Local Embeddings for Relational Data Integration

Riccardo Cappuzzo∗, Paolo Papotti∗, Saravanan Thirumuruganathan#
∗EURECOM, France  #QCRI, HBKU, Qatar
{riccardo.cappuzzo,papotti}@eurecom.fr, sthirumuruganathan@hbku.edu.qa

September 4, 2019

Abstract

Integrating information from heterogeneous data sources is one of the fundamental problems facing any enterprise. Recently, it has been shown that deep learning based techniques such as embeddings are a promising approach for data integration problems. Prior efforts directly use pre-trained embeddings or simplistically adapt techniques from natural language processing to obtain relational embeddings. In this work, we propose algorithms for obtaining local embeddings that are effective for data integration tasks on relational data. We make three major contributions. First, we describe a compact graph-based representation that allows the specification of a rich set of relationships inherent in relational world. Second, we propose how to derive sentences from such graph that effectively “describe” the similarity across elements (tokens, attributes, rows) across the two datasets. The embeddings are learned based on such sentences. Finally, we propose a diverse collection of criteria to evaluate relational embeddings and perform extensive set of experiments validating them. Our experiments show that our system, EmbDI, produces meaningful results for data integration tasks and our embeddings improve the result quality for existing state of the art methods.

1 Introduction

The problem of data integration concerns the combination of information from heterogeneous sources. It is a challenging first step before data analytics can be performed to extract value from data. Due to its importance, the problem of data integration has been studied extensively by the database community. Traditional approaches often require substantial amount of involvement from domain experts in order to achieve acceptable levels of accuracy.

Recently, there has been increasing interest from the database community in adapting techniques from Deep Learning (DL) for data integration. Specifically, word embeddings have been successfully used for data integration tasks such as entity resolution [2, 9], schema mapping [4], identification of related concepts [3], and data curation [11] in general. Typically, these works fall into two dominant paradigms: (a) they seek to adapt pre-trained word embeddings for the given task or (b) they seek to build dataset specific (local) relational embeddings based on natural language processing. Both approaches provide promising results that are comparable to the state-of-the-art.

We advocate for the design of embeddings that leverage both the relational nature of the data and the downstream task of data integration. Simply adapting embedding tech-
niques originally developed for textual data often ignores the richer set of semantics inherent in relational data. Consider a specific cell value \( t[A_i] \) of an attribute \( A_i \) in tuple \( t \). Conceptually, it has semantic connection with other attributes of tuple \( t \) and other values from the domain of attribute \( A_i \). Furthermore, integrity constraints such as functional dependencies imbibe additional notion of semantics. Finally, the task of data integration could be viewed from the prism of similarity between two entities. Thus, embeddings must be learned from the data so that they meaningfully leverage similarity across other data sources and between different types of entities such as tuples and attributes.

We introduce \textsc{EmbDI}, an effective system for building relational, local embeddings for data integration that introduces a number of innovations. First, we leverage a tripartite graph based representation of relational data that is compact and effectively represents a rich set of syntactic and semantic relationships between cell values. The materialization of the walks over this graphs lead to the corpus of sentences from which embeddings are learned. Second, we describe how to produce sentences by exploiting metadata such as the tuple and attribute ids. With our local embeddings, one can infer sophisticated semantic relationships that are key for data integration. Our approach is agnostic to the type of random walks performed and to the specific algorithm used for learning embeddings. The process can be done for each dataset individually or over a pooled meta-dataset. Furthermore, generating the embeddings for tuple and attribute ids allows one to achieve meaningful results on popular integration tasks such as schema mapping and entity resolution.

We propose an extensive set of desiderata for evaluating relational embeddings for data integration. Specifically, our evaluation focuses on three major dimensions that measure how well do the embeddings (a) learn the relationships in the data, (b) learn integration specific information and (c) could be used for improving the behavior of DL based algorithms. As we shall show in the experiments, our proposed algorithms perform well on each of these dimensions.

Outline. Section 2 introduces necessary background about embeddings. Section 3 shows a motivating example that highlights the limitations of prior approaches and identifies a set of desiderata for relational embeddings. Section 4 introduces the system architecture and details the major components. Section 5 reports extensive experiments validating our proposed approach. We conclude in Section 6 with some promising next steps.

