Knowledge Transfer for Entity Resolution with Siamese Neural Networks

MICHAEL LOSTER, IOANNIS KOUMARELAS, and FELIX NAUMANN,
Hasso Plattner Institute, University of Potsdam, Germany

The integration of multiple data sources is a common problem in a large variety of applications. Traditionally, handcrafted similarity measures are used to discover, merge, and integrate multiple representations of the same entity—duplicates—into a large homogeneous collection of data. Often, these similarity measures do not cope well with the heterogeneity of the underlying dataset. In addition, domain experts are needed to manually design and configure such measures, which is both time-consuming and requires extensive domain expertise.

We propose a deep Siamese neural network, capable of learning a similarity measure that is tailored to the characteristics of a particular dataset. With the properties of deep learning methods, we are able to eliminate the manual feature engineering process and thus considerably reduce the effort required for model construction. In addition, we show that it is possible to transfer knowledge acquired during the deduplication of one dataset to another, and thus significantly reduce the amount of data required to train a similarity measure. We evaluated our method on multiple datasets and compare our approach to state-of-the-art deduplication methods. Our approach outperforms competitors by up to +26 percent F-measure, depending on task and dataset. In addition, we show that knowledge transfer is not only feasible, but in our experiments led to an improvement in F-measure of up to +4.7 percent.

CCS Concepts: • **Information systems** → **Entity resolution**; **Deduplication**; • **Computing methodologies** → **Neural networks**; **Transfer learning**;

Additional Key Words and Phrases: Entity resolution, duplicate detection, transfer learning, neural networks, metric learning, similarity learning, data quality

ACM Reference format:
Michael Loster, Ioannis Koumarelas, and Felix Naumann. 2021. Knowledge Transfer for Entity Resolution with Siamese Neural Networks. *J. Data and Information Quality* 13, 1, Article 2 (January 2021), 25 pages.
https://doi.org/10.1145/3410157

1 DUPLICATE DETECTION

The need to integrate multiple data sources into a single dataset is present in many application areas. A major challenge that arises during the integration process emerges from the fact that records from different data sources often contain duplicate entries, i.e., several entries that refer to the same real-world entity. These duplicates, implying poor data quality, can directly affect

Authors' addresses: M. Loster, I. Koumarelas, and F. Naumann, Hasso Plattner Institute, University of Potsdam, Potsdam, Germany; emails: {michael.loster, ioannis.koumarelas, felix.naumann}@hpi.de.

1936-1955/2021/01-ART2 $15.00
https://doi.org/10.1145/3410157

downstream operations, e.g., causing low customer satisfaction in customer-relationship-management, incorrect stock-keeping, or overfitting of machine learning methods.

Resolving this issue requires the detection of such duplicates, which is well studied, also under the terms entity resolution, record linkage, and several others [14, 21, 43]. Typically, duplicate detection refers to the task of identifying duplicates *within* a single dataset, while record linkage refers to the detection of matching records *across* datasets. Conceptually, the two tasks are very similar, and we perform experiments for both. For ease of exposition, we speak of duplicate detection throughout the article. Duplicate detection is commonly divided into two subtasks: (i) the identification of duplicate candidates, and (ii) the classification of these candidates as duplicates and non-duplicates. The problem of subsequently eliminating these duplicates, known as data fusion [6], is not addressed in this work. Since most deduplication approaches depend heavily on the performance of the similarity measure used to assess the similarity between the processed entities, its selection or creation plays a crucial role and must be performed with great care.

So far, a wide variety of similarity measures have been proposed [14], each with its own strengths and weaknesses. As each of these proposed similarity measures takes its own individual approach in quantifying the similarity of two entity representations, they are often highly domain-dependent, making it difficult to conduct deduplication by relying on just one of these traditional measures. Therefore, several similarity measures, such as Levenshtein [37] or Jaro-Winkler [58], are often combined to capture the underlying characteristics of the processed entities. While this combination offers more flexibility, it is often difficult to sufficiently adapt the traditional similarity measures to the characteristics of the underlying entities.

To address this issue, there are essentially two possible solutions: either a custom-made similarity measure is designed manually, or a similarity measure can be learned by applying machine learning techniques. The main disadvantage of manually designing a similarity measure is that it requires extensive domain expertise, which makes it difficult and time-consuming. The alternative approach is to leverage machine learning techniques to learn a new similarity measure that is tailored specifically to the entities of a specific domain.

As is the case for most traditional machine learning techniques for duplicate detection, such as Support Vector Machines (SVMs) [5, 12] or decision trees [20, 55], the main bottleneck lies in their dependence on manual feature engineering. Due to the costs associated with extensive manual feature engineering, it is, in practice, often limited to the creation of feature vectors consisting of various traditional similarity measures, which are ultimately used to train a selected classifier. Although good results might be achieved by combining a number of similarity measures and weighting them optimally, this approach is still bound by the limitations of traditional similarity measures, which were developed for specific domains or data types. Overall, both the manual feature engineering process as well as the manual creation of similarity measures requires considerable amounts of time and effort and is therefore undesirable. It should also be noted that most machine learning techniques require large amounts of training data to train a classifier of sufficient quality. It can therefore be stated that both approaches have their disadvantages with regard to manual labor costs. While the bulk of time for manually creating a custom similarity measure is spent on iterating different design ideas, the majority of time spent on learning a custom similarity measure is used for annotating training data and designing a set of useful features.

