Data Curation with Deep Learning

Saravanan Thirumuruganathan, Nan Tang, Mourad Ouzzani, AnHai Doan‡

Qatar Computing Research Institute, HBKU; ‡University of Wisconsin-Madison
{sthirumuruganathan, ntang, mouzzani}@hbku.edu.qa, anhai@cs.wisc.edu

ABSTRACT

Data curation – the process of discovering, integrating, and cleaning data – is one of the oldest, hardest, yet inevitable data management problems. Despite decades of efforts from both researchers and practitioners, it is still one of the most time consuming and least enjoyable work of data scientists. In most organizations, data curation plays an important role so as to fully unlock the value of big data. Unfortunately, the current solutions are not keeping up with the ever-changing data ecosystem, because they often require substantially high human cost. Meanwhile, deep learning is making strides in achieving remarkable successes in multiple areas, such as image recognition, natural language processing, and speech recognition. In this vision paper, we explore how some of the fundamental innovations in deep learning could be leveraged to improve existing data curation solutions and to help build new ones. We identify interesting research opportunities and dispel common myths. We hope that the synthesis of these important domains will unleash a series of research activities that will lead to significantly improved solutions for many data curation tasks.

1 INTRODUCTION

Data Curation (DC) [32, 44] – the process of discovering, integrating [46] and cleaning data (Figure 1) for downstream analytic tasks – is critical for any organization to extract real business value from their data. The following are the most important problems that have been extensively studied by the database community under the umbrella of data curation: *data discovery*, the process of identifying relevant data for a specific task; *schema matching and schema mapping*, the process of identifying similar columns and learning the transformation between matched columns, respectively; *entity resolution*, the problem of identifying pairs of *tuples* that denote the same entity; and *data cleaning*, the process of identifying errors in the data and possibly repairing them. These are in addition to problems related to outlier detection, data imputation, and data dependencies.

Due to the importance of data curation, there has been many commercial solutions (for example, Tamr [45] and Trifacta [23]) and academic efforts for all aspects of DC, including data discovery [8, 19, 33, 36], data integration [12, 27], and data cleaning [7, 16, 42]. An oft-cited statistic is that data scientists spend 80% of their time curating their data [8]. Most DC solutions cannot be fully automated, as they are often *ad-hoc* and require substantial effort to generate things, such as features and labeled data, which are used to synthesize rules and train machine learning models. Practitioners need practical and usable solutions that can significantly reduce the human cost.

Deep Learning (DL) is a successful paradigm within the area of machine learning (ML) that has achieved significant successes in diverse areas, such as computer vision, natural language processing, speech recognition, genomics, and many more. The trifecta of big data, better algorithms, and faster processing power has resulted in DL achieving outstanding performance in many areas. Due to these successes, there has been extensive new research seeking to apply DL to other areas both inside and outside of computer science.

Data Curation Meets Deep Learning. In this article, we investigate intriguing research opportunities for answering the following questions:

- What does it take to significantly advance a challenging area such as DC?
- How can we leverage techniques from DL for DC?
- Given the many DL research efforts, how can we identify the most promising leads that are most relevant to DC?

We believe that DL brings unique opportunities for DC, which our community must seize upon. We identified two fundamental ideas with great potential to solve challenging DC problems.

Feature (or Representation) Learning. For ML- or non-ML based DC solutions, domain experts are heavily involved in feature engineering and data understanding so as to define data quality rules. *Representation learning* is a fundamental idea in DL where appropriate features for a given task are automatically learned instead of being manually crafted. Developing DC-specific representation learning algorithms could dramatically alleviate many of the frustrations domain experts face when solving DC problems. The learned representation must be generic such that it could be used for multiple DC tasks.

DL Architectures for DC. So far, no DL architecture exists that is cognizant of the characteristics of DC tasks, such as representations for tuples or columns, integrity constraints, and so on. Naively applying existing DL architectures may work well for some, but definitely not all, DC tasks. Instead, designing new DL architectures that are tailored for DC and are cognizant of the characteristics of DC tasks would allow efficient learning both in terms of training time and the amount of required training data. Given the success of domain specific DL architectures in computer vision and natural language processing, it behooves us to design an equivalent of such a DL architecture customized for DC task(s).

Contributions and Roadmap. The major elements of our proposed approach to exploiting the huge potential offered by DL to tackle key DC challenges include:

- **Representation Learning for Data Curation.** We describe several research directions for building representations that are explicitly designed for DC. Developing algorithms for representation learning that can work on multiple modalities of data (structured, unstructured, graphical),

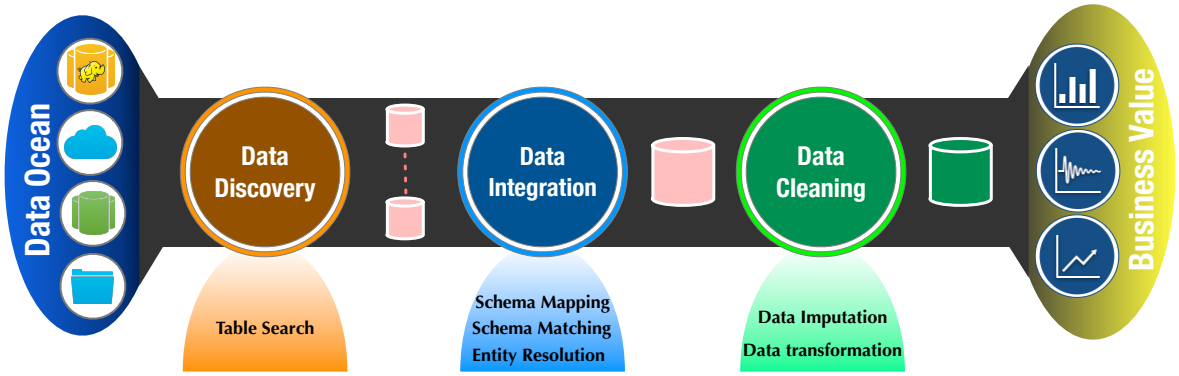

Figure 1: A Data Curation Pipeline

and can work on diverse DC tasks (such as deduplication, error detection, data repair) is challenging.