2 Background

Embeddings. Embeddings (also known as distributed representations) map an entity such as a word to a high dimensional real valued vector. The mapping is performed in such a way that the geometric relation between the vectors of two entities represents the co-occurrence/semantic relationship between them. Algorithms to learn embeddings approaches rely on the notion of “neighborhood”: intuitively, if two entities are similar they frequently belong to the same contextually defined neighborhood. When this occurs, the embeddings generation algorithm will try to force the vectors that represent these two entities to be close to each other in the resulting vector space. In this paper, our proposed algorithms are based on embeddings for words and nodes in a graph. We provide additional details about how these embeddings are learned.

Word Embeddings\cite{12} are trained on a large corpus of text and produces as output a vector space where each word in the corpus
is represented by a real valued vector. Usually, the generated vector space has either 100 or 300 dimensions. The vectors for words that occur in similar context – such as SIGMOD and VLDB – are in close proximity to each other. Popular architectures for learning embeddings include continuous bag-of-words (CBOW) or skip-gram (SG). Suppose we are given a sentence and a word in it. Given a set of context words, CBOW tries to predict the word while SG tries to predict the contextual words from a given word.

**Node Embeddings.** Intuitively, node embeddings [6] map nodes to a high dimensional vector space so that the likelihood of preserving node neighborhoods is maximized. This is achieved by performing random walks starting from each node to define an appropriate neighborhood. Popular node embeddings are often based on the skip-gram model, since it maximizes the probability of observing a node’s neighborhood given its embedding. By varying the type of random walks used, one can obtain diverse types of embeddings.

**Relational Embeddings.** Recently, there has been an uptick in research on using embeddings for relational data. Termite [3] seeks to project tokens from structured and unstructured data into a common representational space that could then be used for identifying related concepts through its Termite-Join approach. In contrast, we focus on local embeddings, learned from the structured data only, and ensure that they are effective for downstream data curation tasks. Other approaches seek to produce relational embeddings that could be used to support SQL queries with semantic similarity [1, 7]. All of the prior approaches rely on viewing the tuple as a textual document.

**Task Agnostic vs Task Specific Embeddings.** Most algorithms for word [12] and node embeddings [6] are defined in a task agnostic manner and trained using an unsupervised approach. This makes them generic and allows the learned embeddings to be used in a wide variety of downstream tasks [10]. In a data integration context, this is also desirable as the downstream tasks are often quite varied and training an embedding in a task specific manner is cumbersome.

### 3 Motivating Example

In this section, we provide an illustrative example that highlights the weaknesses of current approaches and motivates us to design a new approach for relational embedding.

Consider the case where pre-trained embeddings from popular approaches such as word2vec, fastText, ELMo, BERT are utilized. Figure 1 shows the filtered vector spaces for the tokens in an example with two small customer datasets. We see that the pre-trained embeddings suffer from a number of issues when we try to use them to model information in the relations. First, a number of words, such as Rick, are present in the dataset but not in the pre-trained embedding. This is especially problematic for enterprise datasets where tokens are often unique and not found in pre-trained embeddings. Second, such embeddings might contain geometric relationships that exist in the corpus they were trained on, but that are missing in the relational data. For example, the embedding for token Steve is closer to tokens iPad and Apple even though such a relationship is not implied in the data. Third, relationships that do occur in the data such as between tokens Paul and Mike are not observed in the pre-trained vector space.

Naturally, learning local embeddings from the relational data often produces better results. However, computing embeddings for non-integrated data sources is a non-trivial task.
Figure 1: Illustration of a simplified vector space learned from text (prior approaches) and from data (EMBDI).

This becomes especially challenging in settings where data is scattered over different datasets with heterogeneous structures and only partially overlapping content. Prior approaches express such datasets as sentences that can be consumed by existing word embedding methods. However, we find that these solutions are still sub-optimal for downstream data integration tasks.

**Technical Challenges.** We enumerate four challenges that must be overcome to obtain effective embeddings.

*Incorporating Relational Semantics.* Relational data exhibits a rich set of semantics. Relational data also follows set semantics where there is no natural ordering of attributes. Representing the tuple as a single sentence is simplistic and often not expressive enough for these signals.

