Deep Learning for Entity Matching: A Design Space Exploration

Sidharth Mudgal[1], **Han Li**[1], **Theodoros Rekatsinas**[1], **AnHai Doan**[1],
Youngchoon Park[2], **Ganesh Krishnan**[3], **Rohit Deep**[3], **Esteban Arcaute**[4], **Vijay Raghavendra**[3]
[1]University of Wisconsin-Madison, [2]Johnson Controls, [3]@WalmartLabs, [4]Facebook

ABSTRACT

Entity matching (EM) finds data instances that refer to the same real-world entity. In this paper we examine applying deep learning (DL) to EM, to understand DL's benefits and limitations. We review many DL solutions that have been developed for related matching tasks in text processing (e.g., entity linking, textual entailment, etc.). We categorize these solutions and define a space of DL solutions for EM, as embodied by four solutions with varying representational power: SIF, RNN, Attention, and Hybrid. Next, we investigate the types of EM problems for which DL can be helpful. We consider three such problem types, which match structured data instances, textual instances, and dirty instances, respectively. We empirically compare the above four DL solutions with Magellan, a state-of-the-art learning-based EM solution. The results show that DL does not outperform current solutions on structured EM, but it can significantly outperform them on textual and dirty EM. For practitioners, this suggests that they should seriously consider using DL for textual and dirty EM problems. Finally, we analyze DL's performance and discuss future research directions.

KEYWORDS

Deep learning; entity matching; entity resolution

ACM Reference Format:
Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3183713.3196926

1 INTRODUCTION

Entity matching (EM) finds data instances referring to the same real-world entity, such as (Eric Smith, Johns Hopkins) and (E. Smith, JHU). This problem is critical in data cleaning and integration. As a result, it has received significant attention [9]. Tremendous progress has been made. But no satisfactory solution has yet been found.

In the past few years, deep learning (DL) has become a major direction in machine learning [28, 46, 63, 83]. DL yields state-of-the-art results for tasks over data with some hidden structure, e.g., text, image, and speech. On such data, using labeled examples, DL can automatically construct important features, thereby obviating the need for manual feature engineering. This has transformed fields such as image and speech processing, medical diagnosis, autonomous driving, robotics, NLP, and many others [28, 46]. Recently, DL has also gained the attention of the database research community [17, 83].

A natural question then is whether deep learning can help entity matching. Specifically, has DL been applied to EM and other related matching tasks? If so, what are those tasks, and what kinds of solutions have been proposed? How do we categorize those solutions? How would those DL solutions compare to existing (non-DL) EM solutions? On what kinds of EM problems would they help? And on what kinds of problems would they not? What are the opportunities and challenges in applying DL to EM? As far as we know, no published work has studied these questions in depth.

In this paper we study the above questions, with the goal of understanding the benefits and limitations of DL when applied to EM problems. Clearly, DL and EM can be studied in many different settings. In this paper, as a first step, we consider the classic setting in which we can automatically train DL and EM solutions on *labeled training data*, then apply them to test data. This setting excludes unsupervised EM approaches such as clustering, and approaches that require substantial human effort such as crowdsourced EM or EM using hand-crafted rules.

Defining a Space of DL Solutions: We begin by defining a space of DL solutions for EM and related matching tasks. As far as we can tell, only one work has proposed a DL solution called DeepER for EM [18]. But numerous DL solutions have been proposed for related matching tasks in the field of natural language processing (NLP), such as entity linking, coreference resolution, textual entailment, etc. [46]. We provide a categorization of these solutions that factors out their commonalities. Building on this categorization, we describe a DL architecture template for EM, and discuss the trade-offs of the design choices in this template. We select four DL solutions as "representative points" in the design space (formed by the combination of the choices). These solutions include relatively simple models such as DeepER, the existing DL solution for EM [18], as well as DL solutions with significantly more representational power. We refer to these four DL solutions as SIF, RNN, Attention, and Hybrid (see Section 4).

Defining a Space of EM Problems: Next, we investigate for which types of EM problems DL can be helpful. The most popular type of EM problems has been matching *structured data instances*, e.g., matching tuples where the attribute values are short and atomic, such as name, title, age, city, etc. (see Figure 1.a) [9]. Thus, we examine how DL performs on EM setups over such structured data.

In recent years, however, we have also seen an increasing demand for matching *textual data instances*, such as matching descriptions

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3196926

of products that correspond to long spans of text, matching company homepages with Wikipedia pages that describe companies, matching organization descriptions in financial SEC filings, and matching short blurbs describing Twitter users, among others (see Figure 1.b). We suspect that traditional learning-based EM solutions (e.g., those that use random forest, SVM, etc.) may have difficulties matching textual instances, because there are few meaningful features that we can create (e.g., computing word-level Jaccard or TF/IDF scores). On the other hand, we believe that DL can perform well here, due to its ability to learn from raw text, and its current successes in NLP task [12, 79, 88].

Understanding the advantages that DL has to offer for EM over textual data instances raises an intriguing possibility. In our extensive work on EM with many companies, we have seen many cases where the instances to be matched are *structured but dirty*. Specifically, the value for a particular attribute (e.g., brand) is missing from the cell for that attribute, but appears in the cell for another attribute (e.g., name), see for example tuple t_1 in Figure 1.c. This commonly arises due to inaccurate extraction of the attributes (e.g., extracting "leather red" as a value for attribute color, even though "leather" is a value for attribute materials). Traditional EM solutions do not work well for such cases. We suspect, however, that DL can be a promising solution to such EM problems, because it can simply ignore the "attribute boundaries" and treat the whole instance as a piece of text, thus in a sense going back to the case of matching textual instances.

Empirical Evaluation: To evaluate the above hypotheses, we assemble a collection of datasets, which includes all publicly available datasets (with labeled data) for EM that we know of, as well as several large datasets from companies. We create 11 EM tasks for structured instances, 6 tasks for textual instances, and 6 tasks for dirty instances, with the number of labeled instances ranging from 450 to 250K. We compare the four DL solutions described earlier (SIF, RNN, Attention, and Hybrid) with Magellan, a state-of-the-art open-source learning-based EM solution [41].

Our results show that DL solutions are competitive with Magellan on structured instances (87.9% vs 88.8% average F_1), but require far longer training time (5.4h vs 1.5m on average). Thus, it is not clear to what extent DL can help structured EM (compared to just using today learning-based EM solutions). On the other hand, DL significantly outperforms Magellan on textual EM, improving accuracy by 3.0-22.0% F_1. Our results also show that DL significantly outperforms Magellan on dirty EM, improving accuracy by 6.2-32.6% F_1. Thus, DL proves highly promising for textual and dirty EM, as it provides new automatic solutions that significantly outperform current best automatic solutions.

In addition to examining the accuracy of the various DL solutions (compared to current learning-based EM solutions) as we vary the type of EM tasks, we also examine how the different design choices in the space of DL solutions lead to various trade-offs between the accuracy and efficiency of these solutions. We further perform a detailed experimental validation of all identified trade-offs.

Finally, we analyze why DL solutions work better than current EM solutions, and why they do not yet reach 100% accuracy. We discuss the challenges of applying DL to EM (e.g., the effect of domain-specific semantics and training data on the performance of DL solutions, the need for automating the exploration of the accuracy and scalability trade-off for DL solutions for EM), as well as future research directions.

Contributions: To summarize, in this paper we make the following contributions:

- We provide a categorization of DL solutions for numerous matching tasks, and define a design space for these solutions, as embodied by four DL solutions SIF, RNN, Attention, and Hybrid. To our knowledge, this is the first work that defines a design space of DL solutions of varying complexity for learning distributed representations that capture the similarity between data instances.
- We provide a categorization of EM problems into structured EM, textual EM, and dirty EM. Structured EM has been studied extensively, but to our knowledge textual EM and dirty EM, while pervasive, have received very little or no attention in the database research community.
- We provide an extensive empirical evaluation that shows that DL does not outperform current EM solutions on structured EM, but it can significantly outperform them on textual and dirty EM. For practitioners, this suggests that they should consider using DL for textual and dirty EM problems.
- We provide an analysis of DL's performance and a discussion of opportunities for future research.

This project is conducted as a part of the larger Magellan project at UW-Madison [41, 42], which builds Magellan, a novel kind of EM system. Magellan provides support for the entire EM pipeline, and is built as a set of interoperable packages in the Python data science ecosystem (rather than as a single monolithic stand-alone system, as is commonly done today).

Magellan has been successfully applied to a range of EM tasks in domain sciences and at companies, and used in many data science classes [42]. We have open-sourced the four DL solutions described here as the deepmatcher Python package, as a part of the open-source code for Magellan. The code and some data used here are available at sites.google.com/site/anhaidgroup/projects/magellan, and more details are available in a technical report [55].

2 PRELIMINARIES AND RELATED WORK

2.1 Entity Matching

Problem Setting: We define an *entity* to be a distinct real-world object (e.g., person, organization, etc.). We define an *entity mention* to be a reference to a real-world entity, e.g., a data record in a structured dataset or a span of text in a document.

Let D and D' be two collections of entity mentions. We assume that entries in D and D' follow the same representation (e.g., the same schema with attributes $A_1, \ldots, A_N$ in the case of structured data). The goal of entity matching (EM) is to find all pairs of entity mentions between D and D' that refer to the same real-world entity [9]. These pairs are called *matches*. Typically, EM is done in two phases: *blocking* and *matching*. The goal of blocking is to filter the cross product $D \times D'$ to a *candidate set* C that only includes pairs of entity mentions judged likely to be matches. Typical blocking mechanisms are assumed to have no false negatives. The candidate set C often still contains pairs that correspond to non-matching

	Name	City	Age
t_1	Dave Smith	New York	18
t_2	David Smith	New York	18

(a) structured

	Description
t_1	Kingston 133x high-speed 4GB compact flash card ts4gcf133, 21.5 MB per sec data transfer rate, dual-channel support, multi-platform compatibility.
t_2	Kingston ts4gcf133 4GB compactflash memory card (133x).

(b) textual

	Name	Brand	Price
t_1	Adobe Acrobat 8		299.99
t_2	Acrobat 8	Adobe	299.99

(c) dirty

Figure 1: Tuple pair examples for the three EM problem types considered in this paper.

entity mentions. After blocking, a *matcher* is used to identify the true matching entity mentions.

We focus on the matching step of EM. We assume as input two collections D and D' and a candidate set C containing entity mention pairs $(e_1 \in D, e_2 \in D')$. We further assume access to a set T of tuples $\{(e_1^i, e_2^i, l)\}_{i=1}^{|T|}$ where $\{(e_1^i, e_2^i)\}_{i=1}^{|T|} \subseteq C$, and l is a label taking values in {"match", "no-match"}.

Given the labeled data T our goal is to design a matcher M that can accurately distinguish between "match" and "no-match" pairs (e_1, e_2) in C. We focus on machine learning (ML) based entity matching, because they obtain state-of-the-art results in benchmark EM datasets [24, 41, 43] and obviate the need for manually designing accurate matching functions [5, 24, 41]. Specifically, we use T as labeled training data to learn a matcher M that classifies pairs of entity mentions in C as "match" or "no-match".

Types of EM Problems: Recall that we want to know for which types of EM problems DL can be helpful. Toward this goal we consider the following three types:

(1) Structured EM: Entity mentions in D and D' are structured records that follow the same schema with attributes $A_1, \ldots, A_N$. We classify a dataset as structured when its entries are relatively clean, i.e., attribute values are properly aligned and cells of each record contain information that is associated only with the attribute describing each cell (see Figure 1.a). Further, the data may contain text-based attributes but of restricted length (e.g., product title, address).

(2) Textual EM: All attributes for entity mentions in D and D' correspond to raw text entries (see Figure 1.b).

(3) Dirty EM: Entity mentions in D and D' are structured records with the same schema $A_1, \ldots, A_N$. However, attribute values may be "injected" under the wrong attribute (i.e., attribute values are not associated with their appropriate attribute in the schema), see Figure 1.c.

For the above three EM problem types, we will experimentally compare state-of-the-art learning-based entity EM solutions that use traditional classifiers (e.g., logistic regression, SVM, and decision trees [41]) with classifiers that use state-of-the-art DL models (described in Section 4).

Related EM Work: EM has received much attention [9, 19, 24, 56, 64]. Most EM approaches in the database literature match structured records [24], whereas most related works in NLP and similarity learning match entity mentions that are text spans. We review prior work on NLP tasks related to EM in Section 2.3.