- **Deep Learning Architectures for Data Curation Tasks.** We identify some of the most common tasks in DC and propose preliminary solutions using DL. We also highlight the importance of designing DC specific DL architectures.
- **Early Successes of DL for DC.** In the last couple of years, there has been some promising developments in applying DL to some DC problems such as data discovery [19] and entity resolution [14, 35]. We provide a brief overview of these work and the lessons learned.
- **Taming DL's Hunger for Data.** DL often requires a large amount of training data. However, there are multiple promising techniques, which when adapted for DC, could dramatically reduce the required amount of labeled data.
- **Myths and Concerns about DL.** DL is still a relatively new concept, especially for database applications and is not a silver bullet. We thus identify and address a number of myths and concerns about DL.

2 DEEP LEARNING FUNDAMENTALS

We provide a quick overview of relevant DL concepts needed for the latter sections. Please refer to [5, 20, 29] for additional details.

Deep learning is a subfield of ML that seeks to learn meaningful representations from the data using computational models that are composed of multiple processing layers. The representations could be then used to solve the task at hand effectively. The most commonly used DL models are neural networks with many hidden layers. The key insight is that successive layers in this "deep" neural network can be used to learn increasingly useful representations of the data. Intuitively, the input layer often takes in the raw features. As the data is forwarded and transformed by each layer, an increasingly sophisticated and more meaningful information (representation) is extracted. This incremental manner in which increasingly sophisticated representations are learned layer-by-layer is one of the defining characteristic of DL.

2.1 Deep Learning Architectures

Fully-connected Neural Networks. The simplest model is a neural network with many hidden layers – a series of layers where each node in a given layer is connected to every other node in the next layer. They are also called feed-forward neural network. It can learn relationships between any two input features or intermediate representations. Its generality however comes with a cost; one has to learn weights and parameters which requires a lot of training data.

Domain Specific Architectures. There are a number of neural network architectures that are designed to leverage the domain specific characteristics. For example, Convolutional Neural Networks (CNNs) are widely used by the computer vision community. The input is fed through convolutions layers, where neurons in convolutional layers only connect to close neighbors (instead of all neurons connecting to all neurons). This method excels in identifying spatially local patterns and use it to learn spatial hierarchies such as nose → face → human. Recurrent Neural Networks (RNNs) are widely used in Natural Language Processing (NLP) and Speech recognition. RNN processes inputs in a sequence one step at a time. For example, given two words "data curation", it first handles "data" and then "curation". Neurons in an RNN are designed by being fed with information not just from the previous layer but also from themselves from the previous pass that are relevant for NLP.

2.2 Distributed Representations

Deep learning is based on learning data representations, and the concept of distributed representations (*a.k.a.* embeddings) is often central to DL. *Distributed representations* [24] represent one object by many representational elements (or many neurons). Each neuron is associated with more than one represented object. That is, the neurons represent features of objects. Distributed representations are very expressive and can handle semantic similarity. These advantages become especially relevant in domains such as DC. There has been extensive work on developing algorithms for learning distributed representations for different types of entities such as words and nodes.

Distributed Representations of Words (*a.k.a.* word embeddings) seek to map individual words to a vector space, which helps DL algorithms to achieve better performance in NLP tasks by grouping similar words. The dimensionality is often fixed (such as 300) and the representation is often *dense*. Word embeddings are often learned from the data in such a way that semantically related words are often close to each other. The geometric relationship between words often also encodes a semantic relationship between them. An oft-quoted example shows that by adding the vector corresponding to the concept of *female* to the distributed representation of *king*, we (approximately) obtain the distributed representation of *queen*.

Distributed Representations of Graphs (*a.k.a.* network embeddings) are a popular mechanism to effectively learn appropriate features for graphs (see [6] for a survey). Each node is represented as a dense fixed length high dimensional vector. The optimization objective is neighborhood-preserving whereby two nodes that are in the same neighborhood will have similar representations. Different definitions of neighborhood results in different representations for nodes.

Distributed Representations for Sets. Sets are a fundamental data structure where the order of items does not matter. Sets are extensively used in the Codd model where each tuple is a set of atomic units and a relation is a set of tuples. Algorithms for learning set embeddings must ensure that they have permutation invariance so that any permutation of the set should obtain the same embedding. There has been a number of popular algorithms for processing sets to produce such embeddings such as [41, 52].

Compositions of Distributed Representations. One can then use these to design distributional representations of atomic units – words in NLP, or nodes in a graph – for more complex units. For NLP, these could be sentences (*i.e.,* sentence2vec), paragraphs or even documents (*i.e.,* doc2vec) [28]. In the case of graphs, it can be subgraphs or entire graph.

3 REPRESENTATION LEARNING FOR DATA CURATION

In this section, we discuss potential research opportunities for designing DL architectures and algorithms for learning DC specific representations.

3.1 The Need for DC Specific Distributed Representations

Data curation is the process of discovering, integrating and cleaning data for downstream tasks. There are a number of challenging sub-problems in data curation. However, a number of problems could be unified through the prism of "matching".

Data Discovery is the process of identifying relevant data for a specific task. In most enterprises, data is often scattered across a large number of tables that could range in the tens of thousands. As an example, the user might be interested in identifying all tables describing user studies involving insulin. The typical approach involves asking an expert or performing manual data exploration [8]. Data discovery could be considered as identifying tables that *match* a user specification (*e.g.,* a keyword or an SQL query).

Schema Matching is the problem of identifying a pair of *columns* from different tables that are (semantically) related and provide comparable information about an underlying entity. For example, two columns *wage* and *salary* could store similar information. The problem of **schema mapping** seeks to learn the transformation between matched columns such as *hourly wage* and *monthly salary*. One can see that this problem seeks to identify *matching* columns.

