Word embeddings approximate the category of elements of a database

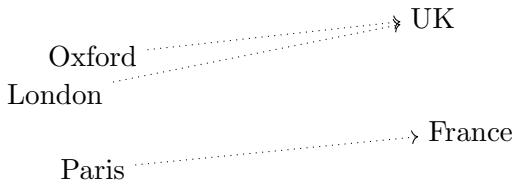
(draft)

Liang Ze Wong

May 10, 2019

1 Introduction

Word embedding algorithms take a corpus of text and embed its words as points in a high dimensional vector space [MCCD13]. A good embedding not only clusters semantically similar words together, but also renders similar *relationships* between words as almost parallel difference vectors between points. For instance, we might obtain an embedding of the form:



(1)

This paper is motivated by the following questions:

1. What kind of mathematical structure best describes the resulting embedding?
2. How do the existing algorithms produce such an embedding?
3. Why do the existing algorithms produce such an embedding?

This paper will *not* answer these questions, at least not in a rigorous sense. Instead, we will pose a different question regarding databases, then prove a purely category-theoretic result about discrete opfibrations that answers that question. We hope that this process will provide some insight into the three questions above, so that by the end of the paper, the following statements will sound like plausible answers to their respective questions:

1. An embedding aims to capture the structure of the *category of elements* of a functor $S \to \mathbf{Set}$.
2. Embedding algorithms work by embedding objects with similar *coslice categories* close together.
3. This works because for categories C satisfying some conditions, there exists a *discrete opfibration* $P \colon C \to S$ such that $Px = Py$ precisely when the coslice categories x/C and y/C are isomorphic.

2 The inverse problem for databases

We follow the account of databases and schemas given in [Spi12], which we briefly review here. A *database* is a functor $F\colon S \to \mathbf{Set}$, and the category S is the *schema* of the database.

We may think of a database as a collection of 2-column tables[1]. The objects of the schema S are column headers for tables within the database, and morphisms $f\colon s \to t$ in S are 2-column 'empty tables' with column headers s and t:

$$f = \begin{array}{c|c} s & t \\ \hline & \\ & \end{array}$$

The sets Fs are sets of values for column s, and the function $Ff\colon Fs \to Ft$ maps each value in column s to its corresponding value in column t. For instance, let $Fs = \{a, b, c\}$ and $Ft = \{x, y\}$, and suppose that $Ff(a) = x, Ff(b) = y, Ff(c) = x$. This may be expressed by filling in the rows of f to obtain the following table[2]:

$$f = \begin{array}{c|c} s & t \\ \hline a & x \\ b & y \\ c & x \end{array}$$

For an even more concrete example, consider the database consisting of a single table

$$\text{isIn} = \begin{array}{c|c} \text{City} & \text{Country} \\ \hline \text{London} & \text{UK} \\ \text{Paris} & \text{France} \\ \text{Oxford} & \text{UK} \end{array}$$

with schema S containing two objects 'City' and 'Country' and a single morphism

$$\text{isIn}\colon \text{City} \to \text{Country}.$$

Given a database $F\colon S \to \mathbf{Set}$, we may form its *category of elements* $\int F$ whose set of objects is the disjoint union of the sets Fs for all $s \in S$, and whose morphisms are

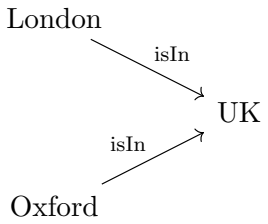
$$\int F(x, y) := \{f\colon s \to t | Ff(x) = y\}, \quad x \in Fs, y \in Ft.$$

The information of the resulting category $\int F$ may equivalently be expressed in terms of *resource descriptive framework (RDF) triples* of the form (x, f, y), where f is a morphism

[1] More general databases may contain tables with more than 2 columns, but such databases can always be factored into a collection of 2-column tables.

[2] Technically, we should call the resulting table Ff, but we abuse notation slightly by calling both the empty and filled tables f.

from x to y in $\int F$. In our example above, the category $\int F$ looks something like:



(2)

$$\text{Paris} \xrightarrow{\text{isIn}} \text{France}$$

The RDF triple store that represents this category is the collection of triples (London, isIn, UK), (Paris, isIn, France) and (Oxford, isIn, UK).

In this example, it is clear that $\int F$ possesses more structure than merely being a category. For instance, there are three morphisms labelled 'isIn', and we want to formalize the idea that these morphisms all arise from the same morphism in the schema S, i.e. that they come from the same table in the database. We may express this by saying that we have a functor $P\colon \int F \to S$ sending each object to its column (e.g. $P(\text{London}) = \text{City}$) and each morphism to its table (e.g. all instances of 'isIn' in $\int F$ are sent to 'isIn' in S). But we can say even more: this functor has the special property of being a *discrete opfibration*.

Definition 2.1. A **discrete opfibration** is a functor $P\colon C \to S$ such that for all $f\colon s \to t$ in S and $c \in P^{-1}(s)$ (i.e. $Pc = s$), there exists a unique lift of f whose domain is c.

