Context Management in Word Embedding Database Systems

Considerations for Finding a Topic
Introduction

Word embedding

Language learning methods

Text corpora in natural language

Word embedding operations

Language learning methods

Text corpora in natural language
Introduction

**Contribution of word embedding to database systems**

- Use of external data sources of unstructured data (text in natural language)
- New operations for unstructured text values in the database
  - Analysing values
  - Extract new information from such values

```sql
SELECT m.title, t.word, t.squaredistance
FROM movies AS m,
     most_similar(m.title, (SELECT title FROM movies)) AS t
```

**Execution of most_similar operation**

Results:
Inception | Shutter Island
...
Word-Embeddings

Word Embeddings

- Mapping: Tokens → Vectors
- Vectors modell semantic as well as syntactic relations between tokens.
- Useful for NLP techniques (Sentiment Analysis, Machine Translation, Information Retrieval, Word Clouds)

Properties

- Pretrained Word Embedding Datasets contain usually a few million vectors
- Dimensionality of the vectors: 200-300

Word Relations

Word-Embeddings: Operationen

Quantify Similarity

- Cosine similarity between vectors:
  \[ \text{sim}_{\text{cos}}(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||} \]
- Example: Top5('birch') → 'pine', 'birch trees', 'birches', 'tamarack', 'cedar'

Analogies

- Analogy Queries: \( a - b \approx c - ? \)
  e.g. man – woman ≈ king - ? → queen
- Pair-Direction: \( \arg \max_{d \in V \setminus \{a,b,c\}} (\text{sim}_{\text{cos}}(a - b, c - d)) \)
- 3CosAdd: \( \arg \max_{d \in V \setminus \{a,b,c\}} (\text{sim}_{\text{cos}}(d, c - a + b)) + \text{sim}_{\text{cos}}(d, b) \)

Relation Plot: man – woman
Source: [https://nlp.stanford.edu/projects/glove/](https://nlp.stanford.edu/projects/glove/)
Last access: 08.03.2018
System architecture

Fast woRd EmbedDings in Datatbase sYstem

Basis
- Postgres database system → Open source, Extensibility

Word Embedding Operations
- implemented as User-Defined-Functions (UDFs)
  → Query optimization still active
  → Can be used in SQL queries
  → Search methods implemented in C
  → Interfaces implemented in PL/pgSQL

Index structures
- Stored in database relations
- Currently used index structure can be selected with UDFs while runtime
WE operations for database system

Use cases

- **Similarity Queries**
  
  ```sql
  SELECT keyword
  FROM keywords
  ORDER BY cosine_similarity('comedy', keyword)
  → comedy, sitcom, dramedy, comic, satire, ...
  ```

- **kNN Queries***
  
  ```sql
  SELECT m.title, t.term, t.score
  FROM movies AS kNN(m.title, 3) AS t
  ORDER BY m.title ASC, t.score DESC
  → Godfather | {Scarface, Goodfellas, Untouchables}
  ```

- **Analogy Queries**
  
  ```sql
  SELECT analogy_3cosadd('Godfather','Francis_Ford_Coppola', m.title)
  FROM movies AS m
  • Inception → Christopher Nolan
  ```

- **kNN_In Queries***
  
  ```sql
  SELECT DISTINCT title
  FROM movies
  WHERE keyword = ANY(
  SELECT term
  FROM kNN_in('historical fiction', 10,
  ARRAY(SELECT keyword FROM movies))
  → Movies for keywords: historical, fiction, literary, fictionalized, novels
  ```

- **Grouping***
  
  ```sql
  SELECT term, groupterm
  FROM grouping(SELECT title FROM movies), '{Europe, America}')
  → Melancholia | Europe
  → Godfather | America
  ```

- **Helper functions**, e.g. to calculate centroids, ...