Data Cleaning is the general problem of detecting errors in a dataset and repairing them in order to improve the quality of the data. This includes qualitative data cleaning which uses mainly integrity constraints, rules, or patterns to detect errors and quantitative approaches which are mainly based on statistical methods. Many sub-problems in data cleaning fall under the ambit of matching. These include:

- **Entity Resolution** is the problem of identifying pairs of *tuples* that denote the same entity. For example, the tuple ⟨Bill Gates, MS⟩ and ⟨William Gates, Microsoft⟩ refer to the same person. This has a number of applications such as identifying duplicate tuples between two tables. Once again, we can see that this corresponds to discovering *matching* entities (or tuples), *i.e.,* those that refer to the same real-world objects.
- **Data Dependencies** specify how one attribute value depends on other attribute values. This is widely used as integrity constraints for capturing data violations. For example, the social security number determines a person's name while the country name typically determines its capital. We can see that this could be treated as an instance of *relevance matching*.
- **Outlier Detection** identifies anomalous data that *does not match* a group of values, either syntactically, semantically, or statistically. For example, the phone number 123 – 456 – 7890 is anomalous when the other values are of the form *nnnnnnnnn*.
- **Data Imputation.** is the problem of guessing the missing value that should *match* its surrounding evidence.

The main drawback of traditional DC solutions is that they typically use syntactic similarity based features. The syntactic approach is often ad-hoc, heavily reliant on domain experts and thus not extensible. While they are effective, they cannot leverage semantic similarity. A different approach is needed.

Intuitively, distributed representations, which can interpret one object in many different ways, may provide great help for various DC problems. However, there are a number of challenges before they could be used in DC. In domains such as NLP, simple co-occurrence is a sufficient approximation for distributional similarity. In contrast, domains such as DC require much more complex syntactic and semantic relationships between (multiple) attribute values, tuples, columns, and tables. Furthermore, the data itself could be noisy and the learned distributions should be resilient to errors.

3.2 Distributed Representation of Cells

A *cell*, which is an attribute value of a tuple, is the atomic data element in a relational database. Learning distributed representation for cells is already quite challenging and requires synthesis of representation learning of words and graphs.

Word Embeddings based Approaches. An initial approach inspired by word2vec [31] treats this as equivalent to the problem of learning word embeddings. Each tuple corresponds to a document where the value of each attribute corresponds to words. Hence, if two cell values occur together often in a similar context, then their distributed representation will be similar. For example, if a relation has attributes Country and Capital with many tuples containing *(USA, Washington DC)* for these two attributes, then their distributed representations would be similar.

Limitations of this Approach. The embeddings produced by this approach suffer from a number of issues. First, databases are typically well normalized to reduce redundancy, which also minimizes the frequency that two semantically related attribute values co-occur in the same tuples. Databases have many data dependencies (or integrity constraints), within tables (*e.g.,* functional dependencies [2], and conditional functional dependencies [17])

An Employee Table

Employee ID	Employee Name	Department ID	Department Name
0001	John Doe	1	Human Resources
0002	Jane Doe	2	Marketing
0003	John Smith	1	Human Resources
0004	John Doe	1	Finance

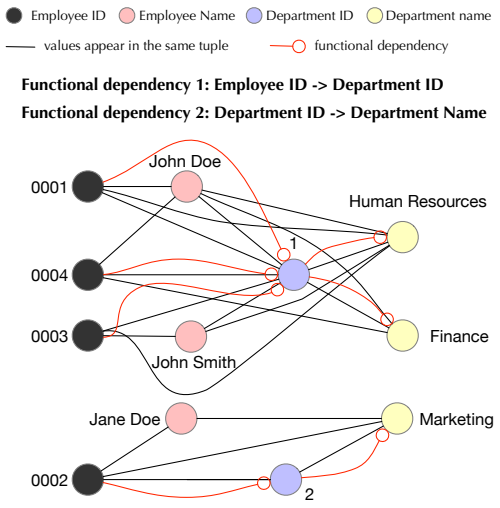

Figure 2: A Heterogeneous Graph of A Table

or across tables (*e.g.,* foreign keys, and matching dependencies). These data dependencies are important hints about semantically related cells that should be captured by learning distributed representations of cells.

Combining Word and Graph Embeddings. To capture the relationships (*e.g.,* integrity constraints) between cells, a more natural way is to treat each relation as a *heterogeneous network*. Each relation D is modeled as a graph $G(V, E)$, where each node $u \in V$ is a unique attribute value, and each edge $(u, v) \in E$ represents multiple relationships, such as (u, v) co-occur in one tuple, there is functional dependency from the attribute of u to the attribute of v, and so on. The edges could be either directed or undirected, have labels and weights. This enriched model might provide a more meaningful distributed representation that is cognizant of both content and constraints.

A sample table and our proposed graph representation of the table is shown in Figure 2. There are four distinct Employee ID values (nodes), three distinct Employee Name values, two distinct Department ID values, and three Department Name values. There are two types of edges: undirected edges indicating values appearing in the same tuple, *e.g.,* 0001 and John Doe, and directed edges for functional dependencies, *e.g.,* Employee ID 0001 implies DepartmentID 1.

Research Opportunities.

- *Algorithms.* How can we design an algorithm for learning cell embeddings that take values, integrity constraints, and other metadata (*e.g.,* a query workload) into consideration?
- *Global Distributed Representations.* We need to learn distributed representations for the cells over the entire data lake, not only on one relation. How can we "transfer" knowledge gained from one relation to another to improve the representations?
- *Handling Rare Values.* Word embeddings often provide inadequate representations for rare values. How can we ensure that primary keys and other rare values have a meaningful representation?

3.3 Hierarchical Distributed Representations

Many DC tasks are often performed at a higher level of granularity than cells. The next fundamental question to solve is to design an algorithm to *compose* the distributed representations of more abstract units from these atomic units. As an example, how can one design an algorithm for tuple embeddings assuming one already has a distributed representation for each of its attribute values? A common approach is to simply *average* the distributed representation of all its component values.

Research Opportunities.