*Handling Lack of Redundancy.* A key reason for the success of word embeddings is that they are trained on large corpora where there are adequate redundancies and co-occurrence to learn relationships. However, databases are often normalized to remove redundant information. This has an especially deleterious impact on the quality of learned embeddings. Rare words that are very common in relational data are typically ignored by word embedding methods. However, this pre-processing is impractical for databases.

*Handling Multiple Datasets.* In a setting consisting of multiple heterogeneous sources, we cannot assume that they have the same set of attributes or that there is sufficient overlapping values in the tuples, or even that there is a common dictionary for the same attribute.

*Handling Hierarchical Data.* Databases are inherently hierarchical, with entities such as cell values, tuples, attributes, dataset and so on. Data integration often involves identifying similarities between two entities such as attributes (schema mapping) or tuples (entity resolution). Incorporating these hierarchical units as first class citizens in learning embedding is a major challenge.

### 4 Overview of Our Approach

In this section, we provide a description of our approach and how these design choices address the aforementioned technical challenges.

Our proposed system, EMBDI, consists of three major components, as depicted in the right hand side of Figure [1]

1. In the **Graph Construction** stage, we process the relational data and transform it to a compact tripartite graph that encodes
its relationships. Tuple and attribute ids are treated as first class citizens.

2. Given this graph, the next step is Sentence Construction by performing biased random walks. These walks are carefully constructed to avoid common issues such as rare words and imbalance in vocabulary sizes. This produces as output a series of sentences.

3. Embedding Construction Once the random walks are performed, the corresponding sentences are passed to an algorithm for learning word embeddings. We are agnostic to the algorithm used.

4.1 Graph Construction

Why construct a Graph? In most of the prior approaches for local embeddings each tuple in a relation is modeled as a sentence that becomes the corpus, which is then used to train the embedding. While this provides promising results, it also ignores various signals that are inherent in relational data. Instead, we model relational data as a graph with sentences created through random walks that provide a diverse notion of neighborhood. With multiple relations, a unified graph enables a view over the different datasets that is invaluable for learning embeddings for data integration.

Simple Approaches. Consider a relation $R$ with attributes $\{A_1, A_2, \ldots, A_m\}$. Let $t$ be an arbitrary tuple and we denote by $t[A_i]$, the value of attribute $A_i$ for tuple $t$. A naive approach is to create a chain graph where tokens corresponding to adjacent attributes such as $t[A_i]$ and $t[A_{i+1}]$ are connected. This will result in $m$ edges for each tuple. Of course, if two different tuples share the same token, then they will reuse the same node for the token. However, relational algebra is based on set semantics where the attributes do not have an inherent order. So simplistically connecting adjacent attributes is doomed to fail. Another extreme is to create a complete subgraph where an edge exists between all possible pairs of $t[A_i]$ and $t[A_{i+1}]$. Clearly, this will result in $\binom{m}{2}$ edges per tuple. There are two issues with this approach. The number of edges is quadratic in the number of attributes – yet this approach ignores other relationships such as belonging to the same attribute.

Relational Data as Heterogeneous Graph. We propose a heterogeneous graph with 3 types of nodes. Token nodes correspond to the unique values found in the dataset. Record Id nodes (RIDs) represent a unique token for each tuple. Column Id nodes (CIDs) represent a unique token for each column/attribute. These nodes are connected by edges based on the structural relationships in the schema. Numerical values get a token like categorical ones, while null values are not represented in the graph.

Consider a tuple $t$ with RID $r_t$. Then, nodes for tokens corresponding to $t[A_1], \ldots, t[A_m]$ are connected to the node $r_t$. Similarly, all the tokens belonging to a specific attribute $A_i$ are connected to the corresponding CID, say $c_i$. Once again, tokens that are identical reuse the same node. This construction is generic enough that it could be augmented with other types of relationships such as integrity constraints. Note that a token could belong to different record ids and column ids when two different tuples/attributes share the same token.

Figure 2 shows a graph constructed for the datasets in Figure 1. Note that this could be considered as a variant of tri-partite graph. A key advantage of this choice is that it has the same expressive power as the complete subgraph approach while requiring orders of magnitude less edges.

