Outline of a PAC learnable class of bounded functions including graphs

Kevin H. Xu[1]
Froglingo Development Association, 2306 Johnson Circle, Bridgewater, New Jersey, USA

Abstract. A graph $G = \{V, E\}$, with a finite number of vertices V and a finite number of edges E, has a property of a possibly infinite number of paths where a path is a sequence of vertices one walks along edges from one vertex to another. In this paper, we describe a graph as a function in the Enterprise-Participant (EP) data model and outline that a graph G, i.e., the function representing G, is Probably Approximately Correct (PAC) learnable, i.e., there is a learning algorithm in polynomial time that constructs edge representations in an EP database as a program *P* from a set of sampled paths and EP's built-in reduction rules enables the reasoning of global transitive relations of the graph: there is a path from *A* to *C* if there are two paths *A* to *B* and *B* to *C.* Such a reasoning is also called a prediction, where predicting that *A* to *C* is not a path is an error when *A* to *C* is actually a path and an edge along the path *A* to *B* or *B* to *C* was not fed to the algorithm as part of a sample path. We generalize this learnability to a class of bounded functions. It is a reminder that data can be better represented in an EP database than in a graph before being embedded in distributional semantic learning models.

Keywords. Learnability, computability, graph, transitive relation, bounded function

1. Introduction

Imagine a robot not built with any knowledge about a city and not deductively programmed to navigate the streets of the city is daily dropped off to a random location of the city and subsequently randomly walks a few blocks on local streets. Days later, if the robot is able to deductively construct a program based on what it had experienced such that the program can predict more routes than what the robot had experienced, e.g., predict *A* to *C* is a path if it experienced two separate paths *A* to *B* and *B* to *C*, then we say that the street network of the city, a graph represented by the program, is learnable. Such a learned program always correctly provides negative predictions, i.e., when there is no path from *A* to *C*. It correctly provides positive predictions almost all the time but a few exceptions, i.e., when there is a path from *A* to *C*. As time goes by, the more walks the robot has, the more precise the prediction of the learned program will be. This learnability example falls under the definition of learnability introduced by Valiant in [1], which was later called Probably Approximately Correct (PAC) learnability, abbreviated as learnability in this paper.

In this paper, we introduce a class of functions, called bounded functions including graphs, and show the class is PAC learnable. We demonstrate that databases from the

[1] Corresponding author: kevin@froglingo.com

Enterprise-Participant (EP) data model are programs that can be constructed by a learning algorithm to represent bounded functions. We say such bounded functions are learnable because of EP's reduction system that can, for example, calculate the global path *A* to *C* from the local paths *A* to *B* and *B* to *C* in a given graph. A thorough work on the computability and the corresponding PAC learnability of the bounded functions was provided in [2]. A separate discussion on a graph's learnability was covered in [3].

2. Related work

A graph's PAC learnability, according to the EP data model, means that the global transitive relations of a graph, e.g., *A* to *C*, are preserved from local transitive relations, e.g., *A* to *B* and *B* to *C*. Preserving the global transitive relations of a graph certainly helps to learn aggregates of a graph, such as nodes clustering, and ranking, link prediction [4]. Many attempts of using a statistical machine learning model to learn the global transitive relations in a graph have not been successful [5, 6, 7], and mathematically proven ineffective [8, 9, 10]. Conceptually, a symbolic representation for a graph has the global transitive relations available. But the technologies, such as Semantic Web's Linked Open Data [11, 12], Hadoop and Spark BigData systems with limited query languages available (www.ibm.com/think/insights/hadoop-vs-spark), to store a graph with billions of nodes do not have the global transitive relations available practically [7]. EP is a better approach to store billions and trillions of nodes in a graph because all data in an EP database are arranged under three types of trees [3], as we know that data in a tree structure is easier to be stored and to learn [13, 14, 15]. Representing a graph in EP with various transitive relation operators and embedding the EP data for the graph is potentially a better approach than representing and embedding the graph itself for statistical machine learning.