- *Tuple, Column and Table Embeddings (Tuple2Vec, Column2Vec, Table2Vec):* Are there any other elegant approaches to compose representations for tuples than averaging? Can it be composed from representations for cells? Or should it be learned directly? How can one extend this idea to learn representations for columns that are often useful for tasks such as schema matching? Finally, how can one learn a representation for the entire relation that can benefit a number of tasks such as copy detection or data discovery (finding similar relations)?
- *Compositional or Direct Learning*: Different domains require different ways to create hierarchical representations. For the computer vision domain, hierarchical representations such as shapes are learned by composing simpler individual representations such as edges. On the other hand, for NLP, hierarchical representations are often learned directly [28]. What is an appropriate mechanism for DC?
- *Contextual Embeddings for DC.* There has been increasing interest in the NLP community in contextual embeddings [10, 38] that can provide different embeddings for a word (such as apple) based on the surrounding context. This is often helpful for tasks requiring word disambiguation. Often, DC tasks require a number of contextual information and hence any learned representation must take that into account. What is an appropriate formalization of context and algorithms for contextual embeddings in DC?
- *Multi Modal Embeddings for DC.* Enterprises often possess data across various modalities, such as structured data (*e.g.,* relations), unstructured data (*e.g.,* documents), graphical data (*e.g.,* enterprise networks), and even videos, audios, and images. An intriguing line of research is *cross modal representation learning* [25], wherein two entities that are similar in one or more modalities, *e.g.,* occurring in the same relation, document, image, and so on, will have similar distributed representations.

4 DEEP LEARNING ARCHITECTURES FOR DATA CURATION TASKS

Representation learning and domain specific architectures are the two pillars that led to the major successes achieved by DL in various domains. While representation learning ensures that the input to the DL model contains all relevant information, the domain specific architecture often processes the input in a meaningful way requiring less training data than generic architectures.

In this section, we motivate the need for designing DC specific architecture and provide a promising design space to explore.

4.1 Need for DC Specific DL Architectures

Recall from Section 2 that a fully connected architecture is the most generic one. It does not make any domain specific assumptions and hence can be used for arbitrary domains. However, this generality comes at a cost: a lot of training data. One can argue that a major part of DL's success in computer vision and NLP is due to the design of specialized DL architectures – CNN and RNN respectively. CNN leverages spatial hierarchies of patterns where complex/global patterns are often composed of simpler/local patterns (*e.g., curves* → *mouth* → *face*). Similarly, RNN processes an input sequence one step at a time while maintaining an internal state. These assumptions allows one to design effective neural architectures for processing images and text that also require less training data. This is especially important in data curation where there is a persistent scarcity of labeled data. There is a pressing need for new DL architectures that are tailored for DC and are cognizant of the characteristics of DC tasks. This would allow them to do the learning efficiently both in terms of training time and the amount of required training data.

Desiderata for DC-DL Architectures.

- *Handling Heterogeneity.* In both CNN and RNN, the input is homogeneous – images and text. However, a relation can contain a wide variety of data, such as categorical, ordinal, numerical, textual, and image.
- *Set/Bag Semantics.* While an image can be considered as a sequence of pixels and a document as a sequence of words, such an abstraction does not work for relational data. Furthermore, the relational algebra is based on set and bag semantics with the major query languages specified as set operators. DL architectures that operate on sets are an under explored area in DL.
- *Long Range Dependencies.* The DC architecture must be able to determine long range dependencies across attributes and sometimes across relations. Many DC tasks rely on the knowledge of such dependencies. They should also be able to leverage additional domain knowledge and integrity constraints.

4.2 Design Space for DC-DL Architectures

Broadly speaking, there are two areas in which DL architectures could be used in DC. First, novel DL architectures are needed for learning effective representations for downstream DC tasks. Second, we need DL architectures for common DC tasks such as matching, data repair, imputation, and so on. Both are challenging on their own right and require significant innovations.

Architectures for DC Representation Learning. In a number of fields such as computer vision, deep neural networks learn general features in early layers and transition to task specific features in the latter layers. Initial layers learn generic features such as edges (which can be used in many tasks) while latter layers learn high level concepts such as a cat or dog (which could be used for task for recognizing cats from dogs). Models trained on large datasets such as ImageNet could be used as feature extractors followed by specific task specific models. Similarly, in NLP there has been a number of pretrained language models such as word2vec [31], ELMo [38], and BERT [10] that could then be customized for various tasks such as classification, question answering, semantic matching, and coreference resolution. In other words, the pre-trained DL models act as a set of Lego bricks that could be put together to perform the required functionality. It is thus important that the DL architecture for DC follows this property by learning features that are generic and could readily generalize to many DC tasks.

While there has been extensive work in linguistics about the hierarchical representations of language, a corresponding investigation in data curation is lacking. Nevertheless, we believe that a generic DC representation learning architecture must be inspired based on the pretrained language models. Once such an architecture is identified, the learned representations could be used for multiple DC tasks.

DL Architectures for DC Tasks. While the generic approach is often powerful, it is important that we must also work on DL architectures for specific DC tasks. This approach is often much simpler and provides incremental gains while also increasing our understanding of DL-DC nexus. In fact, this has been the approach taken by the NLP community - they worked on DL architectures for specific tasks (such as text classification or question answering) even while they searched for more expressive generative language models.

A promising approach is to take specific DC tasks and break them into simpler components and evaluate if there are existing DL architectures that could be reused. Often, it is possible to "compose" individual DL models to create a more complex model for solving a specific DC task. Many of the DC tasks could be considered as a combination of tasks such as matching, error detection, data repair, imputation, ranking, discovery, and syntactic/semantic transformations. An intriguing research question is to identify DL models for each of these and investigate how to instantiate and compose them together.

4.3 Pre-Trained DL Models for DC

In domains such as computer vision and NLP, there is a common tradition of training a DL model on a large dataset and then reusing it (after some tuning) for tasks on other smaller datasets. One example include ImageNet [9] which contains almost 14M images over 20k categories. The spatial hierarchy learned from this dataset is often a proxy for modeling the visual world [5] and can be applied on a different dataset and even different categories [5]. Similarly, in NLP, word embeddings are an effective approximation for language modeling. The embeddings are often learned from large diverse corpora such as Wikipedia or PubMed and could be used for downstream NLP tasks. These pre-trained models can be used in two ways: (a) *feature extraction* where these are used to extract generic features that are fed to a separate classifier for the task at hand; (b) *fine-tuning* where one adjusts the abstract representations from the last few hidden layers of a pre-trained model and make it more relevant to a targeted task. A promising research avenue is the design of pre-trained DL models for DC that could be used by others with comparatively less resources.