Node Merging Our graph representation allows one to incorporate external information
such as wordnet or other domain specific dictionaries in a seamless manner. This is an optional step to improve the quality of embeddings. For example, consider two attributes from different relations – one stores country codes while the other stores complete country names. If some mapping between these two exists, then we can merge the nodes corresponding to, say, Netherlands and NL. The same reasoning applies to tuples (attributes): if trustable information in available about possible token matches, we merge different RIDs (CIDs) in the same node. While in our experiments we merge only tokens based on equality, this task can be improved with external functions, such as matchers based on syntactic similarity, pre-trained embeddings, or clustering, to increase the number of overlapping tokens across datasets.

4.2 Sentence Construction

Graph Traversal by Random Walks. Starting from each token node, one can perform multiple random walks over the graph. Each random walk will correspond to a sentence. The algorithm for learning embedding will use the corpus of such sentences for identifying the contextual neighborhood. Using graphs and random walks allows us to have a rich and diverse set of neighborhoods. Our approach is agnostic to the specific type of random walk used. In this paper, we use naive random walks, as we have observed marginal improvements with more complex, higher order walks (e.g., [6]).

From Walks to Sentences. Once a random walk is completed, we transform it into a sentence by dumping the information stored in each node. The resulting sentence could look like \( R_a T_b R_c T_d C_e T_f C_g T_h \), where \( T_\ast, R_\ast, C_\ast \) correspond to nodes of type tokens, record ids, and column ids, respectively. It is important to note that the random walks include nodes corresponding to RIDs and CIDs. By treating these as first order citizens, we imbibe a richer notion of similarity between these entities and they are naturally compared with similarity in the vector space as any other token. For example, two nodes corresponding to different attributes might co-occur in many random walks, resulting in embeddings that are closer to each other. This implies that these two attributes probably represent similar information.

The sentences from each random walk provides additional information about the neighborhood of each node that is not easy to obtain purely using the structured data format. We obtain better quality embeddings through the process of translating the tuple into sentence by using the graph as an intermediary instead of doing it in simpler fashion (e.g., with permutations) as in prior approaches.

Handling Imbalanced Relations. In order to build relational embeddings for a single relation, one can simply perform multiple random walks from each token node. This approach directly ameliorates the issue of infrequent words that plagues word embedding approaches. The number of random walks starting from nodes corresponding to both frequent
and infrequent tokens are equal. However, this approach is not appropriate when multiple relations are involved. For example, one relation could contain much more nodes than the other. In such circumstances, we found that an effective heuristic is to start random walks only from nodes that co-occur in both datasets. Even with datasets with a minimum amount of overlap (less than 2%), this approach ensures adequate coverage of all nodes and minimizes the issues due to relation imbalance.

The overlapping tokens are the bridge between the two datasets to be integrated. To maximize their impact in the embedding creation, we start every sentence with a RID or CID, randomly picked from those connected to the token at hand. This small change in the random walk creation affects the results as create evidence of similarity for the corresponding rows and columns.

**Example.** Assume node token $T_a$ that appear in two rows $R_a$ and $R_b$ over two large datasets. Since the token is rare, it will appear most likely only once as the first node in the walk, therefore the embedding algorithm will only see it in few patterns, such as $T_aR_bT_c$ or $T_aC_dT_e$. To improve the modeling of the $T_a$ we start the sentence with a RID or CID connected to $T_a$, such as $C_dT_aC_c$ and $R_aT_aR_b$. This way, even if the token is rare, it gives strong signals that the attributes and the row that contain it are related.

### 4.3 Embedding Construction

The sentences from the random walks are pooled together to get a corpus that is used to learn the embeddings. Our approach is agnostic to the actual word embedding algorithm used. We instead piggyback on the plethora of effective algorithms such as word2vec, GloVe, fastText, ELMo, BERT, and so on. Every year, the embedding quality of these algorithms increases dramatically and this has a transitive effect on our approach. Broadly, these techniques can be categorized as word based (such as word2vec) or character based (such as fastText). The latter is able to learn embeddings for characters, substrings such as prefixes and suffixes that could be effective in certain scenarios. Embedding algorithms rely on certain hyperparameters such as the learning method (either CBOW or Skipgram), dimensionality of embeddings and size of window used for prediction. We discuss these in more details in Section 5.