Another benefit of using EP (in distributional semantics learning models) is that non-cyclical related data can be expressed in a more efficient structure than a complex structure for cyclic data in EP. We know many entities in the world, such as streets in a city and social networks, are naturally abstracted as a graph. But there are many other entities, such as a car-component-subcomponent recursive containment relationship and the sequence of the words in a sentence of "Joann went to shopping", don't have to be abstracted as a graph but represented as a graph in the contemporary graph embedding technologies [6, 16]. Further an accumulation of many pieces of non-cyclic data in a graph often ends up with cyclic data, for example, "Joann loves Mike" and "Mike spoke to Joann", where each unique word is a node and each sentence sequence is a path from a node to another linked by edges. In EP, there are two ways to express a hierarchical or sequential data: for a sequence of "a b c", for example, EP can express it hierarchically as *a b c* or graphically as *a b*: = *b*; *b c*: = *c*. We would always choose the former for hierarchical data and the latter for naturally graphical data, particularly cyclical data. In either case, data in EP would never have a chance to form a cycle even for a cyclic graph.

Most significantly, the learnability of a graph is extended to an entire class of bounded functions, where a bounded function is an approximation to a partial recursive function and the union of all (an infinite number of) bounded functions is semantically equivalent to the Lambda Calculus (or equivalent to the Turing Machine) [2]. This conclusion potentially reopens the avenue of symbolic approaches to Natural Language Processing (NLP).

Distributional semantics, rooted with the distributional hypothesis [17, 18] and hallmarked with real-valued vector space to represent entities, with dimensionality reduction, and with similarity computation among entities in the Euclidean space, has been the dominant approach to NLP since the beginning of the 1990s. It has achieved great successes from question answering, document categorization, machine translation, to recent chatbots, especially when it was driven by powerful neural networks with backpropagation techniques. The distributional semantic approach has limits. It is a statistical nature of learning and completely disregards the logical nature of reasoning that has been so successful in symbolic computation. The distributional semantic approach to NLP has been eager to be injected with knowledge represented in a graph for a more satisfactory performance [4, 19, 20, 21]. This desire, however, will continue to face challenges as long as a graph embedding technique is not improved. When evaluating if distributional semantics is the ultimate solution for NLP by asking "Are we climbing the right hill?" in [22], as further resonated in [20], the authors answered: "No".

Valiant's theory of the learnable established a broad framework for machine learning that not only provides the theoretical foundation for the distributional semantics [23], but it also reserved a space for a different approach that could potentially bridge logical nature of reasoning with the statistic nature of learning, as it was intended by the framework itself [24 and Valiant's research interest described in https://valiant.seas.harvard.edu]. Many deductive approaches had been identified to exemplify this purpose, such as the Boolean expressions identified in [1, 24]. Unfortunately, the known learnable functions do not reach a desired expressive power to represent natural languages, as evidenced in the research efforts that resulted in logic programming such as Prolog as well as the projection, disjunction and decision list of size-limited graphs in [25] from the 1950s to the 1990s.

3. The EP data model

The Enterprise-Participant (EP) data model is a language system and equivalently a data structure with which an EP database can be constructed. The idea behind EP is that we treat all objects to be represented as functions. Given a function f that produces a value m when it is applied to an argument n, denoted as $f(n) = m$, let's think of an exercise in which we inventory the behavior of f in a database. We can rewrite $f(n) = m$ as $\{f\ n := m\}$, reading it as: applying f to n is assigned a value m. The set $\{f\ n := m\}$, called a database, is an approximation of f. When we apply f to an additional argument n', we would obtain a better approximation $\{f\ n := m, f\ n' := m'\}$ where $f(n') = m'$. In addition, m could be another function such that $m(p) = q$ for a given input p. So we can exhibit more properties of f with the accumulated approximation $\{f\ n := m, f\ n' := m', m\ p := q\}$ or equivalently $\{f\ n\ p := q, f\ n' := m'\}$. From the database $\{f\ n := m, f\ n' := m', m\ p := q\}$, we can derive: $(f(n))(p) = q$.

