**Dieses Dokument ist eine Zweitveröffentlichung (Postprint) /**

**This is a self-archiving document (postprint):**

Michael Günther

# FREDDY: Fast Word Embeddings in Database Systems

**Erstveröffentlichung in / First published in:**

*SIGMOD'18: 2018 International Conference on Management of Data*, Houston, 2014. ACM Digital Library, S. 1817–1819. ISBN 978-1-4503-4703-7

DOI: https://doi.org/10.1145/3183713.3183717

Diese Version ist verfügbar / This version is available on:

https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-384516

SLUB
Wir führen Wissen.

TECHNISCHE UNIVERSITÄT DRESDEN

QUCOSA
Quality Content of Saxony

# FREDDY: Fast Word Embeddings in Database Systems

Michael Günther
Database Systems Group
Technische Universität Dresden
Michael.Guenther@tu-dresden.de

**ACM Reference Format:**
Michael Günther. 2018. FREDDY: Fast Word Embeddings in Database Systems. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3183713.3183717

## 1 INTRODUCTION

Word embeddings are useful in many tasks in Natural Language Processing and Information Retrieval, such as text mining and classification, sentiment analysis, sentence completion, or dictionary construction. Word2vec [8] and its predecessor fastText [1], both well known models to produce word embeddings, are powerful techniques to study the syntactic and semantic relations between words by representing them in a low-dimensional vector. By applying algebraic operations on these vectors semantic relationships such as word analogies, gender-inflections, or geographical relationships can be easily recovered [7].

The aim of this work is to investigate how word embeddings could be utilized to augment and enrich queries in DBMSs, e.g. to compare text values according to their semantic relation or to group rows according to the similarity of their text values. For this purpose, we use pre-trained word embedding models of large text corpora such as Wikipedia. By exploiting this external knowledge during query processing we are able to apply inductive reasoning on text values. Thereby, we reduce the demand for explicit knowledge in database systems. In the context of the IMDB database schema, this allows for example to query movies that are semantically close to genres such as *historical fiction* or *road movie* without maintaining this information. Another example query is sketched in Listing 1, that returns the top-3 nearest neighbors (NN) of each movie in IMDB. Given the movie "Godfather" as input this results in "Scarface", "Goodfellas" and "Untouchables".

**Contribution:** We developed a system called FREDDY (Fast woRd EmbedDings Database sYstems) based on PostgreSQL to exhibit the rich information encoded in word embeddings. This includes a wide range of UDFs forming the base for novel query types, supported by different index structures and approximation techniques to accelerate the operations on high-dimensional vector spaces. Experimental results on IMDB together with large word2vec models show, that our approach is able to efficiently process these new query types.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*SIGMOD'18, June 10–15, 2018, Houston, TX, USA*
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4703-7/18/06.
https://doi.org/10.1145/3183713.3183717

```
SELECT m.title, t.term, t.score
FROM movies AS m, kNN(m.title, 3) AS t
ORDER BY m.title ASC, t.score DESC
```

**Listing 1: Simplified IMDB Query Example (joins removed)**

## 2 FREDDY: SYSTEM OVERVIEW

We built a prototype[1] based on a PostgreSQL database system sketched in Figure 1. A script creates new relations for the different index structures proposed in Section 3. There is a product quantization for fast approximated exhaustive distance calculation and a non-exhaustive index for operations with no specific output word set. For exact distance computations an additional relation is created storing terms and the respective word2vec vector. If two terms are similar, the corresponding vectors have a low Euclidean distance and high cosine similarity. UDFs are implemented which operate on the index relations to make use of the word embeddings. We developed UDFs for calculating the cosine similarity of two vectors ($cos\_sim(input1, input2)$), kNN search on a selection of tokens ($kNN(input, k, output\_set)$) or the whole set of tokens for which word embeddings are provided ($kNN(input, k)$), analogy queries ($analogy(input1, input2, input3)$, equivalent to 3CosAdd in [7]) and grouping by clustering word vectors ($cluster(k, terms)$). The UDFs accept terms, i.e. text values, or vectors as input and are implemented in C using the PostgreSQL Server Programming Interface (SPI) to run SQL commands inside the functions. All UDFs are bundled into a PostgreSQL extension. Benefits of implementing these operations as UDFs are that they could be used in SQL queries and query optimization is still active.

