. In addition, a weighting function h is introduced to address two problems:

- If x_{ij} equals zero, the loss function would be ill-defined. The limit $\lim_{x \to 0} h(x) \log ^2(x)$ is finite, making the function well-defined
- Small values of x_{ij} carry in general less information and should thus have lower weight.

Question

The most important difference between Glove and skipgram is

1. That Glove considers the complete context of a word
2. That Glove computes a global frequency for word-context pair occurrences
3. That Glove uses a squared error loss function
4. That Glove does not differentiate words and context words

Glove: Nearest words to Frog

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

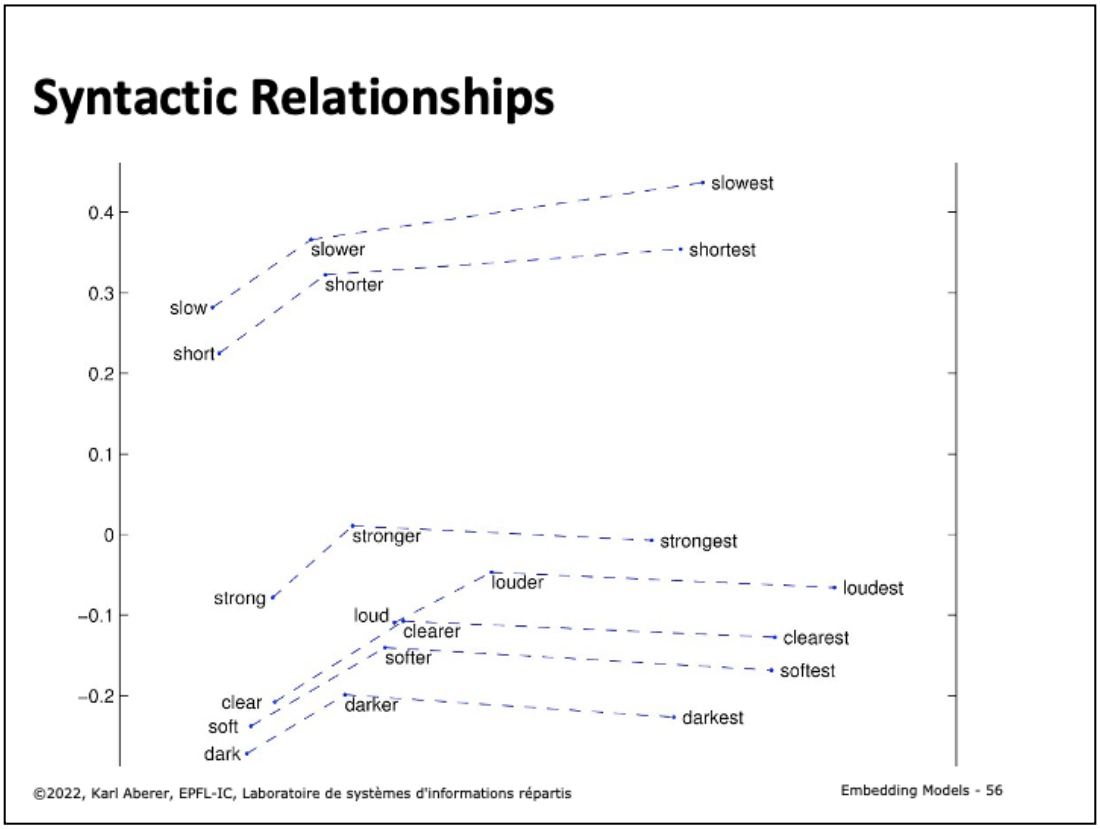
litoria

leptodactylidae

rana

eleutherodactylus

A first property is that similar words are grouped together. Here is a nice example, produced with Glove, that illustrates the point. (The underlying data is from Wikipedia).

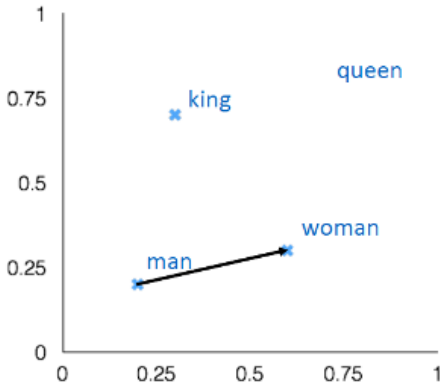


Word embeddings also capture syntactic relationships, like singular-plural or comparative and superlative, as shown here. This type of visualizations is obtained by projecting from the d-dimensional word embedding space (appropriately) into a 2-dimensional space.