5 EARLY SUCCESSES OF DL FOR DC

In this section, we describe the early successes achieved by the DC community on using DL for major DC tasks such as data discovery and entity matching.

5.1 Data Discovery

Large enterprises typically possess hundreds or thousands of databases and relations. Data required for analytic tasks is often scattered across multiple databases depending on who collected the data. This makes the process of finding relevant data for a particular analytic task very challenging. Usually, there is no domain expert who has complete knowledge about the entire data repository. This results in a non-optimal scenario where the data analyst patches together data from her prior knowledge or limited searches – thereby leaving out potentially useful and relevant data.

As an example of applying word embeddings for data discovery, [19] shows how to discover semantic links between different data units, which are materialized in the *enterprise knowledge graph* (EKG)[1]. These links assist in data discovery by linking tables to each other, to facilitate navigating the schemas, and by relating data to external data sources such as ontologies and dictionaries, to help explain the schema meaning. A key component is a semantic matcher based on word embeddings. The key idea is that a group of words is similar to another group of words if the average similarity in the embeddings between all pairs of words is high. This approach was able to surface links that were previously unknown to the analysts, *e.g., isoform*, a type of protein, with *Protein* and *Pcr* – polymerase chain reaction, a technique to amplify a segment of DNA – with *assay*. We can see that any approach using syntactic similarity measures such as edit distance would not have identified these connections.

5.2 Entity Matching

Entity matching is a key problem in data integration, which determines if two tuples refer to the same underlying real-world entity [15].

DeepER. A recent work [14], DeepER, applies DL techniques for ER, whose overall architecture is shown in Figure 3. DeepER outperforms existing ER solutions in terms of accuracy, efficiency, and ease-of-use. For accuracy, DeepER uses sophisticated compositional methods, namely uni- and bi-directional recurrent neural networks (RNNs) with long short term memory (LSTM) hidden units to convert each tuple to a distributed representation, which are used to capture similarities between tuples. DeepER uses a locality sensitive hashing (LSH) based approach for blocking; it takes all attributes of a tuple into consideration and produces much smaller blocks, compared with traditional methods that consider only few attributes. For ease-of-use, DeepER requires much less human labeled data, and does not need feature engineering, compared with traditional machine learning based approaches which require handcrafted features, and similarity functions along with their associated thresholds.

DeepMatcher [35] proposes a template based architecture for entity matching. Figure 4 shows an illustration. It identifies four major components: attribute embedding, attribute summarization, attribute similarity and matching. It proposes various choices for each of these components leading to a rich design space. The four most promising models differ primarily in how the attribute summarization is performed and are dubbed as SIF, RNN, Attention, and Hybrid. SIF model is the simplest and computes a weighted average aggregation over word embeddings to get the attribute embedding. RNN uses a bi-directional

[1] An EKG is a graph structure whose nodes are data elements such as tables, attributes, and reference data such as ontologies and mapping tables, and whose edges represent different relationships between nodes.

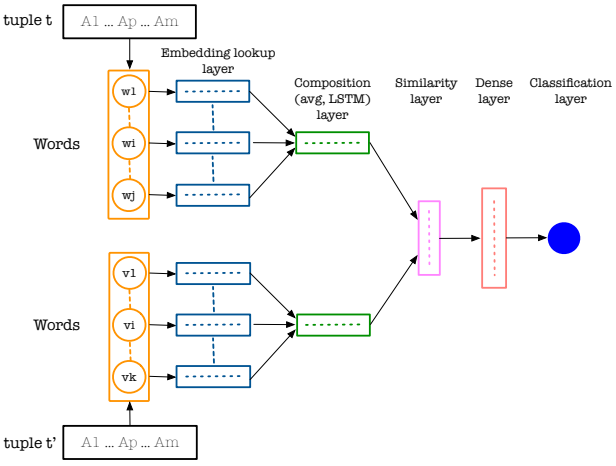

Figure 3: DeepER Framework (Figure from [14])

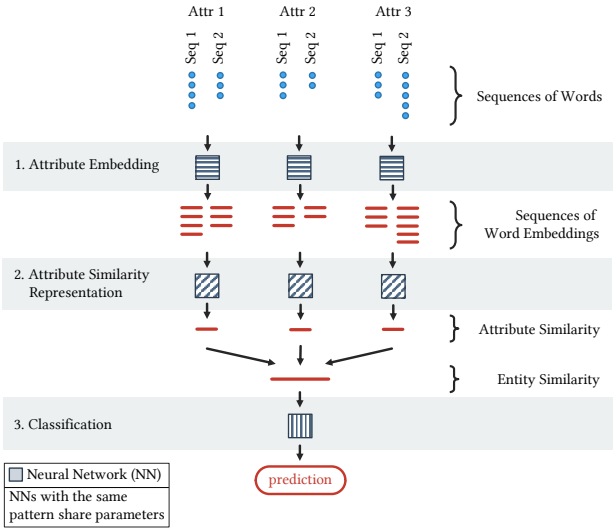

Figure 4: DeepMatcher Framework (Figure from [35])

GRU for summarizing an attribute for efficiency. Attention uses a decomposable attention model that can leverage information from aligned attributes of both tuples to summarize the attribute. Finally, the hybrid approach combines RNN and attention. DeepMatcher was evaluated on a diverse set of datasets such as structured, unstructured and noisy and provides useful rule of thumb on when to use DL for ER.

5.3 Representation Learning

A number of recent works such as DeepER [14], DeepMatcher [35] have applied a compositional approach using averaging or RNN/LSTMs for obtaining tuple embeddings from pre-trained word embeddings. For a number of benchmark datasets in entity resolution, this provides good results. There has been a number of recent efforts for learning distributed representations for data curation directly. Freddy [22] incorporated support for semantic similarity based queries by using pre-trained word embeddings inside Postgres. Termite [18] proposed an effective technique to learn a *common* (distributed) representation for both structured and unstructured data in an organization. In this approach, various entities such as rows, columns, and paragraphs

are all represented as a vector. This unified representation allows Termite to identify related concepts even if they are spread across structured and unstructured data. Finally, EmbDC [4] proposed an interesting approach to learn embeddings for cells that combine ideas from word and node embeddings. It constructs a tripartite graph and performs random walks over them. The walks correspond to sentences that are passed to an algorithm that computes word embeddings. This allows for different permutations of the same data being outputted thereby partially incorporating the set semantics. The authors show that this two step approach provides promising results for unsupervised DC tasks such as schema matching and entity resolution.