Work on the matching step of EM typically uses rules, learning, or crowdsourcing. Rule-based solutions [20, 68] are interpretable but require the heavy involvement of a domain expert. To address this problem, some work has focused on learning matching functions [5, 41, 69]. Finally, a different line of work develops methods to leverage human experts [26, 73, 81] to guide EM and help refine the learned matching functions.

The blocking step of EM has also received significant attention [9, 59]. Our solutions in this paper assume as input the output of a blocking procedure, and thus can work with any blocking procedure that has been proposed.

DeepER, a recent pioneering work [18], also focuses on designing DL solutions to EM. That work proposes two DL models that build upon neural network (NN) architectures used extensively in the NLP literature. Those two models correspond to instances of the design space introduced in this paper and are similar to the two DL models described in Sections 4.1 and 4.2. Our experimental analysis (see Section 5) shows that more complex models tend to outperform these simpler models. In addition, DeepER [18] discusses blocking and how distributed representations can be used to design efficient blocking mechanisms. Blocking is out of the scope of our paper.

Both DeepER and our work here formulate EM as a pairwise binary classification task using logistic loss. But there are other ways to formulate the same problem. For example, triplet learning [34] first learns good embeddings for objects using triplet loss, and then learns an NN classifier. The work in [57] attacks the problem by learning good entity embeddings in vector space using contrastive loss. A distance threshold or an NN classifier is used for classification. Yet another potential approach (used in the Question-Answering (QA) domain) poses matching as a nearest neighbor search problem [86] and can be adapted for EM. Finally, Matching Networks [80], an approach to perform image classification using labels of related images, may also be adapted for EM.

2.2 Deep Learning

We now review the basic DL concepts that are necessary to describe the DL-based matchers described in Section 4.

Neural Networks (NNs): The most basic model in deep learning corresponds to a *fully connected layer*. This layer takes as input a vector $\mathbf{x}$, performs an affine transformation of the input $\mathbf{w}\mathbf{x} + b$, and applies a non-linear *activation* function, e.g., a sigmoid function (σ) to produce the final output $\sigma(\mathbf{w}\mathbf{x} + b)$. Multi-layer NNs are simply a generalization of this basic idea where NN layers are stacked in sequence. For details we refer the reader to recent surveys [28, 63].

Recurrent Neural Networks (RNNs): A model that we build upon is RNNs [76], which have delivered state-of-the-art results in many NLP tasks, including speech recognition [3, 30, 32], translation [50], and information extraction [53, 85]. RNNs are designed to process data that is sequential in nature, especially when the input sequences have variable length. Given a sequence of m vectors as input, an RNN processes one vector at a time, and produces a

sequence of m vectors as output. At time step t an RNN processes the t^{th} input $\mathbf{x}_t$ to produce the t^{th} output $\mathbf{y}_t$, considering all the previous inputs it has seen so far. This is done by using a *recurrent unit*, which is an NN that is shared between all time steps. The recurrent unit contains a *hidden state* vector which is updated at every time step. At time step t, the recurrent unit takes the t^{th} input to the RNN $\mathbf{x}_t$ and the hidden state output of the previous time step $\mathbf{h}_{t-1}$ to produce the hidden state output of the current time step $\mathbf{h}_t$. Hence, the hidden state vector of the t^{th} time step $\mathbf{h}_t$ contains information from inputs $\mathbf{x}_1, \cdots, \mathbf{x}_t$. The output of an RNN for each time step $\mathbf{y}_t$ is the hidden state output at each time step $\mathbf{h}_t$. In many cases the last output of the RNN $\mathbf{y}_m$ is used to encode the input sequence into a fixed-length vector representation. In the rest of this paper we refer to this representation as a *summarization*.

Attention: Traditional RNNs compress the information contained in the input sequence to a fixed-length vector representation. During this process they consider all parts of the input sequence to be equally important. *This representation technique can make it difficult for RNNs to learn meaningful representations (summarizations) from long (and possibly noisy) input sequences.* Recent work [8, 77] has introduced *attention mechanisms* to overcome this limitation. An attention model is a method that takes n arguments $y_1, ..., y_n$ and a context c. It returns a vector z which is supposed to be the "summary" of the y_i, focusing on information linked to the context c. Combining attention mechanisms with RNNs allows the latter to "attend" to different parts of the input sequence at each step of the output generation. Importantly, attention mechanisms let the model learn what to attend to based on the input sentence.

Word Embeddings: Word embeddings are the de-facto standard in language modeling and feature learning in NLP [51]. A word embedding maps words or phrases from a vocabulary to vectors of real numbers [4]. Methods to generate these mappings include neural networks [52], dimensionality reduction techniques such as PCA [45] on the word co-occurrence matrix and other probabilistic techniques [25]. Word embeddings, many times pre-trained ones [36, 61], are combined with RNNs or other neural networks to boost the performance of NLP tasks [70, 71].

2.3 DL Solutions for Matching Tasks in NLP

We now briefly review DL solutions for matching related tasks in NLP (e.g., entity linking, coreference resolution, etc.), then provide a categorization that factors out their commonalities.

Entity Linking: Entity linking aims at linking entity mentions in an input document (usually a small piece of text) to a canonical entity in a knowledge base [67]. For example, given the text span "Apple announced the new generation iPhone" from a document and access to DBpedia, one needs to link entity mention "Apple" to entity "Apple Inc." in DBpedia. The key difference between entity linking and EM is that in entity linking the target knowledge base contains additional information such as the relationships between entities. Most entity linking solutions use this information to collectively reason about multiple entities during linking [24]. DL approaches to entity linking are no exception. For example, recent DL work [21, 75] proposed the use of hierarchical word embeddings to obtain representations that capture entity cooccurrences, Ganea et al. [22] extended attention mechanisms to consider not only the input text span but also surrounding context windows, and Huang et al. [35] rely on the knowledge base structure and develop a deep, semantic entity similarity model using feed-forward NNs.

Coreference Resolution: Coreference resolution takes as input a document (or collection of documents) and aims to identify and group text spans that refer to the same real-world entity [37]. For example, both "N.Y." and "Big Apple" refer to "New York" in an article introducing the city. While related to EM, coreference resolution is significantly different as it operates on entity mentions that correspond to (typically short) text spans that commonly appear in the same document, and thus share similar context. As with most NLP tasks, recent work has proposed DL solutions to coreference resolution. For example, Clark et al. [10] used word embeddings to encode and rank pairs of entity mentions and use a deep neural network to identify clusters of entity mentions that correspond to the same entity, and Wiseman et al. [84] proposed using an RNN to obtain a global representation of entity clusters.

Textual Entailment & Semantic Text Similarity: Textual entailment [14] determines when the meaning of a text excerpt is contained in the meaning of a second piece of text, i.e., if the two meanings are semantically independent, contradictory or in an entailment relationship where one sentence (called the premise) can induce the meaning of the other one (called the hypothesis). For example, the sentence "a cat is chasing a mouse" entails another sentence "a cat is moving", contradicts with "a cat is sleeping", while is neutral with "Tom saw a cat with yellow eyes". A similar task is semantic text similarity, which decides if two given text snippets are semantically similar. Many DL solutions have been proposed for these problems. For example, recent work [7, 49] proposed using Bi-LSTMs—a state-of-the-art RNN—with attention to learn representation vectors for sentence pairs. A different line of work [60, 66] suggested that using only attention mechanisms with simple feed-forward NNs suffice to summarize pairs of sentences, thus avoiding learning complicated representations such as those obtained by RNNs. Neculoiu et al. [57] proposes using Bi-LSTMs with a siamese architecture trained using a contrastive loss function. This is so that matching sentences would be nearby in vector space. More recently, Nicosia et al. [58] builds upon this architecture and proposes training the network by also jointly minimizing a logistic loss apart from the contrastive loss to improve classification accuracy.

Question Answering: In question answering (QA) [31], the task is to answer natural language questions, using either existing text passages or a given knowledge base. Here DL solutions include [27, 78, 87]. Golub et al. [27] build upon RNNs to learn representations that summarize questions and entities. To deal with rare words, [87] adopt a character-level NN to generate the word embeddings.

Categorization of the DL Solutions: While DL models (i.e., solutions) for NLP seems highly specialized at first glance, they do in fact share several commonalities. All models that we have discussed so far take as input a pair of sequences, learn a vectorized representation of the input sequence pair, then perform a comparison between the two sequences. All of these models can be classified along three dimensions: (1) the *language representation* used to encode the input sequences (e.g., use a pre-trained word or character

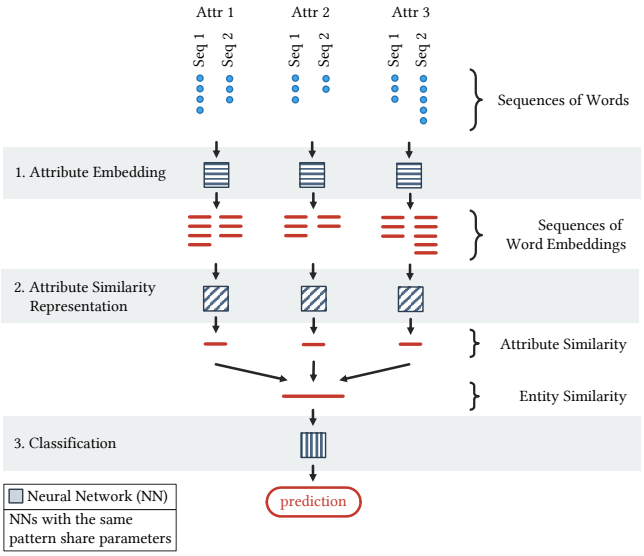

Figure 2: Our architecture template for DL solutions for EM.

embedding or learn one from scratch), (2) the kind of network used to *summarize* the input, i.e., learn a vector representation of the input sequence pair (e.g., use an RNN or an attention-only mechanism or a combination of the two), and (3) the method used to *compare* the two input sequences (e.g., a neural network).

3 A DESIGN SPACE OF DL SOLUTIONS

Building on the above categorization of the DL solutions for matching tasks in NLP, we now describe an architecture template for DL solutions for EM. This template consists of three main modules, and for each module we provide a set of choices. The combinations of these choices form a design space of possible DL solutions for EM. The next section selects four DL solutions for EM (SIF, RNN, Attention, and Hybrid) as "representative points" in this design space. Section 5 then evaluates these four DL solutions, as well as the trade-offs introduced by the different design choices.

3.1 Architecture Template & Design Space

Figure 2 shows our architecture template for DL solutions for EM. This template is for the matching phase of EM only (the focus of this paper). It uses the categorization of DL models for matching related tasks discussed in Section 2.3, and is built around the same categorization dimensions: (1) the language representation, (2) the summarization technique, and (3) the comparison method used to analyze an input pair of sequences. The template consists of three main modules each of which is associated with one of these dimensions.

Before discussing the modules, we discuss assumptions regarding the input. We assume that each input point corresponds to a pair of entity mentions (e_1, e_2), which follow the same schema with attributes $A_1, \ldots, A_N$. Textual data can be represented using a schema with a single attribute. We further assume that the value of attribute A_j for each entity mention e corresponds to a sequence of words $\mathbf{w}_{e,j}$. We allow the length of these sequences to be different across different entity mentions. Given this setup, each input point corresponds to a vector of N entries (one for each attribute $A_j \in \{A_1, \ldots, A_N\}$) where each entry j corresponds to a pair of word sequences $\mathbf{w}_{e_1,j}$ and $\mathbf{w}_{e_2,j}$.

The Attribute Embedding Module: For all attributes $A_j \in A_1 \cdots A_N$, this module takes sequences of words $\mathbf{w}_{e_1,j}$ and $\mathbf{w}_{e_2,j}$ and converts them to two sequences of word embedding vectors $\mathbf{u}_{e_1,j}$ and $\mathbf{u}_{e_2,j}$ whose elements correspond to d-dimensional embeddings of the corresponding words. More precisely, if for e_1, word sequence $\mathbf{w}_{e_1,j}$ contains m elements then we have $\mathbf{u}_{e_1,j} \in \mathbb{R}^{d \times m}$. The same holds for e_2. The overall output of the attribute embedding module of our template is a pair of embeddings $\mathbf{u}_{e_1,j}$ and $\mathbf{u}_{e_2,j}$ for the values of attribute A_j for entity mentions e_1 and e_2. We denote the final output of this module as $\{(\mathbf{u}_{e_1,j}, \mathbf{u}_{e_2,j})\}_{j=1}^{N}$.