To alleviate both issues, we propose SNNDedupe, a deep neural network that is based on the architecture of a Siamese neural network (SNN) [8], and combine it with a transfer learning approach. Using this architecture, we learn a specific similarity measure for a given dataset by letting the neural network automatically adapt to and capture the idiosyncrasies of the respective dataset. Thus, we overcome the limitations of traditional similarity measures, which often lack sufficient adaptability when applied to unintended data. At its core, the proposed SNN architecture consists of two identical subnetworks that are connected via an energy function. We selected this architecture as the basis for our approach, as it has been successfully applied to learn similarity functions in various areas [35, 42]. By using deep learning techniques, we are able to automatically discover promising features, thus eliminating both the costly manual feature engineering and the required domain knowledge needed for this task. In our experiments, we compare SNNDedupe with the SVM-based approach of Christen [13] and the DeepMatcher [41] system. We show that we are able to learn a competitive similarity measure that enables a state-of-the-art duplicate classification.

As annotating vast amounts of training data is in practice often infeasible, we investigate the behavior of SNNDedupe under different amounts of training data. To reduce the amount of training data and thus, the associated effort for their generation, we focus on the design and implementation of a knowledge transfer [32] between deduplication networks. In doing so, we address what is often perceived as the Achilles' heel of many machine learning approaches, namely, their dependence on large amounts of training data. This objective is reflected primarily in the design of the network, as it is tailored to support knowledge transfer at the attribute-level. In a nutshell, the core idea that makes knowledge transfer possible is to process the entities at their attribute level. This decision provides the ability to learn and transfer the properties of each attribute's semantic domain (e.g., band names, book titles). By combining character embeddings and BLSTM layers, we create a dedicated embedding for each attribute. Thus, representation learning takes place at the attribute level, resulting in embeddings for each attribute domain, which can informally be interpreted as knowledge about the processed attributes. This compressed domain knowledge, which accumulates in the weight matrices of the network layers, can then be transferred by initializing the attribute weights of a deduplication network that operates on a different dataset. This transfer can be carried out whenever the source and target domains of the corresponding attribute are either the same or sufficiently similar.

Our experiments show not only how a knowledge transfer affects the training of our network but also that it is possible to increase classification performance and reduce the amount of necessary new training data by transferring parts of the already acquired knowledge. In addition, we investigate how much improvement can be achieved over learning without prior knowledge and to what extent the training data can be reduced while maintaining a good result.

In particular, we make the following contributions:

- We present SNNDedupe, an SNN network architecture for deduplication that facilitates knowledge transfer. To achieve both competitive deduplication performance and knowledge transfer, it combines character embeddings as studied by DeepMatcher with DeepER's architectural design.
- We show how to transfer previously learned knowledge to support the deduplication of other datasets.
- We analyze the impact of transfer learning strategies on the performance of the neural network.
- We compare our system with the state-of-the-art of related work.

The remainder of this article is organized as follows: In Section 2, we discuss related work. Section 3 introduces SNNs and presents the overall structure of our neural network, as well as its unique characteristics. We introduce the concept of knowledge transfer and how it relates to our experiments in Section 4. Section 5 is dedicated to introducing the datasets and gold standards for our experiments. In Section 6, we present our experimental results and Section 7 concludes the article.

2 RELATED WORK

Related work can be classified according to the degree of supervision and whether transfer learning represents a key element of the respective approach. In the following, we therefore discuss the related work that does not involve transfer learning according to its supervision approach, which is divided into the following categories: supervised, unsupervised, and semi-supervised approaches. Thereafter, we separately discuss work that supports the concept of transfer learning.

Supervised Approaches. Apart from traditional similarity measures [14], efforts have been made to learn similarity measures using a range of supervised machine learning techniques. Systems based on these techniques require tagged training data from which they learn to perform a specific, well-defined task. For duplicate detection, this task is to classify a particular entity pair as either duplicate or non-duplicate.

Both Christen [12] as well as Bilenko and Mooney [5] address this problem by using a SVM. Bilenko and Mooney propose two similarity measures that are learned using two different machine learning techniques [5]. The first uses the Expectation-Maximization (EM) algorithm to learn a variant of the edit distance measure with affine gaps, the second is based on the vector space model and trained using a SVM. They were able to show that the learned similarity measures can be adapted to the underlying dataset, resulting in better system performance.

Another approach developed by Christen [13], also uses SVMs to train a classifier for detecting duplicates. Their feature engineering step consists of choosing and calculating different traditional similarity measures between the attribute values of each record pair. The resulting values are then combined into a feature vector, which is used to train the classifier. In addition to SVM-based systems, other studies have investigated the use of decision trees [20, 55].

More recently, DeepER [19] and DeepMatcher [41] have been introduced. Both systems use deep neural networks specifically developed for the entity matching task. The core idea of DeepER is to generate a vector representation for each of the compared tuples, projecting them from a symbolic into a high-dimensional embedding space. To this end, DeepER first uses word embeddings to translate the individual attribute values of the processed tuple into their corresponding vector representation. Subsequently, these attribute vectors are concatenated and form the final vector-based tuple representation. Using this representation, the network is trained so that similar tuples are drawn to each other, while unequal tuples repel each other. An interesting architectural characteristic of DeepER's network is that vector representations are established for both the entire tuple and each of its attributes. Generating the tuple representation has the advantage that the generated tuple embeddings can be used by downstream applications, such as clustering. Unfortunately, a major disadvantage of this approach is the use of word embeddings as the smallest unit for constructing the attribute and in turn tuple embeddings. This design decision makes the system susceptible to out-of-vocabulary (OOV) words, that is, words that are missing in the used word embeddings. By operating at the word level, it becomes significantly more difficult to adequately handle subword structures such as misspellings or common substrings.