In our example, the unique lift of City $\xrightarrow{\text{isIn}}$ Country with domain 'London' is the morphism London $\xrightarrow{\text{isIn}}$ UK, while the unique lift with domain 'Oxford' is the morphism Oxford $\xrightarrow{\text{isIn}}$ UK. Note that lifts of City $\xrightarrow{\text{isIn}}$ Country are only unique for a given domain; there are multiple lifts of City $\xrightarrow{\text{isIn}}$ Country with *codomain* 'UK'.

It turns out that the data of the category of elements $\int F$ and the discrete opfibration $P\colon \int F \to S$ are enough to recover the original database $F\colon S \to \mathbf{Set}$ that we started with. The following classical result captures this idea:

Proposition 2.2. *There is an equivalence of categories*

$$\int : \mathbf{Fun}(S, \mathbf{Set}) \cong \mathbf{DiscOpFib}/S,$$

where $\mathbf{DiscOpFib}/S$ *is the category of discrete opfibrations over* S*.*

In other words, a database may be completely recovered from its RDF triple store, *provided we know the columns and tables that each triple belongs to.*

But what if have an RDF triple store without the information of which columns and tables each triple belongs to, or even what the schema looks like? Can we construct a schema and database that for our RDF triple store? This may be formalized as:

Question (The inverse problem for databases)**.** *Given a category* C*, is there a category* S *and a functor* $F\colon S \to \mathbf{Set}$ *for which* $C = \int F$*? Equivalently, is there a discrete opfibration* $P\colon C \to S$*?*

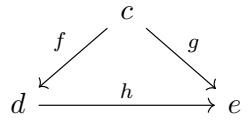
If we treat a category C with a discrete opfibration $P\colon C \to S$ as a form of *structured* data, with the structure provided by the schema S, then a category C on its own is a form of *unstructured* data. The inverse problem for databases then asks if we can extract a structure by which to organize the data in C.

As stated, the answer to the question is always "Yes", since we may simply take $S = C$ and $P = 1_C$. In other words, every C is a schema for itself. We would like to know if there are non-trivial solutions to this problem. On the other hand, it may happen that there are no non-trivial schemas for a particular category C. In that case, we would like some criteria on C that indicates this.

In the next section, we prove the main result of this paper and answer the inverse problem for databases.

3 Minimal discrete opfibrations under a category

We first give a quick overview of coslice categories, which feature prominently in the main result. For a given object c in a category C, the *coslice category* c/C is the category whose objects are morphisms in C with domain c, and whose morphisms are commuting triangles. Thus, in the following commuting diagram, f and g are objects of c/C and h is a morphism from f to g:



Any $f\colon x \to y$ in C induces a functor $- \circ f\colon y/C \to x/C$. Any functor $F\colon C \to D$ also induces a functor $c/C \to Fc/D$. We note some well-known results concerning these functors and isomorphisms of coslice categories:

Lemma 3.1. *Let $\varphi\colon x/C \cong y/C$ be an isomorphism of coslice categories. For $f\colon x \to u$, let v be the codomain of $\varphi(f)$ (so we have $\varphi(f)\colon y \to v$). Then φ induces an isomorphism $u/C \cong v/C$ making the following diagram commute:*

$$\begin{array}{ccc} u/C & \xrightarrow{\cong} & v/C \\ {\scriptstyle -\circ f}\downarrow & & \downarrow{\scriptstyle -\circ\varphi(f)} \\ x/C & \xrightarrow[\cong]{\varphi} & y/C \end{array}$$

Lemma 3.2. *Let $P\colon C \to S$ be a functor. Then P is a discrete opfibration if and only if the induced functor $c/C \to Pc/S$ is an isomorphism of categories for every $c \in C$.*

As an immediate consequence, we have a necessary condition for two objects of C to be identified by a discrete opfibration:

Corollary 3.3. *Let $P\colon C \to S$ be a discrete opfibration, and suppose we have $c, d \in C$ such that $Pc = Pd$. Then c/C and d/C are isomorphic.*

The main theoretical result of this paper is that the converse holds under certain conditions on C:

Proposition 3.4. *Let C be a skeletal category such that each x/C has no non-identity automorphisms. Let c and d be objects in a category C such that c/C and d/C are isomorphic. Then there exists a discrete opfibration $P\colon C \to S$ such $Pc = Pd$.*

The proof of requires the following lemma:

Lemma 3.5. *Let C be a skeletal category such that each x/C has no non-identity automorphisms. Then:*

1. *Any isomorphism $c/C \cong d/C$ sends 1_c to 1_d;*
2. *For c, d such that $c/C \cong d/C$, there is a* unique *isomorphism $\varphi\colon c/C \to d/C$.*

Proof. For 1, since C is skeletal, each x/C has a unique initial object 1_x, and initial objects are preserved by isomorphisms. For 2, if $\varphi, \psi\colon c/C \to d/C$ were two distinct isomorphisms, then $\psi^{-1}\varphi$ would be a non-identity automorphism of c/C. □