The Attribute Similarity Representation Module: The goal of this module is to automatically learn a representation that captures the similarity of two entity mentions given as input. This module takes as input the attribute value embeddings $\{(\mathbf{u}_{e_1,j}, \mathbf{u}_{e_2,j})\}_{j=1}^{N}$ and encodes this input to a representation that captures the attribute value similarities of e_1 and e_2. For each attribute A_j and pair of attribute embeddings $(\mathbf{u}_{e_1,j}, \mathbf{u}_{e_2,j})$ the operations performed by this module are split into two major parts:

(1) Attribute summarization. This module takes as input the two sequences $(\mathbf{u}_{e_1,j}, \mathbf{u}_{e_2,j})$ and applies an operation H that summarizes the information in the input sequences. More precisely, let sequences $\mathbf{u}_{e_1,j}$ and $\mathbf{u}_{e_2,j}$ contain m and k elements respectively. An h dimensional summarization model H takes as input sequences $\mathbf{u}_{e_1,j} \in \mathbb{R}^{d \times m}$ and $u_{e_2,j} \in \mathbb{R}^{d \times k}$ and outputs two summary vectors $\mathbf{s}_{e_1,j} \in \mathbb{R}^h$ and $\mathbf{s}_{e_2,j} \in \mathbb{R}^h$. The role of attribute summarization is to aggregate information across all tokens in the attribute value sequence of an entity mention. This summarization process may consider the pair of sequences $(\mathbf{u}_{e_1,j}, \mathbf{u}_{e_2,j})$ jointly to perform more sophisticated operations such as *soft alignment* [2].

(2) Attribute comparison. This part takes as input the summary vectors $\mathbf{s}_{e_1,j} \in \mathbb{R}^h$ and $\mathbf{s}_{e_2,j} \in \mathbb{R}^h$ and applies a comparison function D over those summaries to obtain the final similarity representation of the attribute values for e_1 and e_2. We denote that similarity representation by $s_j \in \mathbb{R}^l$ with $s_j = D(\mathbf{s}_{e_1,j}, \mathbf{s}_{e_2,j})$.

The output of the similarity representation module is a collection of similarity representation vectors $\{s_1, \ldots, s_N\}$, one for each attribute $A_1, \ldots, A_N$. There are various design choices for the two parts of this module. We discuss those in detail in Section 3.3.

The Classifier Module: This module takes as input the similarity representations $\{s_1, \ldots, s_N\}$ and uses those as features for a classifier M that determines if the input entity mentions e_1 and e_2 refer to the same real-world entity.

A Design Space of DL Solutions for EM: Our architecture template provides a set of choices for each of the above three modules. Figure 3 describes these choices (under "Options" on the right side of the figure). Note that we provide only one choice for the classifier module, namely a multi-layer NN, because this is the most common choice today in DL models for classification. For other modules we provide multiple choices. In what follows we discuss the choices for attribute embedding, summarization, and comparison. The numbering of the choices that we will discuss correspond to the numbering used in Figure 3.

Architecture module		Options	
Attribute embedding		*Granularity:* (1) Word-based (2) Character-based	*Training:* (3) Pre-trained (4) Learned
Attribute similarity representation	(1) Attribute summarization	(1) Heuristic-based (2) RNN-based (3) Attention-based (4) Hybrid	
	(2) Attribute comparison	(1) Fixed distance (cosine, Euclidean) (2) Learnable distance (concatenation, element-wise absolute difference, element-wise multiplication)	
Classifier		NN (multi-layer perceptron)	

Figure 3: The design space of DL solutions for EM.

3.2 Attribute Embedding Choices

Possible embedding choices for this module can be characterized along two axes: (1) the *granularity of the embedding* and (2) whether a *pre-trained embedding is used or a domain specific embedding is learned*. We now discuss these two axes.

(1) Word-level vs. (2) Character-level Embeddings: Given a sequence of words, a word-level embedding encodes each word in the sequence as a fixed d-dimensional vector. Procedurally, word level embeddings use a lookup table to convert a word into an embedding [52, 61]. To learn this lookup table word embeddings are trained either on large external corpora, such as Wikipedia, or on the corpus of the task in hand. An important design choice for word-level embeddings is handling out-of-vocabulary (OOV) tokens at test time [16]. A common approach is to replace infrequent words with a special token `UNK`, and use this to model OOV words.

Another option is that of character-level embeddings. This type of embedding takes as input the characters present in a word and use a neural network to produce a d-dimensional vector representation for the word. Unlike world-level embeddings where the end result of training is a lookup table, the end result here is a *trained model* that can produce word embeddings for any word containing characters in its known character vocabulary [6, 39]. The core idea behind these models is that words are made of *morphemes*, or meaningful sequences of characters, of varying lengths. For example, the word "kindness" is made of two morphemes, "kind" and "ness".

Character-level embeddings can offer significant performance improvements in domains with infrequent words (see Section 5.4) as they take into account the fact that many words can be morphologically related (e.g., "available", "availability" and "unavailable"). Character-level embeddings are more robust to out-of-vocabulary (OOV) words—OOV words may occur due to misspellings—as they leverage possible substrings of the word to approximate its embedding. This leads to better performance in scenarios such as entity matching where long-tail vocabularies are common and typographical errors are widespread (see Section 5.4).

(3) Pre-trained vs. (4) Learned Embeddings: A different choice in our template is to decide between using *pre-trained* word embeddings, such as word-level embeddings (e.g., word2vec [52] and GloVe [61]) or character-level embeddings (e.g., fastText [6]), or train embeddings from scratch. Pre-trained embeddings offer two distinctive advantages: (1) they lead to significantly smaller end-to-end training times, and (2) they have been trained over large corpora, such as Wikipedia, GoogleNews, and Gigaword, and thus, are more robust to linguistic variations. Pre-trained embeddings may not be suitable for domains where the vocabulary contains tokens with highly specialized semantics (e.g., product barcodes for retail applications). In this case, training a domain-specific embedding can offer improved performance (see Section 5.4).

3.3 Attribute Summarization Choices

Recall that the role of attribute summarization is to aggregate information across all tokens in the attribute value sequence of an entity mention. Given the attribute embeddings $\mathbf{u}_{e_1,j} \in \mathbb{R}^{d \times m}$ and $u_{e_2,j} \in \mathbb{R}^{d \times k}$ for attribute A_j, a summarization process H outputs two summary vectors $\mathbf{s}_{e_1,j} \in \mathbb{R}^h$ and $\mathbf{s}_{e_2,j} \in \mathbb{R}^h$. We identify four major options for attribute summarization.

(1) Aggregate Function: The summarization process H corresponds to a simple aggregate function over each embedding sequence $\mathbf{u}_{e_1,j}$ and $\mathbf{u}_{e_2,j}$, e.g., average or weighted average. More precisely, function $H : \mathbb{R}^{d \times \cdot} \rightarrow \mathbb{R}^d$ is applied to each input independently and produces a d-dimensional summarization. Here, the output dimension h is equal to d. The biggest advantage of this type of summarization is *training efficiency* since there is usually no learning involved. However, models that rely on this kind of summarization cannot learn complex interactions between words in the input sequence. The performance of this summarization method depends strongly on the quality of the embedding vectors [1].

(2) Sequence-aware Summarization: Here the summarization process H aims to learn complex interactions across the tokens in the input sequences $\mathbf{u}_{e_1,j}$ and $\mathbf{u}_{e_2,j}$. Specifically, function $H : \mathbb{R}^{d \times \cdot} \rightarrow \mathbb{R}^h$ is applied to each input and produces a h-dimensional summarization. To this end, process H can be built around an RNN (see Section 2) so that it takes into account the order and the semantics of the tokens in the input sequence. There are many variations of RNN models [46], including long short-term memory (LSTM) networks [33], gated recurrent unit (GRU) [8] networks, and bi-directional networks [29, 49]. Given an RNN we implement process H as follows. We pass an input sequence $\mathbf{u}_{e,j}$ through the RNN to obtain a sequence $\mathbf{h}_{e,j}$ of hidden states. These hidden states are then aggregated into a single h-dimensional vector $\mathbf{s}_{e,j}$. Typical operations for this aggregation correspond to taking the last hidden state of the RNN to be $\mathbf{s}_{e,j}$ or taking an average across all hidden states $\mathbf{s}_{e,j}$ [76]. The basic advantage of this summarization method allows us to reason about the context encoded in the entire input sequence. The limitations of this method are that (1) it does not learn meaningful representations in the presence of very long sequences (see Section 2), and (2) it does not analyze the inputs pairs $\mathbf{u}_{e_1,j}$ and $\mathbf{u}_{e_2,j}$ jointly to identify common contexts across sequences. The latter can lead to significant performance loss when the input sequences vary significantly in length [7].

(3) Sequence Alignment: Here, process H takes as input both sequences $\mathbf{u}_{e_1,j}$ and $\mathbf{u}_{e_2,j}$ and uses one as *context* when summarizing the other. To this end, process H can be built around attention mechanisms (see Section 2) that first learn to compute a soft alignment between two given sequences of words and then perform a word by word comparison [23]. Attention mechanisms are also very expressive [79] and have a significant drawback: they only leverage the context given to them as input and ignore any context present in the raw input sequence. For example, given a sequence, attention-based summarization cannot take the position of input tokens into account. As such, attention methods can perform poorly

in scenarios where the most informative token for matching two entity mentions is the first one. This problem can be addressed by combining them with sequence-based summarization methods.

(4) Hybrid: These attribute summarization methods are a combination of the sequence-aware and sequence alignment methods described above (see Section 4). Using these methods leads to very expressive models that are expensive to train. Section 5 empirically shows that DL models for EM that use hybrid attribute summarization methods can be up to 3× slower to train than other DL models. However, hybrid methods obtain more accurate results—up to 4.5% F_1—than other methods.

3.4 Attribute Comparison Choices

Recall that attribute comparison identifies the distance between the summary vectors $\mathbf{s}_{e_1,j} \in \mathbb{R}^h$ and $\mathbf{s}_{e_2,j} \in \mathbb{R}^h$ for attribute A_j. We use D to denote this comparison operation. We assume that the output of this operation is a fixed-dimension vector $s_j \in \mathbb{R}^l$. We identify two main options for the comparison operation D: (1) *fixed* and (2) *learnable* distance functions.

(1) Fixed Distance Functions: The first option is to use a predefined distance metric such as the cosine or Euclidean distance. Here, the output is a scalar capturing how similar the values of the two input entity mentions are for the attribute under consideration. Using fixed distance functions leads to lower training times but enforces strong priors over the similarity of attribute values.

(2) Learnable Distance Functions: To allow for more expressivity we can rely on the classification module of our template to learn a similarity function. In this case, the output vector $s_j \in \mathbb{R}^l$ of function D forms the input (features) to the matching classifier. Different operations such as concatenation, element-wise absolute difference, element-wise multiplication or hybrids of these are viable options for D. We experimentally evaluate different operations in Section 5. We find that using element-wise comparison to obtain the input for the matching classifier is beneficial if an aggregate function or sequence-aware summarization method was used for attribute summarization. This is because these two methods do not perform any cross sequence comparison during summarization.

4 REPRESENTATIVE DL SOLUTIONS FOR EM

The previous section describes a space of DL solutions for EM. We now select four DL solutions as "representative points" in this space. These solutions correspond to DL models of varying representational power and complexity—the more complex a model, the more parameters it has, thus learning requires more resources.

All four solutions use fastText [6]—a pre-trained character-level embedding—to implement the attribute embedding module of our architecture template. (Section 5.4 provides a detailed evaluation of the other design choices for this module.) Further, all four solutions use a two layer fully-connected ReLU HighwayNet [72] followed by a softmax layer to implement the classifier module. HighwayNets were used since they sped up convergence and produced better empirical results than traditional fully connected networks across EM tasks, especially in the case of small datasets. The four solutions use different choices for the attribute summarization process, however. They are named SIF, RNN, Attention, and Hybrid, respectively, after the choice for the attribute summarization part of the similarity representation module of our architecture.

4.1 SIF: An Aggregate Function Model

We first consider a model (i.e., a DL solution, we use "model" and "solution" interchangeably) that uses an aggregate function, specifically a *weighted average* for attribute summarization and an *element-wise absolute difference* comparison operation to form the input to the classifier module. Specifically, the weights used to compute the average over the word embeddings for an input sequence are as follows: given a word w the corresponding embedding is weighted by a weight $f(w) = a/(a + p(w))$ where a is a hyperparameter and $p(w)$ the normalized unigram frequency of w in the input corpus.