The EP data model is described as a language system ($\boldsymbol{F}$, $\boldsymbol{C}$, *null*, $\cdot$, (,), :=, D) where

1) $\boldsymbol{F}$ is a set of identifiers (function names),
2) $\boldsymbol{C}$ is a set of constants, disjoint from $\boldsymbol{F}$. It could include infinite domains such as strings, integers, reals, and timestamps. $\boldsymbol{C}$ includes a special constant *null*.
3) $\cdot$ is a binary operation that produces a set $\boldsymbol{E}$ such that

$$m \in \boldsymbol{F} \Rightarrow m \in \boldsymbol{E}$$

$$m \in \boldsymbol{E}, (n \in \boldsymbol{C} \cup \boldsymbol{E}) \Rightarrow (m \cdot n) \in \boldsymbol{E}$$

Here we simply write $(m \cdot n)$ as $(m\ \ n)$ and further $m\ n$ when $(m\ n)$ is implied, where m, n, and $m\ n$ are called a function, an argument, and the corresponding application. For a $x \in \boldsymbol{E}$, we call x a term. Given an application term $m\ n$, m and n are called proper subterms of $m\ n$, and $m\ n$ is also called a subterm of itself.

4) $:=$ is the Cartesian product $\boldsymbol{E} \times (\boldsymbol{C} \cup \boldsymbol{E})$, i.e., $:= \; = \boldsymbol{E} \times (\boldsymbol{C} \cup \boldsymbol{E})$. When a pair $(p, q) \in \;:=$, we denote it as $p := q$, which is called an <u>assignment</u>, where p and q are the <u>assignee</u> and <u>assigner</u> respectively.

5) D , called a <u>database</u>, is a finite set of terms and a finite set of assignments, i.e., $D \subset (\boldsymbol{E} \cup :=)$, such that for each assignment $p := q \in D$, where $p, q \in \boldsymbol{E}$, the following constraints are met:

1) p has only one assigner, i.e.,
   $p := q$ and $p := q' \in D \Rightarrow q \equiv q'$
2) A proper subterm of p cannot be an assignee, i.e.,
   $p := q \in D \Rightarrow \forall x \in \mathrm{SUB}^{+}(p)\ [\forall m \in \boldsymbol{E}\ [x := m \notin D]]$
3) q can not be an assignee, i.e.,
   $p := q \in D \Rightarrow \forall a \in (\boldsymbol{C} \cup \boldsymbol{E})\ [q := a \notin D]$

Identifiers are the most basic building blocks in EP. Like in programming languages, we can choose alphanumeric tokens as identifiers, such as *abc123*, *_abc*, and more commonly we take words from a natural language vocabulary as identifiers, such as *hello*, *John*, *sport*, *law*, and *person*.

A term is either an identifier $x \in \boldsymbol{F}$ or an application $x\ y \in \boldsymbol{E}$ where $x \in \boldsymbol{E}$, $y \in \boldsymbol{C} \cup \boldsymbol{E}$, such as $x\ x$, $x\ 3.14$, $(a\ b\ c)\ (d\ e\ 3\ (d\ t\ 3))$ are legitimate terms where $x, a, b, c, d, e, t \in \boldsymbol{F}$.

A term alone without an assignment is allowed to be in a database. When a term is in a database, its subterms are considered in the database as well.

By terms alone, we can represent containment relationships. For example, the hierarchical structure of geographical locations can be expressed: (*the United States of America*) (*New York State*) (*New York City*) *Manhattan;* (*Water Street 55*)*;* (*the United States of America*) *Florida Miami; France Paris;*

The data in a database D are arranged in three kinds of trees [3]. The tree structures in EP are the foundation for an efficient system implementation to host billions of records, e.g., nodes in a graph, and potentially for better embedding an EP database that represents a graph rather than the graph itself into a low-dimensional Euclidean space for distributional semantic learning models.

The terms embed transitive relations, such as we can infer *Miami* is part of *the United States of America* because *Miami* is part of *Florida* and *Florida* is part of *the United States of America*.

A set of reduction rules are available on a database. Given a database D, a term $n \in \boldsymbol{C} \cup \boldsymbol{E}$ is in <u>EP normal form</u> (or <u>normal form</u> in brief) if and only if

1) It is a constant $c \in \boldsymbol{C}$, or
2) It is a term $n \in D$ and n is not an assignee, i.e., $n \in D$ and $\forall b \in \boldsymbol{E}\ [n := b \notin D]$.