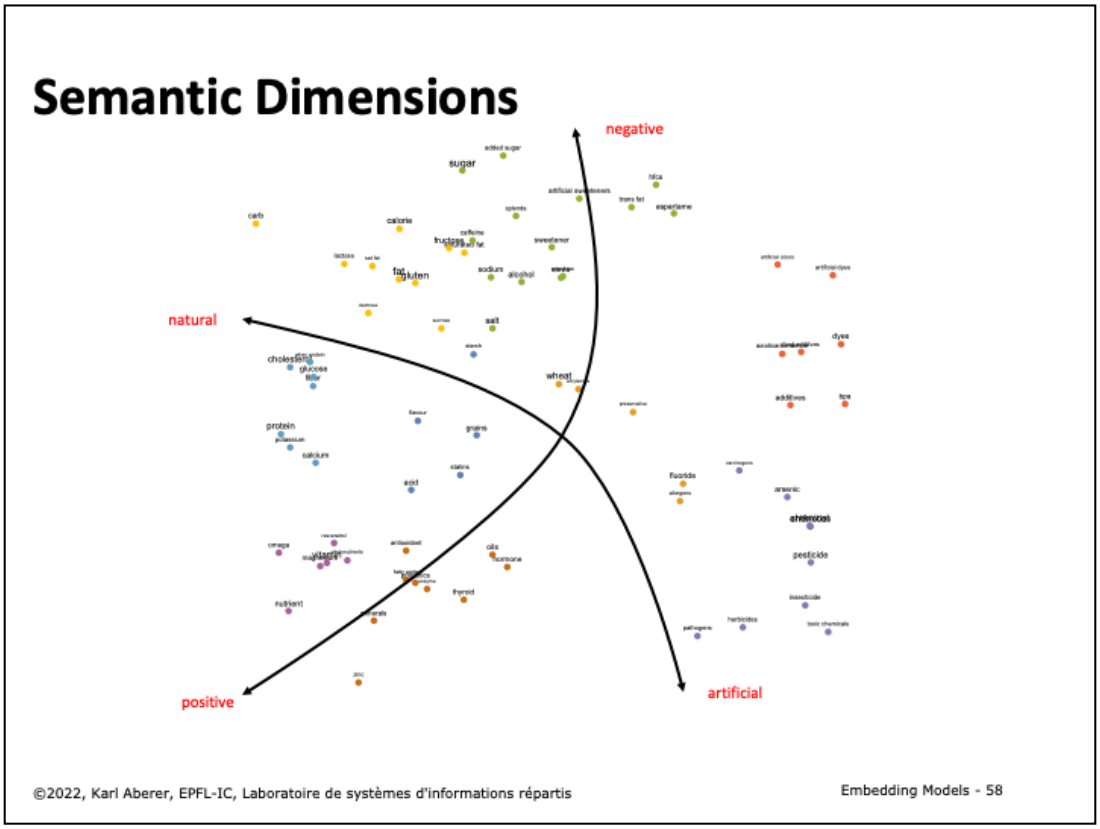
Word Analogies: semantic relationships

man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



Even more interestingly, word embeddings enable “computing” with relationships (this is called the word analogy task). Analogies translate into linear mappings! We will later exploit this property for extracting relationships from text.



Furthermore, word embeddings can capture semantic meaning in dimensions. In this example, data on nutrition has been analyzed and terms indicating qualities of foods are extracted. The word embedding clearly organizes it according to properties that are human understandable, e.g. natural vs. artificial ingredients.

Properties of Word Embeddings

1. Similar terms are clustered
2. Syntactic and semantic relationships encoded as linear mappings
3. Dimensions can capture meaning

We summarise some of the main properties of word embeddings.

Use of Word Embedding Models

Document Search

- Use word embedding vectors as document representation

Thesaurus construction and taxonomy induction

- Search engine for semantically analogous / related terms

Document classification

- Use of word embedding vectors of document terms as features

Word embeddings have beyond retrieval a wide range of applications.

Impact

- Latent semantic indexing

Indexing by latent semantic analysis
S Deerwester, ST Dumais, GW Furnas... - Journal of the ..., 1990 - search.proquest.com
Cited by 12027 Related articles All 87 versions Web of Science: 3525 Cite Save

+2000

Indexing by latent semantic analysis
S Deerwester, ST Dumais, GW Furnas... - Journal of the ..., 1990 - Wiley Online Library
A new method for automatic indexing and retrieval is described. The approach is to take advantage of implicit higher-order structure in the association of terms with documents ("semantic structure") in order to improve the detection of relevant documents on the basis of ...
☆ 99 Cited by 13986 Related articles All 76 versions Web of Science: 4451 »

- Latent Dirichlet allocation

Latent dirichlet allocation
DM Blei, AY Ng, MI Jordan - Journal of machine Learning research, 2003 - jmlr.org
Cited by 17791 Related articles All 123 versions Web of Science: 5278 Cite Save

+8000

Latent dirichlet allocation
DM Blei, AY Ng, MI Jordan - Journal of machine Learning research, 2003 - jmlr.org
We describe latent Dirichlet allocation (LDA), a generative probabilistic model for collections of discrete data such as text corpora. LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of ...
☆ 99 Cited by 25905 Related articles All 98 versions Web of Science: 8674 »

- Word embeddings

Distributed representations of words and phrases and their compositionality
T Mikolov, I Sutskever, K Chen, GS Corrado... - Advances in neural ..., 2013 - papers.nips.cc
Cited by 3370 Related articles All 23 versions Cite Save

+8000

Distributed representations of words and phrases and their compositionality
T Mikolov, I Sutskever, K Chen, GS Corrado... - Advances in neural ..., 2013 - papers.nips.cc
The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships. In this paper we present several improvements that make the Skip-gram model more expressive and enable it to learn higher quality vectors more rapidly. We show that by subsampling frequent words we obtain significant speedup, and also learn higher quality representations as measured by our tasks. We also introduce ...
☆ 99 Cited by 11619 Related articles All 37 versions »

References

Relevant articles

- Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems*. 2013.
- Goldberg, Yoav, and Omer Levy. "word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method." *arXiv preprint arXiv:1402.3722* (2014).
- Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.