This model was chosen since it is a simple but effective baseline deep learning model. Its performance relies mostly on the expressive power of the attribute embedding and the classifier used. The weighting scheme used during averaging follows the Smooth Inverse Frequency (SIF) sentence embedding model introduced by Arora et al. [1]. This model was shown to perform comparably to complex—and much harder to train—models for text similarity, sentence entailment and other NLP tasks. This model is similar to the Tuple2vec-Averaging model by Ebraheem et al. [18], but is more expressive since it takes word frequencies into account.

4.2 RNN: A Sequence-aware Model

This second model uses a *bidirectional RNN* (i.e., a sequence-aware method) for attribute summarization and an *element-wise absolute difference* comparison operation to form the input to the classifier module. This is a medium-complexity model that takes the order of words in the input sequence into account. This model was selected since it is one of the most commonly used DL approaches for computing distributed representations of text snippets. The RNN model we use corresponds to a bidirectional GRU-based RNN model introduced by Cho et al. [8] for machine translation. Bidirectional RNNs are the de-facto deep learning model for NLP tasks [51].

We now provide a high-level description of the model. The model consists of two RNNs: the *forward* RNN that processes the input word embedding sequence $\mathbf{u}$ in its regular order (from element entry $u[1]$ to entry $u[t]$) and produces hidden states $\mathbf{f}_{1:t}$ and the *backwards* network that processes the input sequence in reverse order to produce hidden states $\mathbf{b}_{t:1}$. The final attribute summarization representation corresponds to the concatenation of the last two outputs of the bidirectional RNN, i.e., to the concatenation of f_t and b_1. In our experiments, we did not use multi-layered RNNs since we did not notice compelling performance gains with them (see Appendix B.2). This method is similar to the Tuple2Vec-Compositional DL method introduced by Ebraheem et al. [18].

4.3 Attention: A Sequence Alignment Model

This model uses *decomposable attention* to implement attribute summarization and *vector concatenation* to perform attribute comparison. This is a medium-complexity model that analyzes both input sequences jointly while learning a similarity representation. The attention model we use is a variation of a model introduced by Parikh et al. [60] for text entailment. This model was selected since it has been shown to achieve results close to the state of the art on

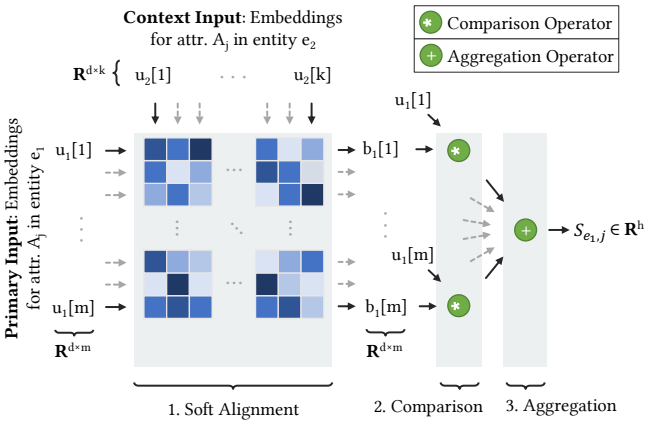

Figure 4: Decomposable attention-based attribute summarization module.

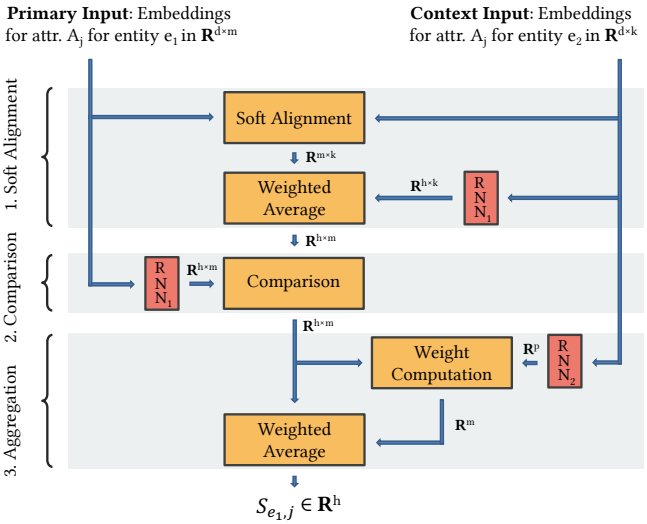

Figure 5: The Hybrid attribute summarization module.

NLP tasks related to EM [60]. Intuitively it performs soft alignment and pairwise token comparison across the two input sequences.

Figure 4 summarizes the working of Attention. Let $\mathbf{u}_1$ and $\mathbf{u}_2$ be two embedding sequences whose summarized representations we want to compute. To compute the summarized representation for $\mathbf{u}_1$ we give $\mathbf{u}_1$ as *primary input* and $\mathbf{u}_2$ as *context* to the attention model. We proceed in three steps:

(1) Soft Alignment: For each element $u_1[k]$ in the primary input $\mathbf{u}_1$ we compute a *soft-aligned encoding* of $u_1[k]$—denoted by $b_1[k]$—using all elements in the context sequence $\mathbf{u}_2$. To do so we first compute a *soft alignment matrix* $\mathcal{W}$ across all pairs of tokens for $\mathbf{u}_1$ and $\mathbf{u}_2$. Each row in $\mathcal{W}$ corresponds to an entry in $\mathbf{u}_1$ and each column to an entry in $\mathbf{u}_2$. Each entry of this matrix is a weight for a pair of elements $(u_1[k], u_2[m])$. This weight corresponds to a log-linear transformation of the dot product over the hidden representations of $u_1[k]$ and $u_2[m]$ obtained by a two layer HighwayNet (see Appendix A). We compute the encoding $b_1[k]$ for each $u_1[k] \in \mathbf{u}_1$ by taking a weighted average over all elements $u_2[m] \in \mathbf{u}_2$ with the weights being the entries of the k-th row of $\mathcal{W}$.

(2) Comparison: We compare each embedding $u_1[k] \in \mathbf{u}_1$ with its soft-aligned encoding $b_1[k]$ using a two layer HighwayNet with ReLU non-linearities. We denote as $x_1[k]$ the comparison representation for each $u_1[k] \in \mathbf{u}_1$.

(3) Aggregation: Here, we sum the comparison representation of all elements in $\mathbf{u}_1$ and normalize the output by dividing with $\sqrt{|\mathbf{u}_1|}$. This extension over the original model ensures that the variance of the final attribute summarization does not change as a function of the number of words in the attribute. *This ensures that the gradient magnitude of the parameters in the comparison module do not depend on the length of the attribute and thus promotes robustness.*

These steps are repeated for $\mathbf{u}_2$ as primary input and $\mathbf{u}_1$ as context.

4.4 Hybrid: Sequence-aware with Attention

This model uses *a bidirectional RNN with decomposable attention* to implement attribute summarization and a *vector concatenation augmented with element-wise absolute difference* during attribute comparison to form the input to the classifier module. This is the model with the highest representational power we consider in this paper. To our knowledge we are the first to consider such a model for entity matching. Our model is inspired by other hybrid models proposed in the NLP literature [82, 88]. However, those models either build upon convolutional neural networks [82] or use different attention mechanisms [88].

We now describe the internals of this model. Again, let $\mathbf{u}_1$ and $\mathbf{u}_2$ be two embedding sequences whose summarized representations we want to compute. Our hybrid model follows steps that are similar to those of the Attention model of Section 4.3. But in contrast to Attention, it also utilizes sequence-aware encodings of $\mathbf{u}_1$ and $\mathbf{u}_2$ obtained by a Bi-RNN. Figure 5 provides an overview. We have:

(1) Soft Alignment: First, Hybrid constructs a soft alignment matrix $\mathcal{W}$ between the primary input sequence $\mathbf{u}_1$ and the context sequence $\mathbf{u}_2$. The construction is the same as that described in Section 4.3. Then, Hybrid obtains a soft-aligned encoding $b_1[k]$ for each element $u_1[k] \in \mathbf{u}_1$. In contrast to Attention $b_1[k]$ is constructed by taking a weighted average over an encoding of $\mathbf{u}_2$. The weights are obtained by the soft-alignment matrix. The encoding of $\mathbf{u}_2$ is obtained by passing $\mathbf{u}_2$ through a Bi-RNN and concatenating all the hidden states of the Bi-RNN. Let RNN_1 denote this RNN. This process generates a vector $\mathbf{b}_1$ of soft-aligned encodings.

(2) Comparison: To obtain the comparison between $\mathbf{b}_1$ and the primary input $\mathbf{u}_1$ we: (i) obtain an encoding of $\mathbf{u}_1$—denoted by $\mathbf{u}'_1$ by passing it via the same RNN_1, and concatenating all of its hidden states; (ii) perform an element-wise comparison between $\mathbf{u}'_1$ and $\mathbf{b}_1$. Similar to Attention we use a two layer HighwayNet with ReLU to perform this element-wise comparison. Let $x_1[k]$ be the comparison representation for each $u_1[k] \in \mathbf{u}_1$.

(3) Aggregation: Finally, Hybrid aggregates the elements of the comparison vector $\mathbf{x}_1$ produced by the previous step using a *weighted average scheme*. The weight for each element $x_1[k]$ is obtained as follows: (i) we first obtain an encoding of $\mathbf{u}_2$—denoted g_2—by using a Bi-RNN (RNN_2) and taking its last hidden state to be the encoding of $\mathbf{u}_2$; (ii) we compute a weight for each element $x_1[k] \in \mathbf{x}_1$ by concatenating $x_1[k]$ with g_2 and passing it via a two layer ReLU HighwayNet followed by a soft-max layer. Intuitively,

Table 1: Comparison of Magellan (a current ML-based solution) vs. the best-performing DL solution.

Problem Type	Average F_1			Average Train Time	
	DL	Magellan	ΔF_1	DL	Magellan
Structured	87.9	88.8	-0.9	5.4h	1.5m
Textual	88.0	83.4	4.6	4.4h	6s
Textual w/o info. attr.	88.3	78.7	9.6	4.0h	5.5s
Dirty	87.9	68.5	19.4	0.7h	1.5m

this corresponds to a simple attention mechanism that identifies the importance of each element $x_1[k]$ given $\mathbf{u}_2$ as context; (iii) we take a weighted average over all elements of $\mathbf{x}_1$ using these weights.

The same steps are repeated for $\mathbf{u}_2$ as primary input and $\mathbf{u}_1$ as context. Steps 1-2 in this model are different from typical hybrid architectures such as [82, 88]. The modifications make the alignment computation between the input word embedding sequences to not rely on the RNN encoding of the input. This enables the model to converge faster since the alignment network can receive useful gradients right from the start of training when the RNN weights are random.

5 EMPIRICAL EVALUATION

Goals and Takeaways: Our first goal is to understand where DL outperforms current EM solutions. Toward this goal, we experimentally compare the four DL models from Section 4 (i.e., SIF, RNN, Attention, and Hybrid) with Magellan, a state-of-the-art learning-based EM solution [41]. We used 23 EM tasks that cover three types of EM problems: structured, textual, and dirty.

The main takeaways are as follows. (1) On structured EM tasks, DL solutions are competitive with Magellan but takes far more training time. Thus, it is not yet clear to what extent DL can help structured EM (see Table 1). (2) On textual EM tasks (i.e., instances having a few attributes all of which are textual blobs), DL outperforms Magellan. The gain may not be large if there are "informative" attributes (e.g., titles full of discriminative information), otherwise it can be significant. (3) On dirty EM tasks, DL significantly outperforms Magellan. Thus, we find that in the absence of labor-intensive data extraction/cleaning DL is highly promising, outperforming current automatic EM solutions for textual and dirty data.

Our second goal is to understand the impact of performance factors, such as model complexity (e.g., do we need complex DL models?) and amount of labeled data. The main takeaways are as follows. (1) When a limited amount of training data is available, models that use soft alignment (see Section 3.3) during attribute summarization should be preferred as they yield up to 23% higher F_1 over simpler DL models (see Section 5.4.2). (2) When a lot of training data is available, the accuracy difference between complex and simpler DL models is smaller. Thus, one can use simpler DL models that are faster to train (see Section 5.4).