5.4 Understanding the Success and What is Missing?

Both DeepER and DeepMatcher provide a mechanism to obtain tuple embeddings from pre-trained word embeddings. The similarity between the embeddings of two tuples is then used to train a DL classifier. In both cases, the classifier was generic and hence the state-of-the-art performance of these approaches could be attributed to learning effective representations for the tuples. A follow-up work [50] showed that such learned representations could improve the performance of even non-DL classifiers. [35] performed extensive empirical evaluation and found that deep learning based methods were especially effective for entity resolution involving textual and dirty data. For example, the use of word embeddings allowed the DL based approach to identify that 'Bill' and 'William' are semantically similar while no string similarity metric could do that. Learning such representations could result in some unexpected applications such as performing entity resolution between relations that are in different languages [3].

Something analogous happens in the data discovery scenario as well. The word embedding based approach learned effective representations that could show that certain concepts were relevant (*isoform* and *Protein* or *Pcr* and *assay* as mentioned above) that could not have been identified by any syntactic similarity measures such as edit distance. This ability to learn representations that could identify similar pair of words even if they are syntactically dissimilar explains the success of DL for schema matching in [34, 43].

From the above discussion, it is clear that learning effective representations was the cornerstone of improved performance of DL based methods. Such representations learn relationships that are hard to pull off using non-DL methods. However, these early works are only scratching the surface as they do not yet leverage powerful techniques such as task specific architectures or transfer learning. As an example, DeepMatcher proposed a template based architecture that has two key layers – composition/aggregation and similarity computation. This could be considered as a rudimentary task specific architecture that resulted in reduced training data. Recent work such as [50] and [26] seek to use transfer learning to improve ER. However, the days of using pre-trained ER models for multiple datasets is still far off. Even within representation learning, works have not yet explored the use of contextual embeddings such as ELMO or BERT.

6 TAMING DL'S HUNGER FOR DATA

Deep learning often requires a large amount of training data. Unfortunately, in domains such as DC, it is often prohibitively expensive to get high quality training data. In order for DL to be widely used to solve DC problems, we must overcome the problem of obtaining training data. Fortunately, there are a number of exciting innovations from DL that can be adapted to solve this issue. We highlight some of these promising ideas and discuss some potential research questions.

6.1 Unsupervised Representation Learning

While the amount of *supervised* training data is limited, most enterprises have substantial amount of *unsupervised* data in various relations and data lakes. These unlabeled data could be used to learn some generic patterns and representations that are then used train a DL model with relatively less labeled data. This technique is widely used in NLP where word embeddings are learned on large unlabeled corpus data such as Wikipedia and provide good performance for many downstream NLP tasks.

Research Opportunities.

- How can we perform a *holistic representation learning* over the enterprise data? How can the learned representations be used as features for downstream DC tasks such as entity matching, schema matching, etc? While there are some promising start in recent work such as [4, 51], more needs to be done.

6.2 Data Augmentation

Data augmentation increases the size of labeled training data without increasing the load on domain experts. The key idea is to apply a series of *label preserving transformations* over the existing training data. For an image recognition task, one can apply a transformations such as translation (moving the location of the dog/cat within the image), rotation (changing the orientation), shearing, scaling, changing brightness/color, and so on. Each of these operations does not change the label of the image (cat or dog) – yet generate additional synthetic training data.

Research Opportunities.

- *Label Preserving Transformations for DC.* What does label preserving transformations mean for DC? Is it possible to design an algebra of such transformations?
- *Domain Knowledge Aware Augmentation.* To avoid creating erroneous data, we could integrate domain knowledge and integrity constraints in the data augmentation process. This would ensure that we do not create tuples that say New York City is the capital of USA.
- *Domain Specific Transformations.* There are some recent approaches such as Tanda [40] that seek to learn domain specific transformations. For example, if one knows that *Country* → *Capital*, we can just swap the (Country, Capital) values of two tuples to generate two new tuples. In this case, even if the new tuple is not necessarily real, its label will be the same as the original tuple. Is it possible to develop an algebra for such transformations?

6.3 Synthetic Data Generation for DC

A seminal event in computer vision was the construction of ImageNet dataset [9] with many million images over thousands of categories. This was a key factor in the development of powerful DL algorithms. We believe that the DC community is in need of such an ImageNet moment.

Research Opportunities.

- *Benchmark for DC.* It is paramount to create a similar benchmark to drive research in DC (both DL and non-DL). While there has been some work for data cleaning such as BART [1], it is often limited to specific scenarios. For example, BART can be used to benchmark data repair algorithms where the integrity constraints are specified as denial constraints.
- *Synthetic Datasets.* If it is not possible to create an open-source dataset that has realistic data quality issues, a useful fall back is to create synthetic datasets that exhibit representative and realistic data quality issues. The family of TPC benchmarks involves a series of synthetic datasets that is somehow realistic and widely used for benchmarking database systems. A recent work [48] proposed a variational autoencoder based model for generating synthetic data that has similar statistical properties as the original data. Is it possible to extend that approach to encode data quality issues as well?

6.4 Weak Supervision

A key bottleneck in creating training data is that there is often an implicit assumption that it must be accurate. However, it is often infeasible to produce sufficient hand-labeled and accurate training data for most DC tasks. This is especially challenging for DL models that require a huge amount of training data. However, if one can relax the need for the veracity of training data, its generation will become much easier for the expert. The domain expert can specify a high level mechanism to generate training data without endeavoring to make it perfect.

Research Opportunities.

- *Weakly Supervised DL Models.* There has been a series of research (such as Snorkel [39]) that seek to *weakly supervise* ML models and provide a convenient programming mechanism to specify "mostly correct" training data. What are the DC specific mechanisms and abstractions for weak supervision? Can we automatically create such a weak supervision through data profiling?