Let NF(D) denote the entire set of normal forms under a database D, where $null \in \mathrm{NF}(D)$ and other constants $c \in \mathrm{NF}(D)$ only if $c \in D$.

Given a database D, we have one-step *<u>reduction</u>* rules, denoted as $\rightarrow$:

1) An assignee is reduced to the assigner, i.e., $a := b \in D \Rightarrow a \rightarrow b$
2) An identifier not in the database is reduced to *null*, i.e., $a \in \boldsymbol{F}, a \notin D \Rightarrow a \rightarrow null$
3) If a and b are normal forms and $a\ b \notin D$, then $a\ b$ is reduced to *null*, i.e.,
   $a, b \in NF(D), a\ b \notin D \Rightarrow a\ b \rightarrow null$
4) $a \rightarrow a', b \rightarrow b' \Rightarrow a\ b \rightarrow a'\ b'$

If we have a sequence of reductions: $a \to a_0$, $a_0 \to a_1$, …, $a_{n-1} \to a_n$, where $n \geq 0$, we say that a is effectively, i.e., in finite steps, reduced to a_n, denoted as $a \to_D a_n$. If $a_1 \to_D b$ and $a_2 \to_D b$, then we say that b, a_1 and a_2 are equal (equivalence relation), denoted as $b == a_1 == a_2$. We also define $a == a$ for any term $a \in D$. Note that given an application, e.g., $a\ b\ c$, the sequence of the reductions toward its normal form is unique because the restricted syntactical form of an application only allows one unique reduction sequence, e.g., $a\ b\ c$ is restrictedly written as $((a\ b)\ c)$.

A term a has a normal form b if b is in normal form and $a \to_D b$. In [26], it has been proven that any term $a \in \boldsymbol{C} \cup \boldsymbol{E}$ under a database D has one and only one normal form and can be effectively reduced in finite steps. A constant, such as *null*, *3.14*, or “Hello”, is always in normal form.

With the EP reduction rules introduced, we give two sample databases representing graphs, from which we can intuitively see that an EP database represents a bounded function. First, we give a graph with a single directed link with two end vertices: $D = \{u_1\ u_2 := u_2\}$. EP has a reduction on D:

$u_1\ u_2 \to_D u_2$

Here the vertex u_1 can be viewed as a function that yields to u_2 when it is applied to u_2, which simulates that one from u_1 can walk over to u_2. Because the database is only defined with the single pair $\{u_1\ u_2 := u_2\}$, applying u_2 to anything else would yield to meaningless, denoted as *null* in EP:

$u_2\ y \to_D null$ for any y.

This reduction says that one from u_2 cannot reach out to anything else. In this database, NF(D) = $\{null, u_1, u_2\}$.

Now, let’s give another EP database representing a graph with a triangle: $D = \{v_1\ v_2 := v_2;\ v_2\ v_1 := v_1;\ v_2\ v_3 := v_3;\ v_3\ v_2 := v_2;\ v_3\ v_1 := v_1;\ v_1\ v_3 := v_3\}$, where each undirected edge is expressed by a pair of directed edges, for example, $v_1\ v_2 := v_2$ and $v_2\ v_1 := v_1$ for the edge between vertices v_1 and v_2. With the database D, the system has the following infinitely possible reductions:

$v_1\ v_2\ v_1 \to_D v_1$

$v_3\ v_2\ v_1\ v_2 \ldots v_1 \to_D v_1$

…

The sample reductions above simulate how one can walk from one vertex to another along the edges of the triangle. In this database, NF(D) = $\{null, v_1, v_2, v_3\}$.

Given a database D, there is a function Y_D: $\boldsymbol{E} \to$ NF(D) that is defined as:

$Y_D = \{(m, n) \mid m \in \boldsymbol{E}, n \in \text{NF}(D), \text{ and } m \to_D n\}$.