Proof of Proposition 3.4. We will define a category S and a functor $P\colon C \to S$ with the desired properties. The objects of S are isomorphism classes of coslice categories, denoted $[c/C]$ for $c \in C$. For morphisms, let

$$S([c/C], [d/C]) := \coprod_{\substack{y \in C \text{ s.t.} \\ y/C \cong d/C}} C(c, y).$$

Note that an isomorphism $\varphi\colon x/C \cong c/C$ induces an isomorphism

$$\coprod_{\substack{y \in C \text{ s.t.} \\ y/C \cong d/C}} C(x, y) \cong \coprod_{\substack{z \in C \text{ s.t.} \\ z/C \cong d/C}} C(c, z),$$

although this isomorphism need not factor through the coproductants (i.e. $C(x, y)$ need not be isomorphic to $C(c, z)$ for any y and z, even if $y/C \cong z/C$). Thus the hom-sets in S are well-defined, up to isomorphism.

The identity on $[c/C]$ is $1_c \in C(c, c) \subset S([c/C], [c/C])$. To compose $f \in S([c/C], [d/C])$ with $g \in S([d/C], [e/C])$, suppose $f \in C(c, y)$, $g \in C(d, z)$ where $y/C \cong d/C$ and $z/C \cong e/C$. Let $\varphi\colon d/C \to y/C$ be the unique isomorphism, and let $\varphi(g)\colon y \to u$. Then $u/C \cong z/C \cong e/C$ by Lemma 3.1, so we may define the composite to be

$$g \circ f := \varphi(g) \circ f \in C(c, u) \subset S([c/C], [e/C]).$$

Item 1 of Lemma 3.5 ensures that composition is unital, while Item 2 ensures that it is associative. We thus have a category S.

It is easy to see that there is a functor $P\colon C \to S$ sending c to $[c/C]$ and $f\colon c \to d$ to itself, treated as an element of $S([c/C], [d/C])$. By construction, $Pc = Pd$ precisely when $c/C \cong d/C$. Given $x \in [c/C]$ (so $Px = [c/C]$) and $f \in C(c, y) \subset S([c/C], [d/C])$, we have a unique map $\varphi(f)\colon x \to u$ lifting f, where $\varphi\colon c/C \cong x/C$. Thus $P\colon C \to S$ is a discrete opfibration. □

Remark 3.6. *The hypotheses on C are only used to show that composition is unital and associative. In the absence of these hypotheses, one would need to coherently choose isomorphisms between coslice categories in order to have unital and associative composition. It is unclear if this can always be done, although the author believes so. After all, not all categories of elements satisfy the hypotheses of the theorem.*

Combining Corollary 3.3 and Proposition 3.4, we obtain:

Theorem 3.7. *Let C be a skeletal category such that each coslice c/C has no non-identity automorphisms. For objects $c, d \in C$, there exists a discrete opfibration $P\colon C \to S$ such that $Pc = Pd$ if and only if c/C and d/C are isomorphic.*

Thus, there is no non-trivial schema for a category C if all its coslice categories are pairwise non-isomorphic. For all other C satisfying the hypotheses of the theorem, we have described the construction of a minimal schema S for C.

4 Word embeddings

Finally, we return to the questions that motivated this paper.

By comparing the outputs of word embeddings (1) with the category of elements of a database (2), we see that word embeddings have many parallel difference vectors that reflect the same relationship, just like how the category of elements has many morphisms representing the same relationship.

In a word embedding algorithm, two words that have similar *relationships* to other words should be clustered close together. Note that this does *not* mean that these two words u, v have the same relationship to a given word w. Rather, if u is related to w in some way, and v is similar to u, then v should be related to some x in the same way, where x is not necessarily w! For instance, 'king' and 'kings' are related in the same way that 'queen' and 'queens' are related, not 'queen' and 'kings'.

Similarly, if two objects u, v of a category C have isomorphic coslice categories $u/C \cong v/C$, it does not mean that $C(u, w) \cong C(v, w)$ for each $w \in C$. Rather, every $u \to w$ has a counterpart $v \to x$ for some x that is not necessarily w. In fact, we could even have two arrows $u \rightrightarrows w$ with the same codomain w corresponding to two arrows $v \to x$ and $v \to y$ where $x \neq y$.

In both word embeddings and isomorphisms of coslice categories, it is the relationships themselves – independent of the objects that they map to or from – that are of primary importance.

Finally, by clustering words with similar relationships ('coslice categories') close together, the resulting word embedding achieves the structure of the category of elements of a database. Similarly, by identifying ('clustering') objects with isomorphic coslice categories, we obtain a schema and a database of which C is the category of elements.

The statements in this section are necessarily speculative and imprecise. After all, a corpus of text does not have the structure of a category. Nevertheless, we hope that the analogy we have drawn is strong enough to provide some insight into how and why word embedding algorithms work.

References

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, *Efficient estimation of word representations in vector space*, arXiv preprint arXiv:1301.3781 (2013).

[Spi12] David I Spivak, *Functorial data migration*, Information and Computation **217** (2012), 31–51.