In addition to conducting a design space exploration, DeepMatcher [41] extends the architecture of DeepER and adds various attribute summarization techniques. In contrast to DeepER, DeepMatcher operates by calculating similarity measures between the attribute pairs of the compared tuples. The resulting similarities are then aggregated into one similarity vector, which constitutes the basis of their classification. In their work, Mudgal et al. [41] show that the use of character embeddings can lead to significant performance improvements, especially when dealing with uncommon words or dirty data.

Despite its similar architecture, our proposed network differs in some aspects from the existing work. While our basic idea of calculating an embedded representation for the processed tuples is similar to that of DeepER, our approach differs in the processing of the individual attribute values. While DeepER uses word embeddings to compute the vector representation of each attribute, our architecture operates by splitting each attribute value into its characters, which are then mapped to a vector representation by using character embeddings. As discussed by Mudgal et al., the choice of word versus character-level embeddings is critical, due to the implications of OOV tokens [41]. The use of word-level embeddings reduces the mapping of a token to its vector representation to a lookup in an embedding matrix. If this lookup fails because the processed token is unknown, then a predefined "unknown vector" is usually used instead of a pre-trained token embedding. Because values occurring in tables can often not be matched to pre-trained word embeddings, such as Word2Vec [39] or GloVe [49], the use of word-level embeddings is not optimal for the deduplication task. To mitigate this issue, retrofitting techniques, such as those employed by DeepER, can be used to generate approximations of OOV words. This can be implemented, for example, by calculating an average embedding of co-occurring word embeddings. However, the calculation of such approximations causes the meaning of multiple word embeddings to be mixed, so that the meaning of the generated interpolation of an OOV word embedding is at best a diffuse representation of the missing word. By using the suggested character embeddings, the need for retrofitting techniques is eliminated, as any word can be created by a combination of individually learned character embeddings.

Another effort to reduce the OOV problem is made by the *fastText* approach introduced in Bojanowski et al. [7]. It complements any word embedding by additionally including all n-gram embeddings that can be generated from the processed word. When compared to the lookup procedure of word embeddings, this approach drastically reduces OOV words as it is able to represent each OOV word as the combination of its individual n-gram embeddings. However, as stated by the authors, *fastText* usually utilizes a range of 3- to 5-grams in its creation of word vector representations. Depending on the granularity of the used n-grams, some of them may not be known during the generation of the individual word vector representations, so that for each unknown n-gram a specific "unknown vector" is used instead.

With our model, we take an even more generic approach by training an embedding for each character in the ASCII table, further reducing the granularity from n-grams to character embeddings. By replacing n-grams with character embeddings, we further mitigate the OOV problem as we are able to construct any word embedding from its individual character embeddings. In addition to reducing the number of lookup errors, the size of the embedding matrix is also significantly reduced. Drawing on the findings of Mudgal et al. [41], the use of character embeddings allows us to overcome DeepER's limitations by being able to handle character-level issues, such as misspellings and common substrings. Looking, for example, at the duplicate pair ("`hibrid theory`," "`hybrid theory`"), which differs only in a single character, it is likely that a lookup of the word "`hibrid`" will return the "unknown vector," as it has never been encountered before.

We aim to learn an embedding for each individual attribute by combining the character embeddings into a vector representation of the underlying entity (much like the original DeepER). The compressed entity representation is then altered during the training process to the effect that similar entities are located closer together than dissimilar ones. Unlike the other systems, we are not trying to classify record pairs directly into duplicates and non-duplicates but concentrate on learning an entity representation that allows us to classify on the basis of simple distance measures, such as the Euclidean distance. As a consequence, this method could also be used as a distance measure in other applications, such as clustering.

With Termite, the authors of Reference [22] propose a system that allows the execution of queries across multiple structured and unstructured data sources. In a nutshell, the system transforms structured and unstructured data into a common embedding space where it can be used to retrieve related data points regardless of their origin and schema. Because the authors are mainly interested in identifying *related* entities across heterogeneous data sources, they address a problem different from ours. Although they also use an SNN architecture, they focus on learning a concept of relatedness between entities, which can be interpreted as a more relaxed case of deduplication, where it is sufficient for the entities to be similar, but do not necessarily correspond to the same real-world entities. Another difference to our work is that the authors consider both structured and unstructured data sources and do not explore the concept of transfer learning.

Unsupervised Approaches. In contrast to supervised methods, unsupervised techniques simplify the process of duplicate detection by not requiring labeled training data to achieve the desired classification. In this case, clustering algorithms, such as k-means or farthest-first clustering, were used to group together duplicate and non-duplicate entity pairs in clusters [20, 25, 28]. To this end, the required parameter k, which determines the number of clusters to be created, is often set to a value of two or three. A value of two clusters (k=2) corresponds to the categories “duplicate” and “non-duplicate,” while a value of three (k=3) considers an additional category with the name “possible duplicates.” Although cluster-based approaches can often be less accurate than their supervised counterparts [25], they are still useful for real-world applications, as sufficient training data is usually not available and has to be created with great effort.

Semi-Supervised Approaches. Algorithms that follow the principle of Active Learning form a subset of semi-supervised learning methods. The core of this principle is to enable the algorithm to query an external information source to resolve particularly challenging or hard-to-decide training cases. Sarawagi et al. designed a duplicate detection system that follows this principle [50]. The system works by having difficult-to-classify duplicate pairs evaluated by a user. The information gained is then fed back into the training process, creating a new and enhanced model. The iteration of this procedure allows the repeated improvement of the model until it is finally able to identify all entities as either duplicates or non-duplicates. A similar system was created by Tajada et al. [55]. It relies on decision trees to detect duplicate entity pairs.