Datasets: We use datasets from a diverse array of domains and different sizes (see Table 2). Dataset details are deferred to Appendix B. For structured EM, we use 11 datasets. The first seven datasets are publicly available and have been used for EM (e.g., [15, 43]). The last four datasets (Clothing_1, etc.) describe products in various categories and come from a major retailer. Column "Size" lists the number of labeled examples for each dataset. Each example has two tuples to be matched. The tuples are *structured*, i.e., attribute values are atomic, i.e., short and pure, and not a composition of multiple values that should appear separately.

Table 2: Datasets for our experiments.

Type	Dataset	Domain	Size	# Pos.	# Attr.
Structured	BeerAdvo-RateBeer	beer	450	68	4
	iTunes-Amazon_1	music	539	132	8
	Fodors-Zagats	restaurant	946	110	6
	DBLP-ACM_1	citation	12,363	2,220	4
	DBLP-Scholar_1	citation	28,707	5,347	4
	Amazon-Google	software	11,460	1,167	3
	Walmart-Amazon_1	electronics	10,242	962	5
	Clothing_1	clothing	247,627	105,608	28
	Electronics_1	electronics	249,904	98,401	28
	Home_1	home	249,513	111,714	28
	Tools_1	tools	249,317	96,836	28
Textual	Abt-Buy	product	9,575	1,028	3
	Company	company	112,632	28,200	1
	Clothing_2	clothing	247,627	105,608	3
	Electronics_2	electronics	249,904	98,401	3
	Home_2	home	249,513	111,714	3
	Tools_2	tools	249,317	96,836	3
Dirty	iTunes-Amazon_2	music	539	132	8
	DBLP-ACM_2	citation	12,363	2,220	4
	DBLP-Scholar_2	citation	28,707	5,347	4
	Walmart-Amazon_2	electronics	10,242	962	5
	Home_3	home	249,513	111,714	28
	Tools_3	tools	249,317	96,836	28

For textual EM, we use six datasets. Abt-Buy describes products [43]. Company is a dataset that we created. It tries to match company homepages and Wikipedia pages describing companies. The last four datasets (Clothing_2, etc.) describe products in different categories and come from a major retailer. In these datasets, each tuple has 1-3 attributes, all of which are *long textual blobs* (e.g., long title, short description, long description).

For dirty EM, we use six datasets. As discussed earlier, we focus on dirty data where attribute values are "injected" into other attributes, e.g., the value of brand is embedded in title while leaving the correct value cell empty. All dirty datasets are derived from the corresponding structured datasets described above. To generate them, for each attribute we randomly move its value to attribute title in the same tuple with 50% probability. This simulates a common dirty-data problem in real-world scenarios (e.g., information extraction) while keeping the modifications simple.

Methods: We evaluate the four DL models described in Section 4: SIF (aggregation-based), RNN (RNN-based), Attention (attention-based), and Hybrid. They are implemented using Torch [13] (a DL framework with extensive support for accelerating training using GPUs), and trained and evaluated on AWS p2.xlarge instances with Intel Xeon E5-2686 CPU, 61 GB memory, and Nvidia K80 GPU. We compare the DL models with Magellan, a state-of-the-art machine-learning based EM solution [41].

To measure accuracy, we use precision (P), the fraction of match predictions that are correct, recall (R), the fraction of correct matches being predicted as matches, and F_1, defined as $2PR/(P+R)$.

To apply the DL models to a dataset, we split all pairs in the dataset (recall that each example is a pair of tuples) into three parts with ratio of 3:1:1, for training, validation, and evaluation respectively. We use Adam [40] as the optimization algorithm for

Table 3: Results for structured data.

Dataset	Model F_1 Score					ΔF_1
	SIF	RNN	Attention	Hybrid	Magellan	
BeerAdvo-RateBeer	58.1	72.2	64.0	72.7	78.8	-6.1
iTunes-Amazon$_1$	81.4	88.5	80.8	88.0	91.2	-2.7
Fodors-Zagats	100	100	82.1	100.0	100	0.0
DBLP-ACM$_1$	97.5	98.3	98.4	98.4	98.4	0.0
DBLP-Scholar$_1$	90.9	93.0	93.3	94.7	92.3	2.4
Amazon-Google	60.6	59.9	61.1	69.3	49.1	20.2
Walmart-Amazon$_1$	65.1	67.6	50.0	66.9	71.9	-4.3
Clothing$_1$	96.6	96.8	96.6	96.6	96.3	0.5
Electronics$_1$	90.2	90.6	90.5	90.2	90.1	0.5
Home$_1$	87.7	88.4	88.7	88.3	88.0	0.7
Tools$_1$	91.8	93.1	93.2	92.9	92.6	0.6

Table 4: Results for textual data (w. informative attributes).

Dataset	Model F_1 Score					ΔF_1
	SIF	RNN	Attention	Hybrid	Magellan	
Abt-Buy	35.1	39.4	56.8	62.8	43.6	19.2
Clothing$_2$	84.7	85.3	85.0	85.5	82.5	3.0
Electronics$_2$	90.4	92.2	91.5	92.1	85.3	6.9
Home$_2$	84.5	85.5	86.1	86.6	82.3	4.3
Tools$_2$	92.9	94.5	93.8	94.3	90.2	4.3

Table 5: Results for textual data (w.o. informative attributes).

Dataset	Model F_1 Score					ΔF_1
	SIF	RNN	Attention	Hybrid	Magellan	
Abt-Buy	32.0	38.5	55.0	47.7	33.0	22.0
Company	71.2	85.6	89.8	92.7	79.8	12.9
Clothing$_2$	84.6	84.4	84.6	84.3	78.8	5.8
Electronics$_2$	89.6	90.4	90.8	91.1	82.0	9.1
Home$_2$	84.0	84.8	83.7	85.4	74.1	11.3
Tools$_2$	91.6	92.5	92.6	93.0	84.4	8.6

all DL models for 15 training epochs. The validation set is used to select the best model snapshot for evaluation after each epoch to avoid over-fitting (see our technical report [55] for more details on training DL models). To apply Magellan to a dataset, we use the same 3:1:1 data split. We train five classifiers (decision tree, random forest, Naive Bayes, SVM and logistic regression), use the validation set to select the best classifier, and then evaluate on the evaluation set. It is important to note that Magellan uses the tuple attributes to automatically generate a large set of features used during training.

5.1 Experiments with Structured Data

Table 3 shows the accuracy of the four DL models and Magellan on the 11 structured datasets. We use red font to highlight the highest score in each row. The last column shows the relative increase in F_1 for the best DL model vs Magellan (see detailed in Appendix B.2).

The results show that DL models perform comparably with Magellan. The best DL model outperforms Magellan in 8 out of 11 cases. But the gain is usually small (less than 0.7%, see the last column), except 2.4% for DBLP-Scholar$_1$ and 20.2% for Amazon-Google (AG). This is because the product titles across matching pairs from the source datasets (Amazon and Google) correspond to synonyms of one another. That is, they are semantically similar but have large string similarity distances. This makes it difficult for Magellan to capture the similarity between mentions as it mainly relies on string similarity based features. However, DL can identify the semantic similarities across data instances to achieve a 20% F_1 gain. In the case of Walmart-Amazon$_1$, however, this ability to learn rich information from data hurts DL, because the model overfits due to the quirks of the training set, thus reaching 4.3% less F_1 compared to Magellan. This is less of an issue for Magellan which has a more restricted search space (as it uses string similarity-based features).

Hybrid performs the best among the four DL models. As discussed previously, DeepER, the current DL solution for EM [18], is comparable to SIF and RNN. Our results suggest that these models are not adequate, and that we should explore more sophisticated models, such as Hybrid, to obtain the highest possible EM accuracy.

Moreover, DL does appear to need a large amount of labeled data to outshine Magellan. The first three datasets (in Table 3) have only 450-946 labeled examples. Here DL performs worse than Magellan, except on Fodors-Zagats, which is easy to match. The next four datasets have 12.3K-28.7K labeled examples. Here DL outperforms Magellan in two cases. It performs reliably better than Magellan in the last four datasets, which have about 249K labeled examples.

Table 6: Results for dirty data.

Dataset	Model F_1 Score					ΔF_1
	SIF	RNN	Attention	Hybrid	Magellan	
iTunes-Amazon$_2$	66.7	79.4	63.6	74.5	46.8	32.6
DBLP-ACM$_2$	93.7	97.5	97.4	98.1	91.9	6.2
DBLP-Scholar$_2$	87.0	93.0	92.7	93.8	82.5	11.3
Walmart-Amazon$_2$	43.2	39.6	53.8	46.0	37.4	16.4
Home$_3$	82.8	86.4	88.0	87.2	68.6	19.4
Tools$_3$	88.5	92.8	92.6	92.8	76.1	16.7

5.2 Experiments with Textual Data

Textual datasets have few attributes, all of which are text blobs. For 5 out of 6 textual datasets listed in Table 2, we observe that they contain an "informative" attribute that packs a lot of information (i.e., having a high "signal-to-noise" ratio), which is title (containing brand name, product model, etc.). For these datasets, we expect Magellan to do well, as it can use similarity measures such as Jaccard to compare this attribute. Table 4 shows that this is indeed the case. Yet DL's F_1-score is 3.0-19.2% higher than Magellan's.

If we remove this "informative" attribute, the gain increases from 5.8 to 22.0% relative F_1, as shown in Table 5. One textual dataset, Company, has no such informative attribute. For this dataset, DL outperforms Magellan, achieving 92.7% F_1 vs 79.8% F_1.

The results suggest that DL outperforms Magellan for textual EM. They suggest that Hybrid is still the best among the DL models. We also find that the F_1-score difference between the best performing attention-based model (Attention or Hybrid) and the best performing model that does not use soft alignment (SIF and RNN) is around 4.5 points on average but goes up to 23.5 points (i.e., for the Abt-Buy dataset in Table 4). To understand this better, we investigated examples from datasets where the F_1 between the two types of methods is large. We found that in all cases the two data instances corresponded to misaligned word sequences similar to those found in the problem of text entailment. An example is matching sequences "samsung 52 ' series 6 lcd black flat panel hdtv ln52a650" and "samsung ln52a650 52 ' lcd tv".

5.3 Experiments with Dirty Data

Recall that dirty datasets are those where some attribute values are not in their correct cells but in the cells of some other attributes (e.g., due to inaccurate extraction). In such cases, we can ignore the "attribute boundaries" and treat the entire tuple as a single text blob, then apply DL, as in the textual case. The results in Table 6 show that DL significantly outperforms Magellan, by 6.2-32.6% relative F_1. Interestingly, even in the presence of extensive noise, for four out of six dirty datasets, Hybrid and Attention still perform only at most 1.1% lower in F_1 compared to their scores for the corresponding structured datasets. This suggests that DL is quite robust to certain kinds of noise (e.g., moving attribute values around).

Table 7: F_1-score for Hybrid with different language representations.

Attribute Embedding		Structured		Textual		Dirty	
		Home$_1$	Tools$_1$	Company	Home$_2$	Home$_1$	Tools$_1$
Pre-trained	Glove	86.5	86.5	93.5	86.6	87.1	88.3
	fastText	88.3	92.9	92.7	86.6	87.2	92.8
Char-based learned		88.2	92.8	87.7	86.9	87.5	93.8

Table 8: Train time comparison for different deep learning solutions and Magellan for different dataset types and sizes.

Dataset		SIF	RNN	Attention	Hybrid	Magellan
Structured	small	3-70s	5-15m	7-25m	10-45m	1s
	large	25m	6.5-7h	7-7.5h	9.5-11h	2-4m
Textual	small	15s	5m	7.5m	15m	1s
	large	8-16m	3-6h	3-6h	7-10h	9-12s
Textual w/o info. attr.	small	13s	4m	5m	10m	1s
	large	6-9m	3-3.5h	3-3.5h	6.5-9h	8-12s
Dirty	small	3-30s	2.5-7m	3-10m	5-20m	1s
	large	5m	25-35m	40-55m	1-1.5h	2-4m

5.4 Trade-offs for Deep EM

We now validate that the different design choices described in Section 3.1 have an impact on the performance of deep learning for entity matching. We report on trade-offs related to all design choices introduced by our architecture template.

5.4.1 Language Representation Selection. We validate that different types of language representations lead to DL models with different performance and that no single option dominates the others. We run our Hybrid model over structured, textual, and dirty data and compare different language representations with respect to (1) the granularity of the embeddings (i.e., word vs. character) and (2) pre-trained vs. learned. The results are shown in Table 7.