6.5 Transfer Learning

Another trick to handle limited training data is to "transfer" representations learned in one task to a different yet related task. For example, DL models for computer vision tasks are often trained on ImageNet [9], a dataset that is commonly used for image recognition purposes. However, these models could be used for tasks that are not necessarily image recognition. This approach of training on a large diverse dataset followed by tuning for a local dataset and tasks has been very successful.

Research Opportunities.

- *Transfer learning.* What is the equivalent of this approach for DC? Is it possible to train on a single large dataset such that it could be used for many downstream DC tasks? Alternatively, is it possible to train on a single dataset and for a single task (such as entity matching) such that the learned representations could be used for entity matching in similar domains? How can we train a DL model for one task and tune the model for new tasks by using the limited labeled data instead of starting from scratch?
- *Pre-trained Models.* Is it possible to provide pre-trained models that have been trained on large, generic datasets? These models could then be tuned by individual practitioners in different enterprises.

7 DEEP LEARNING: MYTHS AND CONCERNS

In the past few years, DL has achieved substantial successes in many areas. However, DC has a number of characteristics that are quite different from prior domains where DL succeeded. We anticipate that applying DL to challenging real-world DC tasks can be messy. We now describe some of the concerns that could be raised by a pragmatic DC practitioner or a skeptical researcher.

7.1 Deep Learning is Computing Heavy

A common concern is that training DL models requires exorbitant computing resources where model training could take days even on a large GPU cluster. In practice, training time often depends on the model complexity, such as the number of parameters to be learnt, and the size of training data. There are many tricks that can reduce the amount of training time. For example, a task-aware DL architecture often requires substantially less parameters to be learned. Alternatively, one can "transfer" knowledge from a pre-trained model from a related task or domain and the training time will now be proportional to the amount of fine-tuning required to customize the model to the new task. For example, DeepER [14] leveraged word embeddings from GloVe (whose training can be time consuming) and built a light-weight DL model that can be trained in a matter of minutes even on a CPU. Finally, while training could be expensive, this can often be done as a pre-processing task. Prediction using DL is often very fast and comparable to that of other ML models.

7.2 Data Curation Tasks are Too Messy or Too Unique

DC tasks often require substantial domain knowledge and a large dose of "common sense". Current DL models are very narrow in the sense that they primarily learn from the correlations present in the training data. However, it is quite likely that this might not be sufficient. Unsupervised representation learning over the entire enterprise data can only partially address this issue. Current DL models are often not amenable to encoding domain knowledge in general as well as those that are specific to DC such as data integrity constraints. As mentioned before, substantial amount of new research on DC-aware DL architectures is needed. However, it is likely that DL, even in its current form, can reduce the work of domain experts.

DC tasks often exhibit a *skewed label distribution.* For the task of entity resolution (ER), the number of non-duplicate tuple pairs are orders of magnitude larger than the number of duplicate tuple pairs. If one is not careful, DL models can provide inaccurate results. Similarly, other DC tasks often exhibit *unbalanced cost model* where the cost of misclassification is not symmetric. Prior DL work utilizes a number of techniques to address these issues such as (a) cost sensitive models where the asymmetric misclassification costs are encoded into the objective function, and (b) sophisticated sampling approach where we under or over sample certain classes of data. For example, DeepER [14] samples non-duplicate tuple pairs that are abundant at a higher level than duplicate tuple pairs.

7.3 Deep Learning Predictions are Inscrutable

Domain experts could be concerned that the predictions of DL models are often uninterpretable. Deep learning models are often very complex and the black-box predictions might not be explainable by even DL experts. However, explaining why a DL model made a particular data repair is very important for a domain expert. Recently, there has been intense interest in developing algorithms for explaining predictions of DL models or designing interpretable DL models in general. Please see [21] for an extensive survey. Designing algorithms that can explain the prediction of DL models for DC tasks is an intriguing research problem. While there are some promising approaches exist for specific tasks such as Entity resolution [11, 13, 49], more research remains.

7.4 Deep Learning can be Easily Fooled

There exist a series of recent works which show that DL models (especially for image recognition) can be easily fooled by perturbing the images in an adversarial manner. The sub-field of adversarial DL [37, 47] studies the problem of constructing synthetic examples by slightly modifying real examples from training data such that the trained DL model (or any ML model) makes an incorrect prediction with high confidence. While this is indeed a long term concern, most DC tasks are often collaborative and limited to an enterprise. Furthermore, there are a series of research efforts that propose DL models that are more resistant to adversarial training such as [30].

7.5 Building Deep Learning Models for Data Curation is "Just Engineering"

Many DC researchers might look at the process of building DL models for DC and simply dismiss it as a pure engineering effort. And they are indeed correct! Despite its stunning success, DL is still at its infancy and the theory of DL is still being developed. To a DC researcher used to purveying a well organized garden of conferences such as VLDB/SIGMOD/ICDE, the DL research landscape might look like the wild west.

In the early stages, researchers might *just apply* an existing DL model or algorithm for a DC task. Or they might slightly tweak a previous model to achieve better results. We argue that database conferences must provide a safe zone in which these DL explorations are conducted in a principled manner. One could take inspiration from the computer vision community. They created a large dataset (ImageNet [9]) that provided a benchmark by which different DL architectures and algorithms can be compared. They also created one or more workshops focused on applying DL for specific tasks in computer vision (such as image recognition and scene understanding). The database community has its own TPC series of synthetic datasets that have been influential in benchmarking database systems. Efforts similar to TPC are essential for the success of DL-driven DC.

7.6 Deep Learning is Data Hungry

Indeed, this is one of the major issues in adopting DL for DC. Most classical DL architectures often have many hidden layers with millions of parameters to learn, which requires a large amount of training data. Unfortunately, the amount of training data for DC is often small. The good news is, there exist a wide variety of techniques and tricks in the DL's arsenal that can help address this issue. We could leverage the techniques described in Section 6 for addressing these issues.