We call such a function Y_D bounded because $\boldsymbol{E}$ is infinite and NF(D) is finite. In addition, there is a subset $E' \subseteq \boldsymbol{E}$ such that

$$Y_D(x) = \begin{cases} y, \text{ where } y \in \text{NF}(D) \setminus \{null\} & \text{if } x \in E' \\ null & \text{if } x \in \boldsymbol{E} \setminus E' \end{cases}$$

We say Y_D has a finite support and simply call it finite if E' is finite. Given a database $D = \{a\ b\ c := d;\ d\ e := f;\}$, for example, we have a finite $E' = \{a, b, c, a\ b, a\ b\ c, d, e, d\ e, f, a\ b\ c\ e\}$ and NF(D) = $\{a, b, c, a\ b, d, e, f, null\}$. Y_D is bounded if E' is either finite or infinite. Given a database $D = \{a\ b\ c := c;\}$, as an additional example of cyclic data like a graph, we have an infinite $E' = \{a, b, c, a\ b, a\ b\ c, a\ b\ (a\ b\ c), a\ b\ (a\ b\ (a\ b\ c)), \ldots\}$ and NF(D) = $\{a, b, c, a\ b, null\}$. A finite function is bounded, but a bounded function may not be finite. The definition of a bounded function was originally introduced in [26] and further elaborated in [2]. Further, a bounded function Y_D is recursive, i.e., $Y_D(x)$ always terminates and yields to a normal form [26].

Because EP databases represent bounded functions including graphs, the bounded functions can be constructed by a learning algorithm in terms of the PAC learnability.

4. The class of bounded functions is PAC learnable

An EP database introduced in Section 3 can be generated from a partial computation on the closed lambda terms of the lambda calculus (or independently from an enumeration based on the data restrictions defined in Section 3) [26]. From a sequence of partial computations in the lambda calculus, where each partial computation is signified by a number of computation steps $s \in \boldsymbol{N}$ (the set of natural numbers), a corresponding sequence of databases can be generated, denoted as $D_0, D_1, \ldots, D_s, \ldots$ and rewritten the sequence as a set $\boldsymbol{D} = \{D_s \mid s \geq 0\}$, where the size of the database $|D_s|$, the number of assignments, is restricted by s, i.e., $|D_s| \leq s$. Accordingly, the class of bounded functions can be denoted as $\boldsymbol{Y} = \{Y_{Ds} \mid D_s \in \boldsymbol{D}$ and $s \geq 0\}$, where NF(Ds) $\leq s$.

When we say that the class of bounded functions $\boldsymbol{Y}$ is PAC learnable, we actually say that $\boldsymbol{Y}^s = \{Y_{Di} \mid D_i \in \boldsymbol{D}$ and $i \leq s\}$ for any $s \in \boldsymbol{N}$ is PAC learnable, where the size of the normal forms in D_i is restricted by s, i.e., $|\mathrm{NF}(D_i)| \leq s$, and equivalently, we say that a $Y_{Di} \in \boldsymbol{Y}^s$ is the target function for a learning algorithm to construct a program, e.g., another database $D_i' \subseteq D_i$, such that $Y_{Di'}$ approximates and converges to Y_{Di} when it receives more and more sample pairs (m, n) that are selected from Y_{Di}, i.e., $(m, n) \in Y_{Di}$, with an arbitrary probability distribution. In the rest of the section, we use D to denote a database D_i where $i \leq s$ for a given $s \in \boldsymbol{N}$.

As a law of the nature, the probability of discovering and constructing a full D from the provided sample pairs (m, n) is quantified by a function L(h, S) in [1], where h is an adjustable real number larger than 0 with which h^{-1} represents a probability and S is a positive integer representing the size of the program. L(h, S) is the smallest number of independent Bernoulli trials, after which a trial with a provided (m, n) has the probability of at least h^{-1} to successfully discover a new assignment. Further after L(h, S) independent Bernoulli trials, the probability of discovering all the assignments in D is at least 1- h^{-1}. The upper bound of the function L(h, S) is determined with a proposition [1]: For all integers $S \geq 1$ and all real $h > 1$, $\mathrm{L}(h, S) \leq 2h(S + \log_e h)$. (There are more accurate models to differentiate the first probability, h^{-1}, of discovering a new assignment from the second probability, $1- h^{-1}$, of discovering all the assignments. But it is sufficient conceptually to choose one number h for both probability for now in this paper.)