* Function calls simplified
Product Quantization

**Idea**
Reduce the computation time of the Euclidean square distance through an approximation by a sum of precomputed distances
→ compact representation of vectors in index structure
→ low computation time for distances

**Preprocessing**
Split vectors in $m$ subvectors
→ apply k-means on subvectors to obtain $k$ centroids for every interval → quantizer $q_1, \ldots, q_m$

**Product-Quantization**
1. Split vector in subvectors
2. Apply quantizers
   → Represent Product-Quantization as sequence
3. approximate squared distances by sums of precomputed squared distances $d(u_j(x), q_j(u_j(y)))^2$

Query vector
Vector from index
Splitting into $m$ subvectors of $y$ with $d$ dimensions
Quantizer: assigns sub vector to one of the centroid of $C_k$

Product quantization:

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

→ Representation as sequence
$$Seq = \{1, \ldots, |C_k|\}^m$$

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

Query vector
$$\mathbf{x} = [x_1, \ldots, x_n]$$
Vector from index
$$\mathbf{y} = [y_1, \ldots, y_n]$$
Splitting into $m$ subvectors of $\mathbf{y}$ with $d$ dimensions
$$u_1(\mathbf{y}), \ldots, u_m(\mathbf{y})$$
Quantizer: assigns sub vector to one of the centroid of $C_k$
$$q : \mathbb{R}^d \rightarrow \{c_1, \ldots, c_k\}$$
Product Quantization - Search

**Index creation**
- Use k-means to calculate centroids for quantizer $q_1, \ldots, q_m$ and store them in a relation called “codebook”
- Calculate sequences for every vector and store them in a quantization table together with the corresponding token

**Search**
- Split query $x$ vector into subvectors
- Precompute square distances $d(u_j(x), q_j(u_j(y)))^2$ by using the codebook relation and the subvectors of $x$
- Determine the approximated kNN using the summation method to calculate distances for all sequences in the lookup table.

---

Quantization table

Word

Sequence for product quantization of Vector $\text{seq} \in \{1, \ldots, |C_k|\}^m$

Distance calculation

Precomputed distances of subvectors

$C_1, 1, \ldots, C_1, |C_1|, \ldots, C_m, |C_m|$

Product quantization search

Query vector $x = [x_1, \ldots, x_n]$
**IVFADC**

**Idea:**
Accelerate computation by providing a non-exhaustive index with an inverted lookup

**Preprocessing:**
- A coarse quantizer $q_c$ which quantize the whole vectors (considering all dimensions) is applied
- The residual vector $r(y) = y - q_c(y)$ is calculated for every vector
- Product quantization is applied on the residual
- A coarse lookup table is created which refers to lists of sequences of product quantizations for residual vectors of vectors with the same coarse quantization

**Calculation:** Approximated distances can be calculated by:

$$
\hat{d}_r(x, y) = \sqrt{\sum_j d(u_j(r(x)), q_j(u_j(r(y))))^2}
$$
Comparison: PQ-, IVFADC- and exact Search

**IVFADC Search**

*Very fast* $\approx$ 300 times faster

*Non-Exhaustive*: Considers only a subset of the vectors in the index

* Appropriated for:*
  - kNN queries
  - 3CosAdd analogy queries $\max(\cos(v_1 - v_2 + v_3), ?)$

*Inappropriate for:*
  - Computation of single similarity values
  - Search queries with specific output set

**PQ Search**

*Intermediate fast*: $\approx$ 9 times faster

*Exhaustive*: Considers all vectors

* Appropriated for:*
  - kNN-In queries
  - 3CosAdd analogy queries on a specific output set
  - Grouping queries

*Inappropriate for:*
  - Computation of single similarity values
  - Pair direction queries

**Exact computation**

*Slow (but no preprocessing)*

*Separate calculation of all similarity values (exact)*

* Appropriated for:*
  - Single similarity calculations
  - Pair direction queries
  - Search queries on a specific output set

*Inappropriate for:*
  - Search queries on huge datasets
Post verification

Method

- Re-ranking of aNN results by exact kNN search
- Improve quality of results by retrieving more results $f > k$ of nearest neighbors in the first run
  → Select best results with exact kNN Search
- Precision could be improved a lot
  → Especially useful for analogy queries

```
SELECT ANN.word
FROM k_nearest_neighbour_ivfadc('Godfather', 500) AS ANN
ORDER BY cosine_similarity('Godfather', ANN.word) DESC
FETCH FIRST 3 ROWS ONLY
```
Range Queries

Problem Setting

- Many SQL queries trigger a lot of aNN queries at one time
- Retrieving index data from database with independent queries needs a lot of time
- Retrieval of the same index data (e.g. codebook) multiple times

Range Query Approach

- Reduce retrieval time for aNN queries with batch-wise execution of queries
- UDF for range queries:

```sql
SELECT word
FROM k_nearest_neighbour_ivfadc_batch(ARRAY(SELECT title FROM movies), 3);
```
Range Queries