### 4.4 Using Embeddings for Integration

We now describe unsupervised algorithms that integrate data by making use of embeddings. We consider two popular tasks: schema matching and entity resolution. The same intuitions apply for matching both attributes and tuples, but the algorithms exploit specific properties of the two problems. We start with schema matching as this is a pre-requisite when matching records across datasets.

Assume the embeddings have been created for tokens, record IDs, and columns ID in two datasets $D_1$ and $D_2$. The datasets can have a different number of tuples and attributes and are partially overlapping in terms of tokens. In both tasks, we do not make decisions based on the absolute distance between elements because it is hard to have a generic threshold that works for every pair of datasets.

**Schema Matching (SM).** Traditional approaches rely on grouping attributes based on the value distributions or use other similarity measures. Recently, [4] used embeddings for identifying relationships between attributes using both syntactic and semantic similarities. However, they use embeddings only on attribute/relatin names and do not consider the instances – i.e. values taken by the attribute. We present a simple method to perform schema
mapping between two attributes by computing the distance between their corresponding embeddings.

For every $C_i$ in the set of CIDs $C_1$ for the dataset $D_1$, we create a list with all CIDs $C_j \in C_2$ (from $D_2$) in increasing order of distance. We do the same for every $C_j \in C_2$ with the resulting list of CIDs for $D_1$ in increasing order of distance. The algorithm then relies on the assumption that for every attribute in a dataset, there is at most one attribute matching in the other dataset. We consider the union of all CIDs (with their corresponding lists) in a pool of candidates. We pick one CID at random from the pool, say $C_x$ from $D_1$, and the first CID in its list, say $C_y$ from $D_2$. If $C_x$ is also the first element in the list for $C_y$, then they are matched and removed from the candidate pool, otherwise we remove $C_x$ from $C_y$’s list and $C_y$ from $C_x$’s list. The process continues until we have an empty pool, all candidates have empty lists, or we have done two loops over the candidates.

**Entity Resolution (ER).** Recent works used pre-existing embeddings for representing tuples \cite{2, 9}. However, our approach uses explicit RIDs as nodes that allows one to learn better embeddings from the data itself. This information is used to perform unsupervised ER by computing the distance between RIDs. We will also discuss in the experiments how one can also piggy back on prior supervised approaches by passing the learned embeddings as features to \cite{2, 9}.

For every RID $R_i$ in the set of RIDs $R_1$ in dataset $D_1$, we identify the set of closest $n$ RIDs in the vector space. The first RID element from $D_2$ in such set is considered ER match. In both algorithms, there will be many elements (either RIDs or CIDs) that have no matches in the other dataset. However, RIDs can match more than one RID.

## 5 Experiments

In this section we show that (i) local embeddings correctly model the relationships in the datasets, (ii) such embeddings are effective in data integration tasks. We discuss the impact of system parameters at the end of the section.

**Datasets.** We used two datasets pairs (scenarios) from the literature \cite{5, 2} and a dataset pair we created starting from open data \footnote{Imdb: https://www.imdb.com/interfaces/ \ MLen: https://grouplens.org/datasets/movielens/}. Details for the scenarios are in Table 1 notice that for the RefL scenario less than 1.8% of the distinct data values are overlapping across the two datasets.

**Metrics.** We measure the quality of the results w.r.t. to hand crafted ground truth for each task with precision, recall, and their combination with the F-measure.

**Execution Setting.** Experiments have been conducted in a laptop with a CPU Intel i7-8550U, 8x1.8GHz cores and 32GB RAM. Our algorithms are written in Python. Code available at https://gitlab.eurecom.fr/cappuzzo/ember.