Given a database D, the target function Y_D to be learned has the size of D to be the size S of the program that represents Y_D, i.e., $S = |D|$ in $\mathrm{L}(h, S) \leq 2h(S + \log_e h)$, where $|D|$ denotes the size of D. Although the number of elements in Y_D can be infinite, the given database D, a finite set of assignments, determines the semantics of Y_D. Therefore, if we have a learning algorithm that constructs D, we say that we have PAC learned Y_D, i.e., the size $|D|$ of the program D that represents Y_D is proportional to the number of assignments in D.

We are going to give a learning algorithm that will call a routine EXAMPLES about L(h, $|D|$) times with a pre-determined h that would meet an expected precision, e.g., h^{-1} to measure how easily to discover a new assignment and $1- h^{-1}$ to measure how close D' to D would be after L(h, S) calls to EXAMPLES. When EXAMPLES is called, which pair (m, n) is chosen from Y_D is determined by a probability distribution P. Because Y_D

can have infinite number of pairs (m, n) and because D consists of a finite set of assignments that determines the entire (infinite) set of pairs (m, n) in Y_D, we practically have the following preferences on the set of pairs (m, n) that EXAMPLES would return and on the probability distribution P:

1) EXAMPLES only returns positive examples, i.e., returns $(m, n) \in Y_D$, where n is not *null*.
2) When EXAMPLES returns a pair (m, n), the size of m, i.e., $|m|$, does not need to be large. So we choose $|m|$ to be no more than the maximum size of assignees in D. This is a special requirement to the probability distribution P to ensure that the learning algorithm is effective when we choose $|D|$ to be the size S of the program to be learned.

Now let's give the learning algorithm: We first initiate an empty database $D = \{\}$. While the number of the calls to EXAMPLES has not reached $2h(|D| + \log_e h)$, call EXAMPLES to get a pair (m, n) and search D to find if m and n have already been served as the assignee and the assigner of an assignment in D.

1) if (m, n) has been in D already, do nothing.
2) if we found another assignment $m' := n'$ in D such that m is a left most subterm (lms) of m', i.e., $m' \equiv m\ q_1 \ldots q_k$ for some $k > 0$, then we delete $m' := n'$ from D and add $m := n$. For the pair ($n\ q_1 \ldots q_k$, n'), we recursively call Step 1 above.
3) if we found another assignment $m' := n'$ in D such that m' is a lms of m, i.e., $m \equiv m'\ q_1 \ldots q_k$ for some $k > 0$, then we add one assignment $n'\ q_1 \ldots q_k := n$.
4) if we did not find any assignment $m' := n'$ in D such that m is a lms of m' or m' is a lms of m, then we create $m := n$ into D.

The complexity of running the algorithm above is within a polynomial. For each loop in the algorithm with an input (m, n), the time complexity to search the term m is $O(\log(|D|))$, where EP terms are sorted in a database. Provided that the learning algorithm may needs to recursively search the database again in Step 2 above for investigating the pair ($n\ q_1 \ldots q_k$, n') and the number of the recursive calls may at the worst case reach the size of the term $n\ q_1 \ldots q_k$, i..e, $|n\ q_1 \ldots q_k|$ which is a constant in average, we add the time complexity $O(\log(|D|)) * |n\ q_1 \ldots q_k| \cong O(\log(|D|))$ times in each loop. Then, the time for each loop is $O(\log(|D|)) * (|n\ q_1 \ldots q_k| + 1) \cong O(\log(|D|))$. The time complexity for the total loops of $2h(|D| + \log_e h)$ is $2h\ (|D| + \log_e h) * O(\log(|D|))$. So the time complexity is $O(h\ |D| \log_e|D| + h \log_e h \log_e|D|)$, which is within polynomial. The space complexity is $|D|$, linear to the size of the database.

5. Conclusion

In this paper, we introduced a class of bounded functions including graphs that is PAC learnable. The learnability comes from the built-in reduction rules and subsequently the built-in transitive relations of the EP data model. We also reminded that an EP data model can better represent data than a graph does in general and potentially in statistical machine learning.

Acknowledgement The author would like to thank Dr. Prof. Anselm Blumer who had an extensive discussion with him on a general property of a PAC learnable class of functions that was discussed in an earlier version of this paper and provided substantial review notes that significantly improved the quality of the paper.

References

[1] L.G. Valiant. A Theory of the Learnable. Communication of the ACM, November 1984, Volume 27, Number 11.