Over the years, many crowd-based systems have been proposed, which also fall into the category of semi-supervised systems [1, 17, 23, 56]. One of these systems, CrowdER [57], pursues a hybrid approach to entity resolution seeking to narrow the gap between purely machine-based and purely human approaches. This is done through a two-step heuristic approach using a machine-based approach to reduce the search space and then presenting possible matches to a human audience for evaluation. Another recently introduced system is CloudMatcher [27], which was introduced as a fast, easy-to-use, scalable, and highly available service for entity matching.

Transfer Learning. In addition to the methods discussed so far, there are approaches in which transfer learning plays an essential role. In their approach, the authors of Reference [46] focus on reducing labeling costs through adaptive sharing of learned structures between scoring problems, involving more than two data sources. To this end, they define the problem of Multi-Source Similarity Learning, which explicitly focuses on entity resolution with more than two data sources. Furthermore, their approach is based on similarity vectors composed of traditional similarity metrics that are used to train an entity resolution that is confined to linear classification models. In contrast, we concentrate on one (*deduplication*) or two (*record linkage*) data sources, learn a similarity measure that is specifically tailored to the underlying dataset, and use a non-linear classification method by means of a neural network.

The deep learning-based approach of Kasai et al. [33] focuses on low resource consumption and, to this end, combines transfer learning with active learning components. Like earlier approaches, the authors turned to fastText embeddings, which contain subword information, to translate the processed tuples into a common embedding space, thus reducing the OOV problem. While this approach produces fewer OOV cases, our approach employs character embeddings, which are even more fine-grained, and almost completely avoid the OOV problem. Furthermore, the authors base their classification on similarity vectors and use a negative log-likelihood loss in conjunction with a softmax function, while our architecture is based on SNNs and therefore employs a contrastive loss function between vector representations of the processed entities. For knowledge transfer, they train all parameters of their network on the source dataset and use them to classify the entities of the target dataset. We follow the same approach for training, but transfer only the learned attribute representations and retrain the upper layers on the target dataset for fine-tuning.

Instead of an SNN architecture, the authors of Reference [59] suggest the use of a hierarchical neural network that combines character and word-level representations as well as attention mechanisms. While they use the same intuition, their approach differs in some aspects from ours. In particular, they extensively use type information from a knowledge base to pre-train the attribute-level EM and type matching models in an offline process. Thus, the use of a knowledge base becomes an essential part of their approach, which is not a prerequisite in ours. For training, they employ a log loss function that, in contrast to our contrastive loss, does not account for class imbalances in the training data.

The approach proposed in this article leverages two key concepts. The application of SNN to recognize similarities between various entities and the trend of automatic feature learning, which was pioneered by the deep learning community. Due to its design, this type of network is particularly suitable for learning distance measures between two entities, which is why we decided to exploit its features for duplicate detection.

3 PROPOSED APPROACH

Duplicate detection is a fundamental task in the field of data quality and data cleansing that has been extensively studied over the years [21]. Research has analyzed different methods that span all necessary steps [14]:

(1) Normalize data,
(2) Index and retrieve candidates of matching records,
(3) Compare the retrieved candidates,
(4) Classify them as matches or non-matches,
(5) Evaluate the entire process.

We contribute to Steps 3 and 4 by introducing a deep neural network capable of learning a similarity measure that is tailored to the idiosyncrasies of a particular dataset. Using this tailor-made, learned measure, we are able to classify entities with high accuracy as “duplicates” or “non-duplicates.” For Steps (1), (2), and (5), we employ standard techniques as described in Reference [14].

Similarity Measures. Because similarity measures are an integral part of the deduplication process, we briefly introduce the most important concepts. Traditionally, similarity measures are chosen w.r.t. the underlying attribute’s type [15]. For instance, if the field is of type Date, then the measure should consider a similarity in years to be more important than just in months or days. In our experiments, we treat all attributes as alphanumerics, as it is the most general type. Thus, here we consider only string similarity measures. These can be subdivided into (a) edit-based, (b) token-based, (c) hybrid, and (d) phonetic measures. Measures of type (a) are based on edit-operations, such as Levenshtein [37] or Jaro-Winkler [58], whereas measures of type (b) tokenize the strings and compare the sets of tokens—Dice [18] and Jaccard [31] being two examples. Type (c) measures are typically a combination of (a) and (b), with MongeElkan [40] serving as a good example. Last, Type (d) measures, such as Soundex [47], are ideal for words that sound similar as they transform syllables into characters that represent that sound.

In this section, we present the architecture, loss function, and training details of our approach. Section 3.1 introduces the general intuition and overall architecture of Siamese neural networks. Then, Section 3.2 gives a detailed description of the subnet architecture. The loss function and its mode of operation is presented in Section 3.3. Finally, we present the training procedure and its parameters in Section 3.4.

3.1 Siamese Neural Networks

When detecting duplicates, it is often the case that standard similarity measures are difficult to choose and cannot be sufficiently adapted to the complex characteristics of the underlying dataset. This difficulty becomes evident upon a closer look at the circumstances in which these measures are used. For example, a similarity measure that is designed to compare first names is clearly unsuitable for comparing timestamps but also other string-types, such as street-addresses, should be compared differently. Moreover, similarity measures are use-case driven. For instance, it strongly depends on the respective use case whether it makes more sense to compare two timestamps with regard to their syntactical or their chronological similarity. Even within the values of a particular domain, the chosen similarity measure may not be equally suitable for all encountered values. As such, an address column might contain both street and postbox addresses for which a case-based similarity measure might work best. Finally, it is often quite difficult, even for experts, to find a good weighting of the various attribute similarities to determine an overall similarity score of two input-records. It is, for example, not always clear whether first names are generally more important than city names when comparing customer records.