### 5.1 Evaluating Embeddings Quality

We created three kinds of tests to measure how well embeddings model the data. Each test has a set of tokens from the dataset as input and the goal is to identify which token does not belong to the set (function `doesnt_match` in Python library *gensim*). For the first kind, tests *MatchAttribute* (MA), we randomly sample four values from an attribute and a fifth value from a different attribute in the same dataset, e.g., given (Rambo III, The matrix, E.T., A star is born, M. Douglas) the test is passed if M. Douglas is identified. In *MatchRow* (MR), we pick all tokens from a row
and replace one of them at random with a value from a different row, e.g., (S. Stallone, Rambo III, 1974, P. MacDonald). Finally, in Match-Concept (MC), we model more subtle relationships. For a random element $x$ on attribute $A_1$, we identify all tuples $T$ with such value ($A_1 = x$), we take three distinct values in $T$ from another attribute $A_2$ and we finally add a random value $y$ from $A_2$. The test is passed if $y$ is identified as unrelated from the other tokens, e.g., (Q. Tarantino, Pulp fiction, Kill Bill, Jackie Brown, Titanic). We manually integrated the datasets and created 256K tests for Movie, 40K for RefS, and 21K for RefL.

We define a baseline method, namely Basic, to create local embeddings without our contributions. We fixed the size of the sentence corpus for the baseline to contain the same number of tokens of EmbDI corpus. Then, we split equally this “budget” between sentences containing permutations of row tokens and sentences with samples of attribute tokens.

We report the quality results in Table 2 where each number represents the fraction of tests passed. While on average the local embeddings for EmbDI are largely superior to the baseline, our solution is beaten once for MA. By increasing the percentage of row permutations in Basic, results for MR improve but decrease for MA, without significant benefit for MC. This shows that random walks are needed to capture more complex relationships that are not modelled by row and attribute co-occurrence. We show next how this has a great impact on the quality of the integration tasks.

### 5.2 Data Integration

We conduct our experimental evaluation by testing for each scenario both schema matching and entity resolution.

**Schema Matching.** We test an unsupervised setting using (1) the algorithm proposed in Section 4.4 with EmbDI local embeddings, and (2) an existing matching system [4] with both pre-trained embeddings (Seep$P$) and our local embeddings (Seep$L$). Pre-trained embeddings for tokens and tuples have been obtained from Glove [2].

Table 3 reports the results w.r.t. manually defined attribute matches. The simple unsupervised method with EmbDI local embeddings outperforms the baseline in terms of F-measure in all scenarios. Results of RefS are the best because of the high overlap between its datasets. The baseline improves when it is executed with EmbDI local embeddings, showing their superior quality w.r.t. pre-trained ones. The Basic local embeddings lead to 0 attribute matches.

We also observe that results for Seep-PreTrain depend on the quality of the original attribute labels. If we replace the original (expressive and correct) labels with synthetic ones, Seep-PreTrain obtains F-measure values between .30 and .38. Local embeddings from EmbDI do not depend on the presence of the attribute labels. Finally, we tested also a traditional instance-based schema matcher that does not use embeddings [8] and its results are lower than the ones obtained by EmbDI in all scenarios.

**Entity Resolution.** For ER, we study both
unsupervised and supervised settings. To enable baselines to execute this scenario, we aligned the attributes with the ground truth. EMBDI can handle the original scenario where the schemas have not been aligned with a limited decrease in ER quality. As baseline, we use our unsupervised algorithm with EMBDI embeddings and pre-trained embeddings. We also test our local embeddings in the supervised setting with a state of the art ER system (DeepER), comparing its results to the ones obtained with pre-trained embeddings (DeepER). We set \( n = 10 \) in our experiments, by varying \( n \) we observe the expected trade-offs between \( P \) and \( R \).

Results in Table 4 show that EMBDI embeddings obtain better quality results in all scenarios in both settings. As observed in the SM experiments, using local embeddings instead of pre-trained ones increases significantly the quality of an existing system. In this case, supervised DeepER shows an average 6% absolute improvement in F-measure in the tested setting with 5% of the ground truth passed as training data. The improvements decreases to 4% with more training data (10%). Also for ER, the local embeddings obtained with the basic method lead to 0 row matched.

### 5.3 Ablation Analysis

**Parameters.** Several parameters in EMBDI affect the quality of the local embeddings. All the results reported in this section has been obtained with a single configuration, but the quality of the results for the different tasks increases significantly by tuning the parameters for the specific tasks.

The default configuration uses walks (sentences) of size 60, 300 dimensions for the embeddings space, and the Skip-Gram model in Word2Vec with a window size of 3. Executing the ER task with CBOW increases Movie’s F-measure to 0.93 (from 0.78) and RefL to 0.88 (from 0.80). Similarly, decreasing the size of the walks to 5 for the SM task raises the F-measure for RefL to 0.92 (from 0.77).