Word vs. Character: We compare GloVe [61] (a word-level embedding) with fastText [6] (a character-level embedding). Table 7 shows that the F_1-scores are similar but there are two exceptions. (1) On Tools fastText achieves 6% higher F_1-scores. This is because Tools' vocabulary include domain-specific words not present in the vocabulary of GloVe. Recall that GloVe maps all OOV words to the same embedding. FastText can approximate the embedding of those words by using a character-level strategy. (2) On Company GloVe outperforms fastText, though the by less than one point F_1. Recall that the entries in Company are obtained from Wikipedia which corresponds to the corpus used to train GloVe. In general, we find that character-level embeddings often obtain higher F_1-scores than word-level embeddings.

Pre-trained vs. Learned: We compare fastText with a character-level embedding trained from scratch. Table 7 shows that the two obtain similar F_1-scores with two exceptions. (1) The learned-from-scratch model is better for Tools, suggesting that learning an embedding from scratch is beneficial for highly-specialized datasets. (2) fastText achieves 5% higher F_1 on Company, demonstrating the effect of limited training data when learning an embedding from scratch. Overall, we find that training an embedding from scratch does not lead to performance improvements unless we perform EM over domains with highly specialized vocabularies.

5.4.2 Attribute Summarization Selection. So far we have shown that attribute summarization methods with cross-sequence alignment (Attention and Hybrid) outperform simpler methods (SIF and RNN). We now validate the *accuracy vs. training time* trade-off with respect to the attribute summarization used by different DL solutions. The factors that affect this trade-off are the complexity of the model and the size of the input dataset.

Table 9: F_1-score comparison for different deep learning solutions and Magellan for different dataset types and sizes.

Dataset		SIF	RNN	Attention	Hybrid	Magellan
Structured	small	79.1	83.9	76.2	84.6	86.5
	large	91.0	91.8	91.8	91.5	91.3
Textual	small	35.1	39.4	56.8	62.8	43.6
	large	87.5	88.9	88.8	89.3	83.9
Textual w/o info. attr.	small	32.0	38.5	55.0	47.7	33.0
	large	86.1	86.9	88.4	89.7	80.2
Dirty	small	76.8	86.2	78.2	84.2	64.7
	large	85.6	89.6	90.3	90.0	72.3

Table 10: F_1-score as we vary attribute comparison choices.

Model	Comparison	Abt-Buy	W-A$_1$	Home$_1$
SIF	Concatenation	22.6	34.7	83.8
	Element-wise Abs. Diff.	35.1	60.6	87.7
RNN	Concatenation	25.9	27.0	86.8
	Element-wise Abs. Diff.	38.5	67.6	88.4
Attention	Concatenation	54.9	50.0	88.7
	Element-wise Abs. Diff.	36.0	65.9	87.6
Hybrid	Concatenation	64.7	60.0	86.0
	Element-wise Abs. Diff.	39.3	67.1	86.7

The complexity of a DL solution depends on the attribute summarization choice in our DL architecture template. In general, the more expressive (complex) a model is, the higher its accuracy will be, but with longer training time. To validate the accuracy vs. training time trade-off we ran the four DL methods and Magellan while varying the size of the input dataset from "small" to "large". The results are shown in Tables 8 and 9.

We observe that the F_1-score gap between attribute summarization methods that perform cross-sequence alignment (Attention and Hybrid) and those that do not (SIF and RNN) decreases as the size of the input datasets increases. The F_1-score difference is up to 8× larger for small datasets than for large datasets—23.5 vs. 2.8 F_1 point difference. On the other hand, the training time for cross-sequence alignment models increases dramatically for large datasets (it sometimes exceeds 10 hours).

We attribute the above results to the soft alignment performed by attention mechanisms. Word embeddings coupled with a soft-alignment mechanism already capture similarity and comparison semantics. On the other hand, mechanisms that encode each input entity mention in isolation (SIF and RNN) rely only on the final classifier module to capture similarity and comparison semantics. We conjecture this to be the reason why methods that perform soft alignment are superior in the presence of little training data (i.e., small datasets). However, with more training data SIF and RNN are more attractive as they take far less time to train.

5.4.3 Attribute Comparison Selection. We validate the effect of different attribute comparison functions (see Section 3.4) on the performance of DL models. We fix the attribute summarization strategy for the four solutions discussed in Section 4 and for each solution we vary the attribute comparison function used. We find that fixed distance functions perform poorly overall. Thus, we focus on the results for learnable distance functions. Specifically, we evaluate the performance of (1) concatenation, where the final classifier

is responsible for learning the semantics of a distance between the entity mention encodings, and (2) element-wise absolute distance, where the features to the final classifier already capture the semantics of distance. Table 10 shows the results. We observe that for methods without cross-sequence alignment, using an element-wise comparison leads to F_1 improvements of up to 40% F_1 (see the results for SIF and RNN). For methods with cross-sequence alignment, however, there is no dominating option.

5.5 Micro-benchmarks

We perform micro-benchmark experiments to evaluate: (1) the effect of training data on the accuracy of models, (2) the sensitivity of DL to noisy labels, (3) how DL models compare to domain-specific approaches to EM, and (4) how different variations in the DL architecture, such as different dropout levels, using multiple layers, etc. affect the performance of DL. We find that Hybrid—the most expressive out of all DL models—is more effective at exploiting the information encoded in training data, DL is significantly more robust to noise than traditional learning techniques (e.g., used in Magellan), DL methods are competitive to domain-specific methods when we have access to 10K training examples or more, the performance of DL is robust to variations in the type of RNN network used (e.g., LSTM vs. GRU), the dropout levels, and the number of layers in the recurrent part of the architecture (see more in Appendix B.2).

6 DISCUSSION

6.1 Understanding What DL Learns

To gain insights into why DL outperforms Magellan on textual and dirty data, we focus on Hybrid and use first derivative saliency [38, 47, 48, 74] to analyze what it learns. Saliency indicates how sensitive the model is to each word in the string sequence of an entity mention. We expect words with higher saliency to influence the model output more. This is an indirect measure of how important a word is for Hybrid's final prediction. We consider "Home" and "Company" and compute the importance of each word in one attribute. Overall, we find that Hybrid was able to assign high weights to tokens that carry important semantic information, e.g., serial numbers of products, names of locations or people associated with an entity and special entity-specific attributes such as patterns or product color (see Figure 8 in the Appendix). We obtained similar results for all datasets. Finally, we find that the errors made by DL models are mostly due to (1) linguistic variations of domain-specific terms, (2) missing highly-informative tokens, and (3) tokens that are similar but semantically different. A detailed discussion can be found in our technical report [55].

6.2 Challenges and Opportunities

We discuss challenges (C) and opportunities (O) for DL for EM.

(C1) DL for Structured Data: Overall, we find the advantages of sophisticated DL methods to be limited when operating over clean, structured data, in that simpler learning models (e.g., logistic regression, random forests) often offer competitive results. This topic will need more empirical evaluation before we can reach a solid conclusion.

(C2) Scalability vs. Accuracy: For textual or dirty data, we find that complex DL offer significant accuracy improvements over existing state-of-the-art EM methods, but often require far longer training time. Their poor scalability will need to be addressed. (In addition, we find that given a large amounts of training data one can leverage simpler and faster DL solution without significant losses in accuracy.)

(C3) The Value of Training Data: As expected, DL models often require large amounts of training data to achieve good performance. Obtaining large amounts of training examples can be resource-intensive in many practical scenarios. Recent work on weak supervision [62] focuses on obviating the need for tedious and manual annotation of training examples. What makes this challenge unique to EM is that existing weak supervision approaches have focused primarily on textual data for tasks such as information extraction [62, 85], or visual data for tasks such as image classification [54]. As such, a fundamental challenge for DL-based EM is to devise new weak supervision methods for structured data as well as methods that are more robust to the class imbalance (between positive and negative examples) in EM.

(O1) DL and Data Integration: Our DL results suggest that DL can be promising for many problems in the broader field of data integration (DI), e.g., data cleaning, automated data extraction, data reformatting, and value canonicalization. Preliminary successes have already been reported in recent work [65, 85] but more effort to examine how DL can help DI is necessary.

(O2) Optimizers for DL Models: As showed in Section 5 there are several design choices with trade-offs when constructing DL models for EM, e.g., the choice of attribute summarization network, the kind of word embeddings, etc. An exciting future direction is to design simple rule-based optimizers that would analyze the EM task at hand and automate the deployment of such DL models. We can also explore how to use database-inspired optimization techniques to scale up DL models [83].

(O3) Semantic-aware DL: Our study revealed that DL has limited capability of capturing domain-specific semantics. A promising research direction is to explore mechanisms for introducing domain-specific knowledge to DL models. We envision this to be possible either via new weak-supervision methods [85] or by integrating domain knowledge in the architecture of DL models itself [11]. We also envision the design of new domain-specific representation learning models, such as domain-specific word embeddings for EM.

7 CONCLUSION

We examined the advantages and limitations of DL models when applied to a diverse range of EM tasks, specifically EM over structured, textual, and dirty data. We conducted a detailed experimental study that revealed the advantages of DL for EM, especially in the case of textual and dirty data. We also explored the design space for DL solutions for EM and studied the accuracy-efficiency trade-offs introduced by different choices in that space. Our study highlights several limitations and challenges associated with DL models, and outlines several open problems on how DL can push the boundaries of automated solutions for data integration related tasks.

Acknowledgment: This work is generously supported by @WalmartLabs, Google, Johnson Controls, UW-Madison UW2020 grant, NIH BD2K grant U54 AI117924, and NSF Medium grant IIS-1564282.

REFERENCES

[1] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings. ICLR.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. ICLR.

[3] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, et al. 2016. End-to-end attention-based large vocabulary speech recognition. IEEE ICASSP.

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A Neural Probabilistic Language Model. *JMLR* (March 2003), 1137–1155.

[5] Mikhail Bilenko and Raymond J. Mooney. 2003. Adaptive Duplicate Detection Using Learnable String Similarity Measures. KDD.

[6] Piotr Bojanowski, Edouard Grave, Armand Joulin, et al. 2016. Enriching Word Vectors with Subword Information. *CoRR* abs/1607.04606 (2016).

[7] Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, et al. 2017. Recurrent neural network-based sentence encoder with gated attention for natural language inference. *CoRR* abs/1708.01353 (2017).

[8] Kyunghyun Cho et al. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. EMNLP.

[9] Peter Christen. 2012. *Data Matching*. Springer.

[10] Kevin Clark et al. 2016. Improving coreference resolution by learning entity-level distributed representations. *CoRR* abs/1606.01323 (2016).

[11] William W. Cohen. 2016. TensorLog: A Differentiable Deductive Database. *CoRR* abs/1605.06523 (2016).

[12] Ronan Collobert et al. 2011. Natural language processing (almost) from scratch. JMLR.

[13] R. Collobert, K. Kavukcuoglu, and C. Farabet. 2011. Torch7: A Matlab-like Environment for Machine Learning. In *BigLearn, NIPS Workshop*.

[14] Ido Dagan, Dan Roth, Fabio Zanzotto, and Graeme Hirst. 2012. *Recognizing Textual Entailment*. Morgan & Claypool Publishers.

[15] Sanjib Das et al. [n. d.]. The Magellan Data Repository. https://sites.google.com/site/anhaidgroup/useful-stuff/data. ([n. d.]).

[16] Bhuwan Dhingra, Hanxiao Liu, et al. 2017. A Comparative Study of Word Embeddings for Reading Comprehension. *CoRR* abs/1703.00993 (2017).

[17] Jens Dittrich. 2017. Deep Learning (m)eats Databases. VLDB Keynote.

[18] Muhammad Ebraheem, Saravanan Thirumuruganathan, et al. 2017. DeepER–Deep Entity Resolution. *CoRR* abs/1710.00597 (2017).

[19] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. 2007. Duplicate Record Detection: A Survey. *TKDE* 19, 1 (Jan. 2007), 1–16.

[20] Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. 2009. Reasoning About Record Matching Rules. VLDB.

[21] Matthew Francis-Landau et al. 2016. Capturing semantic similarity for entity linking with convolutional neural networks. *CoRR* abs/1604.00734 (2016).

[22] Octavian-Eugen Ganea and Thomas Hofmann. 2017. Deep Joint Entity Disambiguation with Local Neural Attention. *CoRR* abs/1704.04920 (2017).