8 A CALL TO ARMS

In this paper, we make two key observations. Data Curation is a long standing problem and needs novel solutions in order to handle the emerging big data ecosystem. Deep Learning is gaining traction across many disciplines, both inside and outside computer science. The meeting of these two disciplines will unleash a series of research activities that will lead to usable solutions for many DC tasks. We identified research opportunities in learning distributed representations for database aware objects such as tuples or columns and designing DC-aware DL architectures. We described a number of promising approaches to tame DL's hunger for data. We discuss the early successes in using DL for important DC tasks such as data discovery and entity matching that required novel adaptations of DL techniques. We make a call to arms for the database community in general, and the DC community in particular, to seize this opportunity to significantly advance the area, while keeping in mind the risks and mitigation that were also highlighted in this paper.

REFERENCES

[1] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro. Messing up with BART: error generation for evaluating data-cleaning algorithms. *PVLDB*, 2015.

[2] L. Berti-Equille, H. Harmouch, F. Naumann, N. Novelli, and S. Thirumuruganathan. Discovery of genuine functional dependencies from relational data with missing values. *Proceedings of the VLDB Endowment*, 11(8):880–892, 2018.

[3] Ö. Ö. Çakal, M. Mahdavi, and Z. Abedjan. Clrl: Feature engineering for cross-language record linkage. In *EDBT*, pages 678–681, 2019.

[4] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan. Local embeddings for relational data integration. *arXiv preprint arXiv:1909.01120*, 2019.

[5] F. Chollet. *Deep learning with Python*. Manning Publications, 2018.

[6] P. Cui, X. Wang, J. Pei, and W. Zhu. A survey on network embedding. *CoRR*, abs/1711.08752, 2017.

[7] M. Dallachiesa, A. Ebaid, A. Eldawy, A. K. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. NADEEF: a commodity data cleaning system. In *SIGMOD*, 2013.

[8] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang. The data civilizer system. In *CIDR*, 2017.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[11] V. Di Cicco, D. Firmani, N. Koudas, P. Merialdo, and D. Srivastava. Interpreting deep learning models for entity resolution: An experience report using lime. In *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, aiDM '19, pages 8:1–8:4, New York, NY, USA, 2019. ACM.

[12] A. Doan, A. Y. Halevy, and Z. G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[13] A. Ebaid, S. Thirumuruganathan, W. G. Aref, A. Elmagarmid, and M. Ouzzani. Explainer: Entity resolution explanations. In *ICDE*, pages 2000–2003. IEEE, 2019.

[14] M. Ebraheem, S. Thirumuruganathan, S. R. Joty, M. Ouzzani, and N. Tang. Deeper - deep entity resolution. *VLDB*, 12, 2018.

[15] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[16] W. Fan and F. Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[17] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(2), 2008.

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

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

[20] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[21] R. Guidotti, A. Monreale, F. Turini, D. Pedreschi, and F. Giannotti. A survey of methods for explaining black box models. *arXiv:1802.01933*, 2018.

[22] M. Günther. Freddy: Fast word embeddings in database systems. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1817–1819. ACM, 2018.

[23] J. Heer, J. M. Hellerstein, and S. Kandel. Predictive interaction for data transformation. In *CIDR*, 2015.

[24] G. E. Hinton. Learning distributed representations of concepts. In *CogSci*, volume 1, page 12. Amherst, MA, 1986.

[25] X. Huang, Y. Peng, and M. Yuan. Cross-modal common representation learning by hybrid transfer network. In *IJCAI*, 2017.

[26] J. Kasai, K. Qian, S. Gurajada, Y. Li, and L. Popa. Low-resource deep entity resolution with transfer and active learning. *arXiv preprint arXiv:1906.08042*, 2019.

[27] P. Konda, S. Das, P. S. G. C., A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. F. Naughton, S. Prasad, G. Krishnan, R. Deep, and V. Raghavendra. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12):1197–1208, 2016.

[28] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. In *ICML*, 2014.

[29] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[30] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv:1706.06083*, 2017.

[31] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[32] R. J. Miller. Big data curation. In *COMAD*, page 4, 2014.

[33] R. J. Miller, F. Nargesian, E. Zhu, C. Christodoulakis, K. Q. Pu, and P. Andritsos. Making open data transparent: Data discovery on open data. *IEEE Data Eng. Bull.*, 41(2):59–70.

[34] M. J. Mior, I. Ororbia, and G. Alexander. Column2vec: Structural understanding via distributed representations of database schemas. *arXiv preprint arXiv:1903.08621*, 2019.

[35] 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.

[36] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller. Table union search on open data. *PVLDB*, 11(7):813–825, 2018.

[37] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In *EuroS&P*, 2016.

[38] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[39] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. *arXiv:1711.10160*, 2017.

[40] A. J. Ratner, H. R. Ehrenberg, Z. Hussain, J. Dunnmon, and C. Ré. Learning to compose domain-specific transformations for data augmentation. In *NIPS*, 2017.

[41] S. Ravanbakhsh, J. Schneider, and B. Poczos. Deep learning with sets and point clouds. *arXiv preprint arXiv:1611.04500*, 2016.

[42] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11), 2017.

[43] R. Shraga, A. Gal, and H. Roitman. Deep smane-deep similarity matrix adjustment and evaluation to improve schema matching. Technical report, Technical Report IE/IS-2019-02. Technion–Israel Institute of Technology âĂę, 2019.

[44] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The data tamer system. In *CIDR*, 2013.

[45] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The data tamer system. In *CIDR*, 2013.

[46] M. Stonebraker and I. F. Ilyas. Data integration: The current status and the way forward. *IEEE Data Eng. Bull.*, 41(2):3–9, 2018.

[47] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv:1312.6199*, 2013.

[48] S. Thirumuruganathan, S. Hasan, N. Koudas, and G. Das. Approximate query processing using deep generative models. *arXiv preprint arXiv:1903.10000*, 2019.

[49] S. Thirumuruganathan, M. Ouzzani, and N. Tang. Explaining entity resolution predictions: Where are we and what needs to be done? 2019.

[50] S. Thirumuruganathan, S. A. P. Parambath, M. Ouzzani, N. Tang, and S. Joty. Reuse and adaptation for entity resolution through transfer learning. *arXiv preprint arXiv:1809.11084*, 2018.

[51] R. Wu, S. Chaba, S. Sawlani, X. Chu, and S. Thirumuruganathan. Autoer: Automated entity resolution using generative modelling. *arXiv preprint arXiv:1908.06049*, 2019.

[52] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.