The SNN architecture was first introduced by Bromley et al. [8] to verify signatures on credit-cards. Since then, it has been used in many different areas, such as one-shot learning [35] as well as recognizing textual [45] and facial similarities [11]. As Siamese networks have already been shown to work well in areas where similarities between different entities need to be evaluated [38, 45], we chose this type of network to address the task of duplicate detection. Due to the twin architecture of these networks, they are particularly well suited to identify differences and similarities between the considered entities.

We intend to learn a similarity measure directly from the raw data, thus skipping the traditional feature engineering step required in other machine learning approaches. For this purpose, the network is trained by exposing it to both positive as well as negative example pairs. In our case, a training example consists of two tuples t_1 and t_2, and a corresponding binary label Y, which indicates whether the tuples refer to the same ($Y = 1$) or different real-world entities ($Y = 0$). In turn, a tuple consists of several attributes $a_1, \ldots, a_n$, which are individually preprocessed and passed to the subnets. The architectural details of the subnets are explained in more detail in Section 3.2.

In training the network, we aim to learn a function capable of mapping the input values into a lower dimensional embedding space, where a simple distance measure, such as the Euclidean distance, can be used to estimate the semantic similarity between the two tuples. The learned function should be able to position similar tuples close to each other within the embedding space, while different tuples should be positioned further apart. During training, we therefore use the positive examples to reduce the distance between known duplicates as much as possible, while using the negative examples to maximize the distance between known non-duplicates. The final function should yield a distance value that is very close to zero if two tuples are duplicates, and a value close to one otherwise.

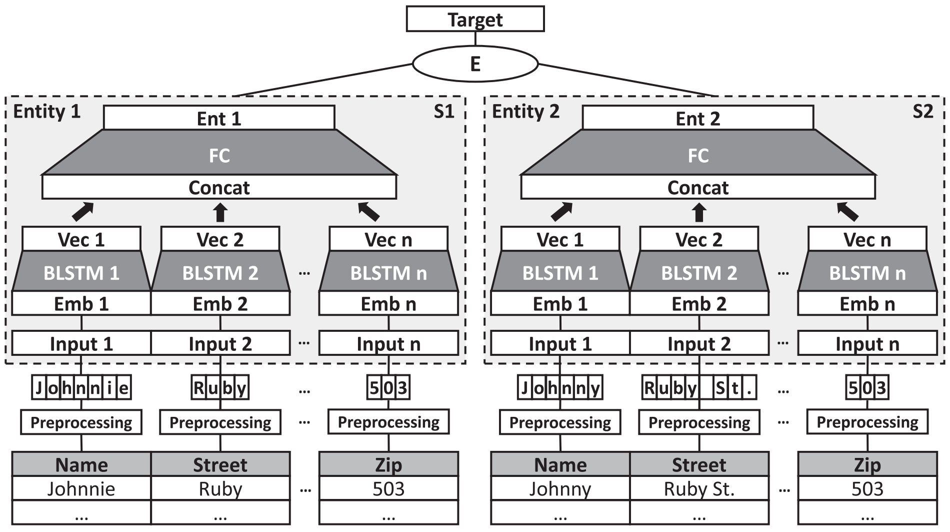

Fig. 1. Overview of the SNN architecture.

An SNN consists of two identical subnetworks S_1 and S_2, which are interconnected via a common energy function E. Since both subnets are identical in all aspects, this also means that they share a common weight or parameter matrix W. By sharing the model parameters between the subnets, both networks calculate the same function, thus making the learned similarity measure symmetric.

Figure 1 provides a conceptual overview of the network structure and illustrates all key components. As input, the network expects an entity pair in the form of two tuples and the corresponding label, which serves as the optimization target. More specifically, each tuple is provided to the network in an attribute-by-attribute fashion. Each of the two subnets thus processes one tuple of a given entity pair. Viewed from a conceptual level (see Figure 1) both input tuples pass through three network layers and eventually end up in the last layer as two reduced vectors of fixed dimensionality (Ent_1, Ent_2). The two resulting vectors serve as inputs to an energy function, which calculates a scalar value that can be interpreted as a similarity estimate. Next, we explain the structure of the two subnets (S_1, S_2) in detail.

3.2 Neural Network Model

This section contains a detailed description of the subnet architecture and an explanation of our design decisions. During the development phase, we experimented with different network architectures, exploring different layer depths, pooling layers, and attention mechanisms. In the following, we describe the architecture that achieved the best results.

As input, each subnetwork receives one of the tuples t_1 or t_2. These tuples consist of attribute values $\mathcal{A} = (t[a_1], \ldots, t[a_n])$ that belong to a specific domain $\mathcal{D} = (d_1, \ldots, d_n)$. Since our intention is to transfer knowledge between two networks, the tuples are fed to the network attribute by attribute. The reasons for this approach are explained in Section 4. Although a domain can be categorized by its data type, for example, d_1 can be of type integer or string, we are more interested in the semantic content of each domain. For instance, attributes belonging to a specific domain represent street names, while attributes from another domain represent movie titles. The core idea is to use neural networks to learn the characteristic data distribution of each domain, using all domain attributes as training data. Once the weight matrices of the individual domains have been formed as a result of the network training, they can be transferred to another network.