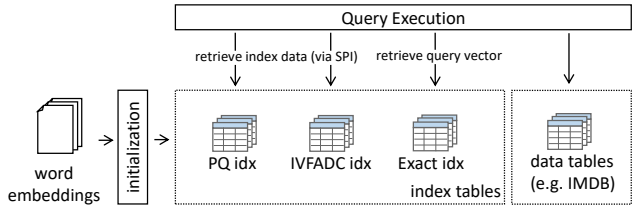


**Figure 1: FREDDY System Overview**

## 3 INDEX STRUCTURES

Word embeddings usually encompass a few million vectors with 100-300 dimensions. Since for kNN queries it is necessary to calculate the distances from an input vector to every other vector, the distance computation is getting quite comprehensive. To speed up kNN queries many approaches hierarchically divide the vector space, reducing the number of vectors to consider by using

[1] https://github.com/guenthermi/postgres-word2vec

implications from the triangle inequality [4]. In a small analysis using a large word2vec model we found that the average angles between randomly sampled vectors and there 5 NNs are over 40 degree. Because of this spread distribution of the vector space and the high dimensionality of the vectors such indexing methods are not applicable for exact kNN search. Another popular approach for exact kNN is to use KD-trees [3], but as stated in [10] it is also not suitable for high dimensional vectors. For this reason, approximated nearest neighbor search (aNN) like LSH [5] and product quantization (PQ) [6] are widely disseminated. While LSH might be one of the most popular methods for aNN it is based on hash functions which can not be adapted to the set of input vectors. Thus we decided to use product quantization that provides an exhaustive method for aNN.

**Product quantization** is a method to decompose a high dimensional vector space into the Cartesian product of subspaces and then quantize these subspaces separately. Therefore, it divides a vector $\mathbf{y} = [y_1, \ldots, y_n]$ of dimensionality $n$ into $m$ subvectors $u_i(\mathbf{y})$ with a dimension $dim(u_i) = d = n/m$. On each of these subvectors, a function is applied which is called quantizer. A quantizer $q$ assigns an arbitrary input vector $\mathbf{v}$ to the nearest vector $c_j$ out of a set of vectors $C$ which we call *centroids*. There are $m$ different quantizers $q_1, \ldots q_m$ and corresponding centroids sets $C_1, \ldots, C_m$ for the subvectors. The product quantization of $\mathbf{y}$ can be defined as follows:

$$\underbrace{y_1, \ldots y_d}_{u_1(\mathbf{y})}, \ldots, \underbrace{y_{(n-d)+1}, \ldots y_n}_{u_m(\mathbf{y})} \rightarrow q_1(u_1(\mathbf{y})), \ldots, q_m(u_m(\mathbf{y}))$$

It can be represented by an id sequence $s \in Seq$ which consists of numbers $l \in \{1, \ldots, |C_k|\}$ corresponding to centroid vectors $c_{j,k}$ where $k \in \{1, \ldots, m\}$ represents the quantizer and $j \in \{1, \ldots, |C_k|\}$ refers to one of the centroids in $C_k$. Usually, the number of centroids in every set $C_k$ is equal.

$$Seq = \{1, \ldots, |C_k|\}^m$$

The product quantization can be used for the fast computation of an approximated Eucleadian distance $\hat{d}(\mathbf{x}, \mathbf{y})$ of $d(\mathbf{x}, \mathbf{y})$ between a query vector $\mathbf{x}$ and a word vector $\mathbf{y}$ from the index.

$$\hat{d}(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_j d(u_j(\mathbf{x}), q_j(u_j(\mathbf{y})))^2}$$

Therefore, all $m \cdot |C_k|$ possible values for the square distances $d(u_j(\mathbf{x}), q_j(u_j(\mathbf{y}))^2$ are pre-computed. Thus the calculation of $\hat{d}(\mathbf{x}, \mathbf{y})^2$ is then done by simply summing up such distances. To provide an index based on product quantization we have to store the centroids $C_1, \ldots, C_m$ called *codebook*, and the sequences of nearest centroid identifiers for each vector. The centroids are determined by applying k-means on the vectors (or a subset of them).