[23] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. ICML.

[24] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity Resolution: Theory, Practice & Open Challenges. VLDB.

[25] Amir Globerson, Gal Chechik, Fernando Pereira, and Naftali Tishby. 2007. Euclidean Embedding of Co-occurrence Data. *JMLR* 8 (Dec. 2007), 2265–2295.

[26] Chaitanya Gokhale, Sanjib Das, AnHai Doan, et al. 2014. Corleone: Hands-off Crowdsourcing for Entity Matching. SIGMOD.

[27] David Golub and Xiaodong He. 2016. Character-level question answering with attention. *CoRR* abs/1604.00727 (2016).

[28] Ian Goodfellow et al. 2016. *Deep Learning*. MIT Press.

[29] Alex Graves, Santiago Fernández, et al. 2005. Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition. ICANN'05.

[30] Alex Graves and Navdeep Jaitly. 2014. Towards End-To-End Speech Recognition with Recurrent Neural Networks. ICML.

[31] Bert F. Green, Jr., Alice K. Wolf, Carol Chomsky, and Kenneth Laughery. 1961. Baseball: An Automatic Question-answerer. IRE-AIEE-ACM '61 (Western).

[32] Geoffrey Hinton, Li Deng, Dong Yu, et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[33] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[34] Elad Hoffer and Nir Ailon. 2015. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*. Springer.

[35] Hongzhao Huang et al. 2015. Leveraging deep neural networks and knowledge graphs for entity disambiguation. *CoRR* abs/1504.07678 (2015).

[36] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of Tricks for Efficient Text Classification. *CoRR* abs/1607.01759 (2016).

[37] Daniel Jurafsky and James H. Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (1st ed.). Prentice Hall PTR.

[38] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2015. Visualizing and understanding recurrent networks. ICLR Workshop.

[39] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. AAAI.

[40] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980 (2014).

[41] Pradap Konda et al. 2016. Magellan: Toward building entity matching management systems. VLDB.

[42] Pradap Konda et al. 2018. Magellan: Toward Building Entity Matching Management Systems (SIGMOD Research Highlight). *SIGMOD Record* (2018).

[43] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. VLDB.

[44] Hanna Köpcke, Andreas Thor, Stefan Thomas, and Erhard Rahm. 2012. Tailoring Entity Resolution for Matching Product Offers. EDBT.

[45] Rémi Lebret et al. 2014. Word Embeddings through Hellinger PCA. EACL.

[46] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[47] Jiwei Li et al. 2016. Visualizing and Understanding Neural Models in NLP. NAACL.

[48] Jiwei Li, Will Monroe, and Dan Jurafsky. 2016. Understanding Neural Networks through Representation Erasure. *CoRR abs/1612.08220* (2016).

[49] Yang Liu et al. 2016. Learning natural language inference using bidirectional LSTM model and inner-attention. *CoRR* abs/1605.09090 (2016).

[50] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. EMNLP.

[51] Christopher Manning. 2017. Representations for Language: From Word Embeddings to Sentence Meanings. https://simons.berkeley.edu/talks/christopher-manning-2017-3-27. (2017).

[52] Tomas Mikolov, Ilya Sutskever, Kai Chen, et al. 2013. Distributed Representations of Words and Phrases and Their Compositionality. NIPS.

[53] Makoto Miwa and Mohit Bansal. 2016. End-to-End Relation Extraction using LSTMs on Sequences and Tree Structures. ACL.

[54] Volodymyr Mnih and Geoffrey E. Hinton. 2012. Learning to Label Aerial Images from Noisy Data. ICML.

[55] Sidharth Mudgal et al. 2018. *Deep Learning For Entity Matching: A Design Space Exploration*. Technical Report. http://pages.cs.wisc.edu/~anhai/papers/deepmatcher-tr.pdf.

[56] Felix Naumann and Melanie Herschel. 2010. *An Introduction to Duplicate Detection*. Morgan and Claypool Publishers.

[57] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. 2016. Learning text similarity with siamese recurrent networks. ACL.

[58] Massimo Nicosia and Alessandro Moschitti. 2017. Accurate Sentence Matching with Hybrid Siamese Networks. CIKM.

[59] George Papadakis, Jonathan Svirsky, Avigdor Gal, et al. 2016. Comparative Analysis of Approximate Blocking Techniques for Entity Resolution. VLDB.

[60] Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. EMNLP.

[61] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. EMNLP.

[62] Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, et al. 2017. Snorkel: Rapid Training Data Creation with Weak Supervision. VLDB.

[63] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.

[64] Ziad Sehili, Lars Kolb, Christian Borgs, Rainer Schnell, and Erhard Rahm. 2015. Privacy Preserving Record Linkage with PPJoin. BTW.

[65] Uri Shaham, Xiuyuan Cheng, Omer Dror, et al. 2016. A Deep Learning Approach to Unsupervised Ensemble Learning. ICML.

[66] Tao Shen et al. 2017. DiSAN: Directional Self-Attention Network for RNN/CNN-free Language Understanding. *CoRR* abs/1709.04696 (2017).

[67] Wei Shen, Jianyong Wang, and Jiawei Han. 2015. Entity linking with a knowledge base: Issues, techniques, and solutions. *TKDE* 27, 2 (2015), 443–460.

[68] Rohit Singh, Vamsi Meduri, Ahmed Elmagarmid, et al. 2017. Generating Concise Entity Matching Rules. SIGMOD.

[69] Parag Singla et al. 2006. Entity Resolution with Markov Logic. ICDM.

[70] Richard Socher et al. 2013. Parsing with compositional vector grammars. ACL.

[71] Richard Socher et al. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. EMNLP.

[72] Rupesh Kumar Srivastava et al. 2015. Highway networks. ICML.

[73] Michael Stonebraker, Daniel Bruckner, Ihab F. Ilyas, et al. 2013. Data Curation at Scale: The Data Tamer System. CIDR.

[74] Hendrik Strobelt et al. 2016. Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks. *CoRR abs/1606.07461* (2016).

[75] Yaming Sun, Lei Lin, Duyu Tang, et al. 2015. Modeling Mention, Context and Entity with Neural Networks for Entity Disambiguation. IJCAI.

[76] Ilya Sutskever. 2013. *Training recurrent neural networks*. Ph.D. Dissertation. University of Toronto.

[77] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. NIPS.

[78] Ming Tan et al. 2016. Improved Representation Learning for Question Answer Matching. ACL.

[79] Ashish Vaswani et al. 2017. Attention Is All You Need. NIPS.
[80] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. 2016. Matching networks for one shot learning. ACL.
[81] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. 2012. CrowdER: Crowdsourcing Entity Resolution. VLDB.
[82] Shuohang Wang and Jing Jiang. 2017. A Compare-Aggregate Model for Matching Text Sequences. ICLR.
[83] Wei Wang et al. 2016. Database Meets Deep Learning: Challenges and Opportunities. *ACM SIGMOD Record* 45, 2 (2016), 17–22.
[84] Sam Wiseman, Alexander M. Rush, and Stuart M. Shieber. 2016. Learning Global Features for Coreference Resolution. NAACL.
[85] Sen Wu, Luke Hsiao, Xiao Cheng, et al. 2017. Fonduer: Knowledge Base Construction from Richly Formatted Data. *CoRR* abs/1703.05028 (2017).
[86] Wenpeng Yin et al. 2016. Simple Question Answering by Attentive Convolutional Neural Network. COLING.
[87] Wenpeng Yin, Mo Yu, Bing Xiang, et al. 2016. Simple question answering by attentive convolutional neural network. *CoRR* abs/1606.03391 (2016).
[88] Radu Florian Zhiguo Wang, Wael Hamza. 2017. Bilateral Multi-Perspective Matching for Natural Language Sentences. IJCAI.

A SOFT-ALIGNMENT WEIGHTS

We describe how we compute the soft-alignment matrix for Attention and Hybrid in Section 4. Let $\mathbf{u}_1$ and $\mathbf{u}_2$ be the primary input and context respectively. Let K and M be the total number of elements in each sequence. For each pair of entries $u_1[k] \in \mathbf{u}_1$ and $u_2[m] \in \mathbf{u}_2$ we compute a weight $w_{k,m}$ as follows:

(1) We first obtain a d-dimensional representation for $u_1[k]$ and $u_2[m]$ by using a two layer HighWay net with ReLU:

$$\mathbf{q}_{1,k} = h(u_1[k])$$
$$\mathbf{q}_{2,m} = h(u_2[m])$$

where h denotes the HighWay net and $\mathbf{q}_{1,k}$ and $\mathbf{q}_{2,m}$ denote the hidden representations for $u_1[k]$ and $u_2[m]$.

(2) We obtain an unnormalized score $s_{k,m}$ by taking the dot product of $\mathbf{q}_{1,k}$ and $\mathbf{q}_{2,m}$, i.e., $s_{k,m} = \mathbf{q}_{1,k}^T \mathbf{q}_{2,m}$.

(3) We normalize all weights using soft-max:

$$w_{k,m} = \frac{\exp(s_{k,m})}{\sum_{i=1}^{M} \exp(s_{k,i})}$$

B EMPIRICAL EVALUATION

B.1 Experiment setup

Dataset Creation: We provide more details on the creation of the datasets shown in Table 2. There are three types of datasets: structured, textual and dirty. We use 11 structured datasets in different sizes and domains in this paper. The first 7 are smaller datasets from [15] except for "Amazon-Google" which is from [43]. Each dataset contains two tables that need matching. Since we only focus on the matching step in EM, we first apply blocking using [41] to get a candidate set. Next, we obtain the correct labels for all pairs in the candidate set which is then taken as the final dataset to work on. The last four large product datasets are from a major product retailer. Since the datasets are already in the after-blocking candidate set format with all pairs labeled, we can directly use them.

For textual EM, we use 6 datasets. The first small dataset Abt-Buy is from [43]. Since it contains two tables, we also first conduct blocking and then label the candidate set. "Company" is a dataset we created, consisting of pairs (a, b), where a is the text of a Wikipedia page describing a company and b is the text of a company's homepage. We created matching pairs in this dataset by crawling Wikipedia pages describing companies, then following company URLs in those pages to retrieve company homepages. To generate the non-matching pairs, for each matching pair (a, b), we fix a and form two negative pairs (a, b_1) and (a, b_2), where b_1 and b_2 are the top-2 most similar pages other than b in the company homepage collection, calculated by word-based Jaccard similarity . This dataset will be publicly released. The last four textual product datasets are from the same retailer as mentioned above.

For dirty EM we also use 6 datasets. For this data type we want to mimic high data variations in real EM problems. In this paper, we specifically focus on one very common type of dirtiness, which is that some attribute values are sprinkled in others, e.g., the value of attribute "brand" is embedded in "title" while leaving the correct value cell empty. This is very common due to imperfect IE methods. The six dirty datasets are all derived from the corresponding structured datasets described above. To generate the datasets, for each attribute other than "title", we randomly move each value to the attribute "title" in the same tuple with 50% probability. This simulates a common problem in dirty data seen in the wild while keeping the modifications simple.

B.2 Additional experiments

Detailed Evaluation Tables: In our technical report [55], we present the detailed F_1 scores summarized in Section 5 for all types of EM tasks considered (i.e., structured, textual, and dirty).

Varying the Size of Training Set: We analyze the sensitivity of different entity matching methods to changes in the amount of available training data. For each of the three types of datasets considered, we pick two large representative datasets and analyze how the performance of two DL models and Magellan varies as we change the size of training data. The results are shown in Figure 6. For each dataset, we keep the ratio of training set size to validation set size constant (used 3:1 as we discussed in Section 5), and vary the total number of entity pairs in these two sets (called dataset size from here on) by sampling from the original large datasets. We pick the best DL model considered, i.e., the hybrid model and the simplest and fastest DL model, i.e., the SIF model, for this analysis.

In the case of structured data, we see that Magellan outperformed the hybrid DL model when the dataset size is less than 50K. With more data, the hybrid model starts becoming comparable to Magellan and largely stays this way, until the dataset size reaches 200K at which point it slightly outperforms Magellan.

For textual data we picked dataset "Home-Textual" with an atomic informative attribute ("Title") and the purely textual dataset "Company". In the first case, we see that DL starts outperforming Magellan even with a dataset size of 1K but the difference is not very significant. With a few thousand instances, the difference becomes more significant. In the purely textual case, we see that Magellan quickly attains a relatively high F_1 score with a dataset size of just 1K, due to its heuristic-based string similarity features, while the hybrid DL model lags behind. It takes nearly 10K data instances before the hybrid model starts outperforming Magellan. With more data its performance steadily continues to increase until we finally exhaust our set of labeled data instances.