Algorithm

1. Determine coarse quantizations for query vectors and differentz vectors (residual vectors)
2. Create lookup: coarse quantizations -> query vector
3. Precalculate quadratic distances of subvectors
4. Retrieve IVFADC index entries (CoarseID, PQ-Sequenz von Residuum)
5. Iterative Processing of index entries:
   1. Retrieve residual vectors of query vectors with the same coarse id via lookup
   2. Calculate approximated distances between residual vectors
   3. Update aNN for query vector
Evaluation

Evaluation Setup

- Search for 5 nearest neighbors
  - Calculation of response time and precision
  - Measurement for 100 Queries
    → Determine average values

- Dataset:
  - 3 million vectors
  - Dimensionality: 300

- Index parameter:
  - Length of PQ-sequences $m = 12$
  - Number of centroids for $q_1 \ldots q_m$: 1024
  - Number of centroids for $q_c$: 1000
  - Results for post verification $f$: 1000
  - Size of batches: 100

### Time and precision measurements

<table>
<thead>
<tr>
<th>Index</th>
<th>Response time</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact search</td>
<td>8.79s</td>
<td>1.0</td>
</tr>
<tr>
<td>PQ search</td>
<td>1.06s</td>
<td>0.38</td>
</tr>
<tr>
<td>IVFADC</td>
<td>0.03s</td>
<td>0.35</td>
</tr>
<tr>
<td>PQ search (postverif.)</td>
<td>1.29s</td>
<td>0.87</td>
</tr>
<tr>
<td>IVFADC (postverif.)</td>
<td>0.26s</td>
<td>0.65</td>
</tr>
<tr>
<td>IVFADC (batch wise)</td>
<td>0.01s</td>
<td>0.35</td>
</tr>
</tbody>
</table>
Current and Further Research
System Performance

Word2Bits


▪ Quantization of coordinate values in the training algorithm
▪ Allows compressed representation
▪ Act as a regularizer
▪ Study work: Lukas Stracke

Research Idea

▪ Combine Word2Bits Approach with PQ- and IVFADC-search methods
▪ Finite number of possible centroids
  → Allows fast exact search

\[ Q_1(x) = \begin{cases} 
\frac{1}{3} & x \geq 0 \\
-\frac{1}{3} & x < 0 
\end{cases} \]
LSH (Locality Sensitive Hashing)

- Hash functions mapping vectors which are nearby with high probability to the same bit sequence
- Index is obtained by applying multiple such locality sensitive hash functions on the vectors → Create lookup: hash value → vector
- Hash functions can be applied to the query vector → lookup vectors with same or similar hash values

Research Idea

- Integration in relational database system
- Comparison with current aNN search methods
- Realization in bachelor theses: Carl Naumann

\[ h : \mathbb{R}^D \to \{0,1\}^k \]
\[ LSH(v) = h_1(v), \ldots, h_l(v) \]
Demonstrator

**FREDDY Demo**

- Web application as an interface for the WE-DBS → Currently only a command line interface
- Interactive Visualization of the performance and precision of the implemented search methods
- Submission for the CIKM 2018 (Deadline: 25.5.) (Demonstrator + Demo-Paper)
- Realization in Bachelor Theses: Zdravko Yanakiev

http://141.76.47.127:3000/
Context Advisor and Preprocessing

**Problem Setting**
- Word vectors may refer to different entities as the tokens in the database (e.g. apple: fruit vs. Apple Inc.)

→ Analyze context of the word vectors and database entities to make it possible to combine both information sources

**Challenges**

1. **Extract structured Information of text values in the DBS**
   - Database does not contain explicit knowledge about the semantic of textual values
     → Obtain semantic knowledge by observing the relations
   - Column describes a context for the text values in it
     → Could be used to cope with polysemy of words
   - Different text values can refer to the same instance (e.g. aliases, nicknames, etc.)
Context Advisor and Preprocessing

Challenges

(2) Determine if entities are represented in the word embedding dataset
- Observe how far structured knowledge is encoded in the word vectors
- Do relations encoded in the word vectors contradict with relations in the database?