**IVFADC.** To speed up the search, even more, a non-exhaus-tive index based on product quantization called IVFADC [6] could be used. IVFADC is build up by applying a *coarse quantizer* $q_c$ on the vectors which should be indexed. For each vector $\mathbf{v}$ the residual $\mathbf{r}(\mathbf{v}) = \mathbf{v} - q_c(\mathbf{v})$ is calculated and stored in a list with every other residual of vectors $\mathbf{v}'$ with the same quantization $q_c(\mathbf{v}') = q_c(\mathbf{v})$. For aNN search on the query vector $\mathbf{x}$ first the quantization $q_c(\mathbf{x})$ and the residual $\mathbf{r}(\mathbf{x})$ is calculated. Then the nearest neighbors are determined by determining the vectors according to the nearest residuals of the list for the respective quantization $q_c(\mathbf{x})$.

## 4 EVALUATION

We used the popular free available word2vec dataset generated by [8] that contains 3M 300-dimensional vectors. For PQ and IVFADC we set the number of subvectors to $m = 12$ and the number of centroids for the fine quantizer to 1024. For IVFADC the number of centroids for the coarse quantizer is set to 1,000. For the evaluation of the index structures, we executed 100 kNN queries that find the 5 most similar words to a randomly chosen word out of all 3M entries. As shown in Table 1 product quantization is about 9 times and IVFADC about 300 times faster than the exact search at the expense of precision which drops down to about 35%. An batchwise execution of queries can speed up the computation even more. While the precision measurements are quite low they can be improved using postverification and a manual investigation showed that the terms in the result set are still very useful. Since IVFADC is a non-exhaustive search method it is not applicable to all types of word embedding operations. For example, for *kNN* queries which consider just a subset of word vectors and *kNN* queries with a high $k$ a simple PQ index should be used.

| **Index Structure** | **⌀Resp. Time** | **⌀Precision** |
|---|---|---|
| Exact Search | 8.79s | 1.00 |
| Product Quantization | 1.06s | 0.38 |
| IVFADC | 0.03s | 0.35 |
| IVFADC (batchwise) | 0.01s | 0.35 |
| Product Quantization (postverif.) | 1.29s | 0.87 |
| IVFADC (postverif.) | 0.26s | 0.65 |

**Table 1: Time and Precision Measurement of Index Structures**

## 5 RELATED WORK

There are many methods for generating word embeddings like [1], [8] and [9]. An integration of word embeddings into an Apache Spark system was proposed by [2]. Product quantization was introduced by [6] for approximated kNN search on SIFT and GIST image descriptors which are datasets with vectors of similar dimensionality.

## 6 CONCLUSIONS

In this paper, we propose FREDDY, a system based on PostgreSQL, to exploit the expressive power of word embeddings. UDFs have been shown to be an elegant way to integrate operations like distance calculation, kNN search, grouping, etc. into a database system. We tackled the performance problems regarding word embedding operations by using product quantization and IVFADC [6].
Currently, we are improving our tokenization techniques and working on two other types of analogies (PairDirection and 3CosMul) [7], for which we are not able to use PQ or related data structures. Finally, we are looking into real-world application domains that can draw benefit from our system.

# REFERENCES


[1] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606* (2016).

[2] Rajesh Bordawekar and Oded Shmueli. 2016. Enabling cognitive intelligence queries in relational databases using low-dimensional word embeddings. *arXiv preprint arXiv:1603.07185* (2016).

[3] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. 1977. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)* 3, 3 (1977), 209–226.

[4] Keinosuke Fukunaga and Patrenahalli M. Narendra. 1975. A branch and bound algorithm for computing k-nearest neighbors. *IEEE transactions on computers* 100, 7 (1975), 750–753.

[5] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, Vol. 99. 518–529.

[6] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2011), 117–128.

[7] Omer Levy and Yoav Goldberg. 2014. Linguistic Regularities in Sparse and Explicit Word Representations. (2014), 171–180 pages. https://doi.org/10.3115/v1/W14-1618

[8] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[9] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[10] Chanop Silpa-Anan and Richard Hartley. 2008. Optimised KD-trees for fast image descriptor matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 1–8.