Assuming that movies are often based on books, it is plausible to use the weight matrix of a book title attribute from network N_1 to initialize the weight matrix of a movie title attribute in network N_2 to perform knowledge transfer.

Examining the attributes of a particular domain, we find that they are often very different in length. For example, the street name "A Street" is relatively short, while "Newport Pagnell Motorway Services Areajunction 14 15 M1" is rather long. Since the network design has to deal with these varying attribute lengths, this affects both the structure as well as the building blocks used to construct the network.

Although it is possible to use 1D Convolutional Neural Networks (CNNs) [26] to process sequential data streams of varying lengths $(x_1, \ldots, x_t)$, our architecture uses Recurrent Neural Networks (RNNs) [26] for the sequential processing of the attribute values. Unfortunately, simple RNNs suffer from the vanishing [3] or exploding [48] gradient problem and are therefore difficult to train. Due to this limitation, we decided to use Long Short-Term Memory (LSTM) networks [30], which are an extension of RNNs that mitigate these problems. We arrange two LSTM networks according to the Bidirectional RNN architecture of Schuster and Paliwal [51], forming a *bidirectional LSTM network* (BLSTM). A BLSTM network essentially consists of two LSTM networks that process the transferred character embeddings of a given attribute in opposite directions. The first network processes the input sequence from left to right, while the second processes the input sequence from right to left. Processing the input sequence from both sides allows us to consider context information from future and past states. The final results of both networks are then concatenated and forwarded to the subsequent network layer.

As shown in Figure 1, we use an embedding layer ($Emb_{1\ldots|\mathcal{A}|}$) as the first layer of each subnet. As discussed in Section 2, we feed each attribute value character by character into the network, building a character embedding for each character used within the attributes. We decided to process the attributes on a character basis, in the hope that this enables the network to learn how to cope with character level issues, such as transposed numbers, misspellings, or shared character sequences. Such effects are particularly prevalent in attribute domains with many name variants and ambiguities, such as last names, company names, product names, and brands. These embeddings are utilized both by the subsequent BLSTM layer and during our knowledge transfer experiments, where they are used to transfer the underlying character distribution of the attribute domains. We employ the BLSTM architecture described above as the next network layer. Since we use one BLSTM network per attribute, the structure of the network ultimately depends on the number of existing attributes in a tuple. This decomposition is what allows us to automatically learn a condensed representation of the underlying attribute domain for each attribute separately.

As discussed in more detail in Section 4, the knowledge accumulated in both the weight matrices of the attribute embeddings ($Emb_{1\ldots|\mathcal{A}|}$) and BLSTM layers ($BLSTM_{1\ldots|\mathcal{A}|}$) shall be transferred to another deduplication network, operating on a different dataset. Each BLSTM network generates a fixed-length vector ($Vec_{1\ldots|\mathcal{A}|}$), which represents a compressed version of the currently processed attribute. These vectors are then concatenated by a *Concat* layer, thus forming a compressed representation of the entire tuple. The resulting tuple representation is then passed through a fully connected (FC) layer, enabling the network to learn relationships between the individual attributes of the constructed tuple.The resulting tuple embeddings (Ent_1, Ent_2) are then passed to the energy function whose results are used to calculate the gradients for the backpropagation step.

The noise that occurs in entities of structured data sources is often caused by typos or spelling variations in their attribute values. By leveraging character embeddings as well as end-to-end network training, we are able to learn that even though two entities differ in some characters, they still represent the same entity. This is achieved by adjusting the weights of the network during backpropagation so that the FC layer causes different weights to be assigned to character embeddings of common characters than to those that differ. In this way, the FC layer generates an entity representation that reduces or increases the distance of duplicate or non-duplicate entity pairs in the latent space despite the presence of noise.

The individual dimensions of our model are as follows: The dimensionality of the character embeddings is 32. Our LSTM layer has 128 dimensions, and the FC layer has 64 dimensions. The details of our loss function are discussed in the next section.

3.3 Loss Function

The general intuition of our system is to train a neural network $N_W(X)$ that is able to project the given tuples into a low-dimensional embedding space. In this space, duplicate and non-duplicate tuple pairs can be distinguished by using simple distance measures, such as the Euclidean distance or other more complex measures [10]. Learning in this context means adapting the weights of N_W so that the calculated loss is minimized. The goal of the mapping is to place similar tuples close to each other in the embedding space while dissimilar tuples are placed further apart. To learn a function with these properties, we use the contrastive, energy-based loss function presented in Hadsell et al. [29]. As described in the last section, both tuples are passed through the network yielding one embedding vector (Ent_1, Ent_2) for each tuple. If we choose the Euclidean distance as energy function $E_W(Ent_1, Ent_2) = \|Ent_1 - Ent_2\|$, then the general definition of the loss function is

$$\mathcal{L}(E_W, Y) = YL_D(E_W(Ent_1, Ent_2)) + (1 - Y)L_{ND}(E_W(Ent_1, Ent_2)). \tag{1}$$

The dependence of the energy function on the parameter values of the network is expressed by the W in the definition. At its core, the loss function consists of the terms $L_D(E_W)$ and $L_{ND}(E_W)$, which ensure that similar inputs receive a low energy value whereas unequal inputs receive a higher one. In their work, Hadsell et al. motivate the intuition behind these terms with the behavior of mechanical springs, where L_D brings similar embedding vectors closer together and L_{ND} repels different vectors [29]. If we define the two terms as follows:

$$L_D(E_W) = \frac{1}{2}(E_W)^2, \tag{2}$$

$$L_{ND}(E_W) = \begin{cases} \frac{1}{2}(m - E_W)^2 & \text{if } E_W < m, \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

then the result is the following loss function:

$$\mathcal{L}(E_W, Y) = Y\frac{1}{2}(E_W)^2 + (1 - Y)\left(\frac{1}{2}\max(0, m - E_W)\right)^2. \tag{4}$$

The value m in this definition is a margin value, which controls that only tuple pairs within the specified margin contribute to the loss function. In addition, Hadsell et al. emphasize the meaning of the L_{ND} term, which makes it possible to take dissimilar tuples as well as similar tuples into account when minimizing E_W, thus enabling a more optimized solution.