[2] K. Xu. A class of bounded functions that approximate the lambda calculus is PAC learnable. Submitted to ALT 2025.

[3] K. Xu. A graph is PAC learnable. DOI: 10.13140/RG.2.2.27392.65282, September 2024.

[4] H. Cai, V.W. Zheng, K. Chang. A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. IEEE Transaction on Knowledge and Data Engineering, Sept. 2017.

[5] J. Berant, I. Dagan, J. Goldberger (2012). Learning entailment relations by global graph structure optimization. Journal of Computational Linguistics, 38(1):73-111.

[6] A. Bordes, J. Weston, R. Collobert, Y. Bengio (2011). Learning structured embeddings of knowledge bases. Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence.

[7] M. Nickel, V. Tresp, H. P. Kriegel. Factorizing YAGO – Scalable Machine Learning for Linked Data. WWW2012 – Session: Creating and Using Links between Data Objects. April 2012, Lyon, France.

[8] C. Seshadhri, A. Sharma, A. Stolman, A. Goel. The impossibility of low rank representation for triangle-rich complex network. The Proceedings of National Academy of Sciences, March 2020.

[9] R. Bhattacharjee, S. Dasgupta. What relations are reliably embeddable in Euclidean space? Journal of Machine Learning Research 1 (2019) 1-48.

[10] D. Korman, J. Jett, A. Renear. Defining textual entailment. Journal of the Association for Information Science and Technology, 2018.

[11] C. Bizer, T. Heath, and T. Berners-Lee. Linked data-the story so far. International Journal on Semantic Web and Information Systems, 5(3):1–22, 2009.

[12] H. Halpin, P. Hayes, J. McCusker, D. Mcguinness, and H. Thompson. When owl: sameAs isn't the same: An analysis of identity in linked data. The Semantic Web–ISWC 2010, page 305–320, 2010.

[13] J. Berant, I.Dagan, M. Adler, J. Goldberger. Efficient tree-based approximation for entailment graph learning. Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, pages 117-125.

[14] M. Nickel, D. Kiela. Poincaré embeddings for learning hierarchical representations. 2017.

[15] K. Xia, K.Z. Lee, Y. Bengio, E. Bareinboim. The Causal-Neural Connection: Expressiveness, Learnability, and Inference. 35th Conference on Neural Information Processing Systems (NeurIPS 2021)

[16] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in KDD, 2014, pp. 701–710

[17] J. Firth. A synopsis of linguistic theory 1930-1955. In Studies in Linguistic Analysis, Philological Society, Oxford. Reprinted in Palmer, F. (ed. 1968) Selected papers of J. R. Firth, Longman, Harlow.

[18] Z. Harris. Distributional structure. Word, 10(2-3): 146-162. https://doi.org/10.1080/00437956.1954.11659520

[19] M. Apidianaki. From word types to tokens and back: a Survey of approaches to word meaning representation and interpretation. Computational Linguistics (2023) 49 (2): 465–523.

[20] A. Lenci, M. Sahlgren, Distributional Semantics, Cambridge University Press, 2023.

[21] R. Patil, S. Boit, V. Gudivada, J. Nandigam. A survey of text representation and embedding technologies in NLP. IEEE Access.

[22] E.M. Bender, A. Koller. Climbing towards NLU: On meaning, form, and understanding in the age of data. Pages 5185–5198 of: Proc. ACL, 2020.

[23] A. Blumer, A. Ehrenfeucht, D. Haussler, M.K. Warmuth. Learnability and the Vapnik-Chervonenkis Dimension. Journal of the Association for Computing Machinery. Vol. 36. No. 4. October 1989, pp. 929-965.

[24] L.G. Valiant, J.C. Shepherdson. Deductive Learning, Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences. Vol. 312, No. 1522, Mathematical Logic and Programming Languages [Displayed chronologically; published out of order] (Oct. 1, 1984), pp. 441-446 (6 pages)

[25] P. Jappy, R. Nock. PAC learning conceptual graphs, ICCS 1998.

[26] K. Xu. A class of bounded functions, a database language and an extended lambda calculus. Journal of Theoretical Computer Science, Vol. 691, August 2017, Page 81 - 106.