(3) Map Text Values to Word Vectors
- Align structured knowledge in the database with the word vectors
- Decide which word embedding fits to the text value
- System can contain multiple word embedding datasets → Decide which word embedding dataset fits best

(4) Result Set Interpretation for WE-DBS-Queries
- kNN is not always meaningful (too small similarity values have low validity or at least could hardly be interpreted)
  → Quantify certainty of the truth of results

(5) Word Embedding Imputation: Integrate Missing Entities in Word Embedding Dataset
Vectorization of structured knowledge and align to word vectors
Related Work

**Sense2Vec: NER tagging before training**


- Named Entity Recognition as preprocessing → Classes are annotated to named entities
- Instead of vector set for words vector set for senses

![Diagram](image.png)

- Text corpora in natural language
- NER Tool
- Apple has presented a new product in Chicago ...
- Text corpora where NE are tagged
- Apple_ORG has presented a new product in Chicago_GPE ...
- Word2Vec
- Apple_ORG: [0.343, -0.212, ...]
- Apple_NOUN: [0.343, -0.212, ...]
Related Work

Context-Specific Multi-Prototype Word Embeddings


- Assumption: Different word senses occur in different contexts
- Idea: Convolutional Layer represents context → Trained to predict word sense from context representation
- Second Step: If context vector is dissimilar to sense vector → Create additional sense vectors for the respective words
  → Retrain the model with multiple sense vectors per token

Related Work

Densifer: Focus information for specific properties (sentiment, frequency, concreteness) in ultradense subspaces


- Training of Orthogonal Matrix $Q \in \mathbb{R}^{d \times d}$ for projecting Word Embedding $e_w \in \mathbb{R}^d$ in a vector space where specific dimensions (ultradense subspaces) represent specific properties of the token (e.g. sentiment, concreteness).
- Subspace $u_w \in \mathbb{R}^{d^*}$ can be obtained by multiplication with an Identity Matrix $P \in \mathbb{R}^{d^* \times d}$ specific for the property:
  \[
u_w = PQ e_w\]

Related Work

**Translation Matrix**


- Training of Translation Matrix $W$ for Transformation of Embeddings from one vector space to another
  \[
  \min_W \sum_{i=1}^{n} \|Wx_i - z_i\|^2
  \]

- Training Data: small dictionary of token pairs $(x_i, z_i)$

- Training with Stochastical Gradient Decent

Related Work

**Joint Embeddings (for Knowledge Graph Completion)**

- **Joint Model - Embeddings for nodes in knowledge graphs and tokens in texts**
- **Knowledge Graph Nodes:**
  - Minimize: $||h + r - t||$ for a fact $(h, r, t)$ in the Graph (Edge)
- **Text Model:**
  - Similar to Word2Vec SkipGram Model (Trained to predict probability of co-occurrence)
- 3 Likelihood consists of three terms:
  - Knowledge Model $L_K$
  - Text Model $L_T$
  - Alignment $L_A$:
  - Nodes with the same label as named entities should have similar vectors as the according tokens
- **Training:** Maximization of $L_K + L_T + L_A$
Related Work

Uncertainty of word vector similarity


- Measured the distribution of Similarity values
- Determine uncertainty \( \varphi \) of similarity values by training a model two times on the same text corpora with different model initialization
  → Low similarity values are more uncertain
  → For specific tasks usability of word vectors could be improved by thresholds for similarity values

Complexity of the WE-Queries

Multiple Datasets:
- Word embedding operations can be executed on different WE datasets
- Word embedding datasets could be combined
- (Not all entities in a column have an corresponding instance in the word embedding dataset)

Multiple Parameters:
- Different queries have different demands in terms of
  - Precision of Search Operations itself
  - Execution Time of the Operations
  - Certainty of similarity values

Index structures
- Multiple index structures for one dataset (different types and different parameters)
- Change over time (It is possible to add entities during runtime)
Complexity of the WE-Queries

Deal with the Complexity

- Huge number of User Defined Functions:
  At the moment 86 additional UDFs (and there will be more ...)
  But: Only 5 basic operations

- At the moment two options:
  1) Cope with complexity – Operations with a lot of parameters
  2) Transfer configuration to separate functions – define configuration global

  Problems:
  • Non-transparent: same query returns different results (with different configuration)
  • Inflexible: multiple operations in one query share the same configuration

- Possible Solution:
  - Objects storing for database entries how they could be examined by word embedding operations
  - In specific contexts where an entity is used it might play different roles