Duplicate detection often involves highly skewed datasets containing many more non-duplicates than duplicates. We introduce the weights W_D and W_{ND}, which allow us to weigh the loss terms L_D and L_{ND} differently. In this way, we can control how strong duplicates or non-duplicates contribute to the calculated loss. Extending Equation (4) with the weight parameters W_D and W_{ND} yields our final loss function:

$$\mathcal{L}(E_W, W_D, W_{ND}, Y) = W_D Y\frac{1}{2}(E_W)^2 + W_{ND}(1 - Y)\left(\frac{1}{2}\max(0, m - E_W)\right)^2. \tag{5}$$

We used the duplicate to non-duplicate ratio of the respective training set to determine appropriate weights (W_D, W_{ND}) for the loss function. Because we generated our datasets with a duplicate, non-duplicate ratio of 1 to 10 (described in Section 5.3), we adopt these settings also for the weight parameters W_D and W_{ND}. In cases where it is difficult to estimate the parameter values for W_D and W_{ND}, they can be determined by means of hyperparameter optimization [4, 52].

3.4 Training Details

We use backpropagation to determine the gradient of the loss function with respect to the weights W. We update the weights by employing Adaptive Moment Estimation (Adam) [34], which is capable of computing adaptive learning rates for each parameter. We kept the default settings for the optimizer at $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 10^{-8}$. Due to the weight sharing between S_1 and S_2, the gradients of both networks behave additively, so that we use the summed gradient contributions of both subnets while updating the weights. For training, we use batch sizes of 16 and 32. The margin on the loss function was set to a value of 1. To initialize the weights of each layer, we use the Xavier initialization scheme as presented in Reference [24]. To counteract overfitting of the network, we employ dropout regularization for each time step of the LSTM by setting the dropout value to 0.1 [53]. We trained the network until convergence, which in our case required between 10–15 epochs. For the implementation of the model, we used the Keras Framework using Tensorflow as its backend. All models were trained using an Nvidia Titan X Pascal GPU.

Training a model took between 30 min and 24 h, depending on the dataset size. For all datasets that were categorized as *small* in Table 1, we performed a 10-fold cross-validation. We initially divided the entire dataset into a training and a test set at a ratio of 70 to 30 percent, using the training set to perform a 10-fold cross-validation. For each fold, we selected the model from the epoch with the best F-measure on the cross-validation's test data to make a prediction on the previously separated test set. We use these predictions to report on the average precision, recall, and F-measure over all 10 folds.

Since the datasets categorized as *large* in Table 1 were too large for a full cross-validation, we employed a different evaluation scheme. We divided the corresponding datasets according to a ratio of 3:1:1 into training, validation, and test sets. We then trained on the training set and used the validation set to select the model from the epoch with the best F-measure, which we used to make a prediction on the test set. In this scenario, we report the best precision, recall, and F-measure on the test set. For the entire model, we used rectified linear units (RELUs) as our activation function.

4 KNOWLEDGE TRANSFER

To reduce the amount of data needed to train neural networks, it is our goal to transfer already learned knowledge between networks. As pointed out by Jialin Pan and Yang [32], transfer learning addresses this by allowing domains, tasks, and distributions used in training and test to be different. According to their proposed categorization, we identify our case of knowledge transfer as "Transductive Transfer Learning." While for us the task of duplicate detection always remains unchanged, we change the underlying domain by switching from detecting duplicates on one dataset (e.g., Movies) to detecting duplicates on a different dataset (e.g., CDs). Thus, knowledge transfer makes it possible to train models for which this would not have been possible due to insufficient training data.

As in other areas, the available datasets for duplicate detection differ greatly in terms of labeled training data. On the one hand, there are datasets that contain a large number of already labeled duplicate pairs; on the other hand, there are datasets that contain almost no labeled pairs. Our central idea is therefore to exploit this imbalance by training a neural network on a dataset with a large number of duplicate pairs and transferring the acquired knowledge to the classification of duplicates, where the dataset contains significantly fewer training examples.

Because we learn to deduplicate entities, which are represented as tuples, the tuple attributes and their domains are of vital importance. As described in Section 3.2 a crucial step in our deduplication approach is to learn the characteristic data distribution for each attribute domain individually. This is achieved by the network design shown in Figure 1. Here, the critical part is making use of individual embedding and BLSTM layers for each attribute. By deliberately separating the individual attributes, the domain knowledge gained during training accumulates in the weight matrices of the dedicated embedding and BLSTM layers. This effect allows us to transfer the domain knowledge for specific attributes individually. For example, we can choose to transfer only the knowledge of specifically selected attribute domains, which can give us an advantage in the target domain. Formally, the knowledge transfer can be defined as follows:

> **Knowledge Transfer & Domain Compatibility:** Given two neural networks—a source network N_1 and a destination network N_2—as well as their associated training data T_1 and T_2. We define knowledge transfer as the transfer of selected weight matrices from N_1 to N_2. To this end, we train N_1 using T_1 and transfer the gained knowledge to N_2 before training it on T_2. In addition, we define that two attribute domains—a source attribute domain D_S and a destination attribute domain D_T—are *domain-compatible* to each other, if they are either equal ($D_S = D_T$) (e.g., both street names) or structurally similar ($D_S \approx D_T$) to each other (e.g., book title and movie title).