A larger corpus leads to better results in general, but we empirically identified that there is diminishing return after a certain size. As a rule of thumb, we fix the total number of tokens in the corpus with the following formula: \#corpus tokens=(\#dist.values+\#rows)*1000. The number of walks is derived by dividing the number of tokens by their size.

**Execution times.** Computing local embeddings takes about 17 minutes for Movie, 24 for RefS, and 25 for RefL. The execution times for the embeddings creation for the sentences de-

---

### Table 2: Quality results for local embeddings.

|          | EMBDI | SEEPL | SEEPP |
|----------|--------|--------|--------|
| **P**    | **R**  | **F**  | **P**  | **R**  | **F**  |
| Movie    | .75    | .86    | .80    | .75    | .75    |
| RefS     | 1      | 1      | 1      | .75    | .75    | .75    |
| RefL     | .71    | .83    | .77    | .50    | .75    | .60    | .63    | .88    | .73    |

### Table 3: Quality results for SM.

|          | EMBDI | SEEPL | SEEPP |
|----------|--------|--------|--------|
| **P**    | **R**  | **F**  | **P**  | **R**  | **F**  |
| Movie    | .80    | .54    | .41    | .59    | .55    | .80    | .90    | .75    |
| RefS     | .39    | .40    | .24    | .34    | .90    | .59    | .62    | .70    |
| RefL     | .63    | .50    | .29    | .47    | .63    | .69    | .78    | .70    |
pends drastically on the algorithm used and its configuration (CBOW is much faster than Skip-Gram). In the experiments with the default configuration, the embedding creation takes about 30% of the time, while both graph and sentence creation take about 35% of the total time.

6 Next Steps

Thanks to the proposed techniques to generate sentences, EmbDI is able to learn local embeddings of high quality, which have proven to be very effective for data integration. Looking forward, we are studying how to combine pre-trained and local embeddings. We aim at making use in a principled way of the high quality, generic embeddings provided by internet giants in the creation of our local, data specific representations. We also aim at exploring the use of local embeddings in applications beyond data integration, such as recommender systems and data exploration.

References

[1] R. Bordawekar and O. Shmueli. Using word embedding to enable semantic queries in relational databases. In DEEM Workshop, page 5. ACM, 2017.

[2] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang. Distributed representations of tuples for entity resolution. PVLDB, 11(11):1454–1467, 2018.

[3] R. C. Fernandez and S. Madden. Termite: a system for tunneling through heterogeneous data. arXiv preprint arXiv:1903.05008, 2019.

[4] R. C. Fernandez, E. Mansour, A. A. Qahtan, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Seeping semantics: Linking datasets using word embeddings for data discovery. In ICDE, 2018.

[5] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu. Corleone: hands-off crowdsourcing for entity matching. In SIGMOD, 2014.

[6] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In SIGKDD, pages 855–864. ACM, 2016.

[7] M. Günther. Freedly: Fast word embeddings in database systems. In SIGMOD, pages 1817–1819. ACM, 2018.

[8] B. Marnette, G. Mecca, P. Papotti, S. Rau nich, and D. Santoro. ++Spicy: an opensource tool for second-generation schema mapping and data exchange. PVLDB, 4(12):1438–1441, 2011.

[9] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. Deep learning for entity matching: A design space exploration. In SIGMOD, 2018.

[10] T. Schnabel, I. Labutov, D. Minno, and T. Joachims. Evaluation methods for unsupervised word embeddings. In EMNLP, pages 298–307, 2015.

[11] S. Thirumuruganathan, N. Tang, and M. Ouzzani. Data curation with deep learning [vision]: Towards self driving data curation. arXiv preprint arXiv:1803.01384, 2018.

[12] J. Turian, L. Ratini, and Y. Bengio. Word representations: a simple and general method for semi-supervised learning. In ACL, pages 384–394. ACL, 2010.

|        | Unsupervised | Supervised |
|--------|--------------|------------|
| BASIC  |              |            |
| GLOVE  |              |            |
| EMBDI  |              |            |
| DeepER |              |            |
| DeepER |              |            |

Table 4: F-Measure results for ER.