For dirty structured data, Magellan initially outperforms DL when the dataset size is only 1K, but DL starts outperforming Magellan when a few thousand training instances are available.

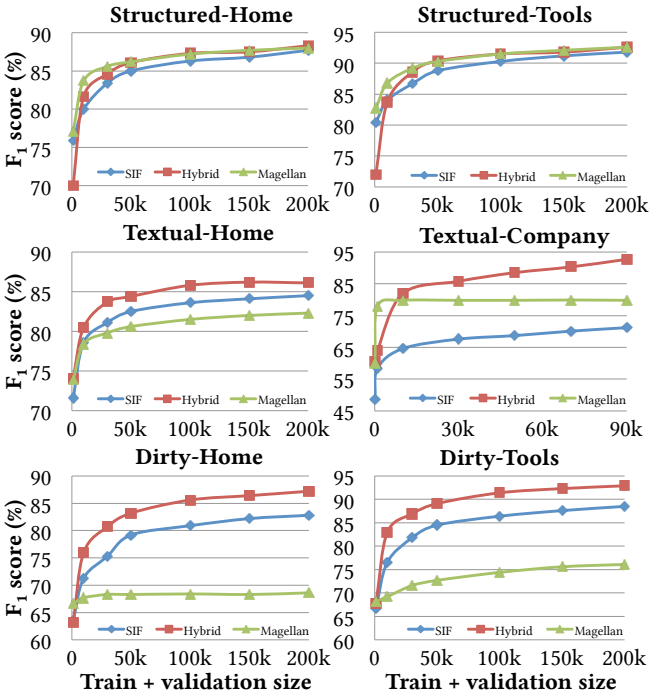

Figure 6: Varying the training size.

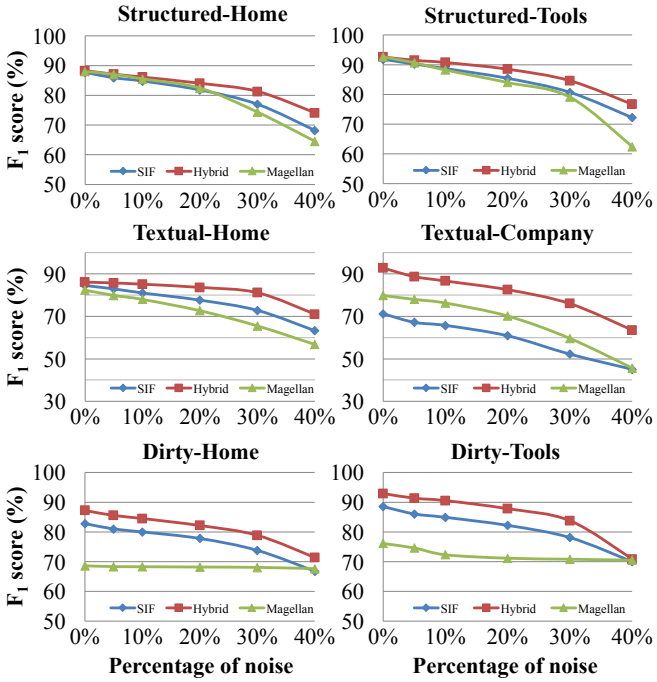

Figure 7: Varying label noise.

Home	(a)	any	manufacturing	defects	li	colonial	mills	twilight	palm	area	rug	home	&	garden	features	technique
	(b)	silver	metal	armless	ships	individually	tempo	vinyl	tp1349	br	features	ul	li	stationary	stool	li
	(c)	height	top	to	bottom	16	inches	li	li	overall	width	side	to	side	16	inches
Company	(d)	group	started	about	70	years	ago	at	kyoto	uzumasa	japan	the	center	of	the	japanese
	(e)	the	deltic	group	exciting	and	entertaining	venues	the	deltic	group	exciting	and	entertaining	venues	menu
	(f)	b.	riley	financial	on	july	1	2016	robert	j.	taragan	became	the	new	ceo	of

Figure 8: Saliency scores indicating the importance of words in six entities. Darker colors mean higher importance.

Robustness to Label Noise: We introduce noise in the match / non-match labels assigned to entity pairs in six EM datasets, two from each EM category considered. We picked the same six datasets as in the previous experiment varying training set size. The results are shown in Figure 7. For each dataset, we introduce label noise in the range of 0 - 40%. For example, for the case with 20% noise, we flip the labels of a randomly selected 20% subset of the entity pairs in the dataset.

In all cases we see that the hybrid model is fairly robust to noise especially until 30% noise, after which we see a steeper drop. On structured and textual datasets, we note that the performance gap between the hybrid model and Magellan increases as the noise increases indicating that the hybrid model is more robust to noise.

Comparison to Domain-Specific EM Approaches: We would like to understand how DL models compare to domain specific (DS) EM approaches involving manual information extraction and feature engineering. To do so, we take four product datasets from each of the three EM categories described in Section 2.1. For each dataset, we compare the performance of our best DL model considered with rigorous DS approach based EM.

We perform several experiments, the results of which are shown in Table 12. In the "Approach" column we describe how the domain specific EM was performed. In general, for each dataset we first perform IE and feature engineering, then train a machine learning based system (Magellan) using these features.

Note that in the case of textual datasets we make use of structured information extracted from the textual attributes by data analysts over *several months*. We also make use of relevant ideas discussed in [44] to perform our own extraction and feature engineering which took multiple days.

We see that domain specific extraction does help as compared to the DL model without domain specific features. However, the average improvement is only 0.8%, 1.1% and 1.4% respectively for structured, textual and dirty datasets. The DL model was able to approach quite close to the performance of months of intensive domain specific EM effort with less than half a day of training.

We also perform additional experiments to investigate how DL models compare to domain specific approaches in the presence of limited training data. We vary the training data size in the same way as described in the paragraph “Varying the Size of Training Set” above. The results in Figure 9 indicate that with very limited training data, DS EM approaches are very helpful. However, with a few 10s of thousands of labeled data instances the DL model catches up — with 10K labeled instances, the hybrid DL model is within

Table 11: Model variations.

RNN Unit & Layers	Layer Stacking	Dropout		Attention	Structured		Textual		Dirty	
		Probability	Location		Home	Tools	Company	Home	Home	Tools
1L GRU	Standard	0	None	Standard	88.3	92.9	92.7	86.6	87.2	92.8
2L GRU	Standard	0	None	Standard	87.6	92.5	92.8	86.4	87.1	92.5
1L LSTM	Standard	0	None	Standard	88.1	92.7	93.3	86.4	87.8	93.5
2L LSTM	Standard	0	None	Standard	87.5	92.0	93.7	85.8	88.3	93.0
2L GRU	Residual	0	None	Standard	88.0	93.0	92.5	86.7	87.3	93.2
2L GRU	Highway	0	None	Standard	88.0	93.4	93.0	86.6	87.6	93.3
1L GRU	Standard	0.05	Before RNN	Standard	88.0	93.7	92.4	85.7	87.6	93.0
1L GRU	Standard	0.2	Before RNN	Standard	87.5	93.4	92.7	84.8	88.5	93.8
1L GRU	Standard	0.05	After RNN	Standard	88.1	92.8	92.4	86.4	87.3	92.9
1L GRU	Standard	0.2	After RNN	Standard	87.7	92.6	91.8	86.0	87.1	92.9
2L GRU	Standard	0.05	Between RNN layers	Standard	87.7	92.5	92.4	86.2	86.8	92.5
2L GRU	Standard	0.2	Between RNN layers	Standard	87.3	92.3	92.5	85.6	87.9	92.5
1L GRU	Standard	0	None	2 head attention	88.3	92.8	92.5	86.6	87.2	92.8
1L GRU	Standard	0	None	Scaled dot product	88.3	92.8	87.0	86.5	86.7	92.7
2L LSTM	Standard	0.2	Between RNN layers	Standard	87.3	92.1	93.4	85.2	88.6	93.2
2L LSTM	Highway	0	None	Standard	88.2	92.7	93.5	86.5	88.5	94.0
1L LSTM	Standard	0.2	Before RNN	Standard	87.2	93.0	93.3	84.2	88.8	94.3

Table 12: Comparison to domain specific approaches.

Type	Dataset	DL-Hybrid	DS-Magellan	ΔF_1	DS Approach
Structured	Clothing	96.6	96.5	-0.1	(1) Create domain specific features. (2) Train classifiers using Magellan.
	Electronics	90.2	91.3	1.1	
	Home	88.3	89.3	1.0	
	Tools	92.9	94.0	1.1	
Textual	Clothing	85.5	89.2	3.7	(1) Perform IE to extract attributes from text. (2) Create domain specific features. (3) Train classifiers using Magellan.
	Electronics	92.1	90.9	-1.2	
	Home	86.6	89.0	2.4	
	Tools	94.3	93.9	-0.4	
Dirty	Clothing	96.3	96.5	0.2	(1) Clean all dirty attributes. (2) Create domain specific features. (3) Train classifiers using Magellan.
	Electronics	89.0	91.3	2.3	
	Home	87.2	89.3	2.1	
	Tools	92.8	94.0	1.2	

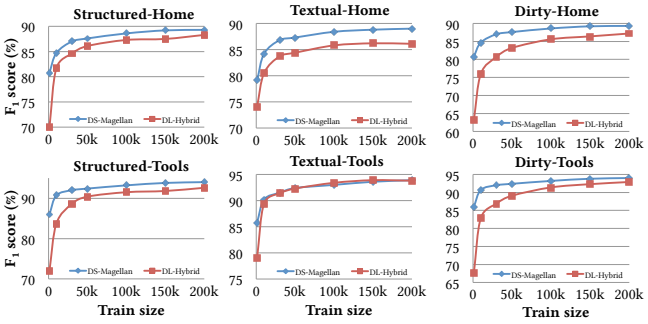

Figure 9: Varying the training size (domain specific).

5.1 percent points of the DS approach on average, and within 2.9 percent points on average with 50K labeled data.

Performance of DL Model Variations: As with most DL models, the models we presented in Section 4 can be altered along several dimensions to obtain variants. We analyzed several variants of the best model we considered, i.e., the hybrid model in order to determine its optimal configuration. To do so we altered the model along 4 primary dimensions and evaluated 17 variants of the hybrid model on six datasets, two from each EM category discussed in Section 2.1. We picked the same six datasets as in the experiment varying training set size. We present the F_1 scores corresponding to each variant for each dataset considered in Table 11. In order to efficiently determine the optimal setting from the large space of configurations formed by the 4 primary dimensions, we initially make the assumption that all dimensions are independent of each other, and vary each dimension one by one to form the first 14 variants listed in Table 11. After this set of experiments, we altered multiple dimensions concurrently, based on the best configurations we observed for each dimension independently. The last 3 rows in Table 11 show the results for these.

We note that no single configuration is universally much better than the most basic variant of the hybrid model (standard 1 layer GRU, highlighted in blue in Table 11). The best setting compared to this variant, the 2 layer LSTM with highway stacking (second last row in Table 11), is only 0.5% better on average across the six datasets. Moreover, the maximum improvement in F_1 score by any variant compared to the most basic setting across the six datasets is only 1.6%. Hence for our analysis we only considered the simplest setting of the hybrid model, with no bells and whistles, to keep the exposition straightforward and to avoid unnecessary complexity. However, in practical application scenarios, the model variation dimensions can be treated as hyperparameters and the best configuration can be automatically discovered using hyperparameter optimization tools.

Comparing with Other Ways of Formulating EM: As discussed in Section 2.1, the triplet framework [34] can be adapted for EM. We have done so and compared it with the hybrid model, on two structured datasets (Home and Tools), two textual datasets (Company and Home), and two dirty datasets (Home and Tools). The triplet solution performs significantly worse than Hybrid in all six cases, with a F_1 score difference ranging from 2.5% to 21.1%, or 7.6% lower on average. This is likely due to the fact that our DL model directly optimizes the DL model to maximize the classification accuracy, whereas the triplet approach focuses on obtaining better hidden representations of entities, even at the cost of classification performance. We have also empirically found that the solution in [57] performs worse than Hybrid and RNN (e.g., by 4.1-9.6% compared to Hybrid). We omit further details for space reasons and refer the reader to the technical report [55].