Although we assign compatible attribute domains manually in our experiments, similarity search techniques such as LSH Ensemble or BML can be used to find compatible attribute domains suitable for knowledge transfer [44, 60]. Systems such as DeepAlignment [36] can also be used to automatically determine a corresponding domain mapping. This aspect is outside the scope of this article.

Technically, we perform a knowledge transfer by using both the weight matrices of the embedding and the BLSTM layers in N1 to initialize compatible attribute domains in N2. As a result, we are able to train N_2 even if T_2 contains much fewer training examples than T_1 ($|T_2| \ll |T_1|$). This method is often referred to as "fine tuning" the network. Note that the knowledge accumulating in the weight matrices of the FC layer (W) during the training process is specific to the entities being compared (e.g., books or movies), not the attributes (e.g., name, zipcode). For this reason, the weights of the FC layer are not easily transferable for deduplication of other entities, such as cars or hotels. While the the learned attribute representations can be reused for the deduplication of other entities, the knowledge of how to combine the individual attribute representations into a vector-based representation of the processed entities must be relearned for the specific entities being deduplicated. To transfer the maximum available knowledge, this process can be performed for multiple attribute domains coming from different source networks. For example, it is possible to transfer weights from several different source networks to corresponding attributes of a target network. For instance, assuming two previously trained source networks N_1 and N_2, it is possible to transfer a weight matrix for *art titles* from N_1 and a weight matrix for a *location* domain from N_2 to a target network N_3. The elegance of attribute-based knowledge transfer lies in the fact that over time a steadily growing repository of weight matrices for a wide range of attribute-domains accumulates. In case only some attributes are domain-compatible during knowledge transfer, the remaining attribute values could be initialized with weights from the repository. In addition, one could try to learn the weight matrices for a particular domain completely unsupervised through the use of encoder-decoder architectures, which would allow the repository to be easily extended. By following this protocol, it is now possible to combine knowledge from multiple different source networks to initialize as many attributes of the target network as possible with pre-trained weights. Finally, it should be noted, that the transferred weight matrices can be used to initialize multiple attribute domains of the target network, if a domain meets the above mentioned requirements.

5 DATA AND GOLD-STANDARD

In this section, we give a brief overview of the used datasets (Section 5.1), their preprocessing (Section 5.2), and describe our process for creating non-duplicate pairs (Section 5.3).

5.1 Datasets

Identifying and evaluating solutions to problems existing in the real world requires the use of real-world datasets. Since the proposed method can be used to find duplicates both within the same dataset (duplicate detection) and between different datasets (record linkage), we test it on datasets for both tasks. The main difference between the two scenarios is that duplicate detection aims to find all duplicates within a single dataset, whereas record linkage seeks to uncover duplicates across multiple datasets. Given two relations $\mathcal{A}$ and $\mathcal{B}$ of sizes n and m, respectively, the process of duplicate detection produces candidate pairs in $O((n+m)^2)$, whereas in a record linkage scenario, the number of pairs that need to be evaluated are in $O(n \cdot m)$. Therefore, the two scenarios differ in that deduplication creates more non-duplicate candidate pairs, which can result in more duplicates being misclassified, which in turn can negatively impact classification results. Although the record linkage scenario can also be mapped to a deduplication scenario, we have deliberately refrained from doing so to reflect the evaluation setup of the competitor approaches.

For the task of duplicate detection, we compare our approach with that of Christen [13], which is essentially based on the classification of feature vectors created on the basis of the compared entities. To this end, they use an SVM classifier with various hyperparameters. We use the same datasets as Christen, which can be found on our website.[1] The details of the individual datasets can be found in Table 1 and in the brief descriptions below.

Table 1. Datasets Used for Our Experiments

Dataset	**#dpl**	**#ndpl**	**#records**	**class**
Deduplication				
Restaurants	112	1,120	864	small
Census	376	3,760	841	small
CD	300	3,000	9,763	small
Cora	64,578	179,125	1,879	large
Movies	14,190	141,900	39,180	large
Record linkage				
BeerAdvo-RateBeer	68	382	544	small
iTunes-AMA	132	407	933	small
DBLP-Scholar	5,473	54,730	66,879	large
DBLP-ACM	2,224	22,240	4,910	large
WMT-AMA	1,157	11,570	24,628	large
AMA-GOOG	1,300	13,000	4,589	large

- **CD**: Entries of audio CDs with descriptive attributes, such as artist, title, tracks, genre, and year.

[1] https://hpi.de/naumann/projects/repeatability/duplicate-detection/knowledge-transfer-for-duplicate-detection.html.

- **Census:** Based on real data and generated by the U.S. Census Bureau, it contains a single attribute with a record value, called "text." This dataset contains two relations "A" and "B," and could also be used for record linkage, i.e., linking the two relations. Like Christen, we treat it as a typical single-relation dataset and try to find duplicates instead of linking matches.
- **Cora:** Bibliographic records of machine learning publications, it includes reference information, such as authors, title, and year.
- **Movies:** Result of merging two different datasets, such that real-world duplicates are available. The information provided is limited to actors and movie titles.
- **Restaurants:** Mixed dataset of two other relations, based on the Fodor's and Zagat's restaurant guides. This corresponds to the Fodor-Zagat dataset in Reference [41].

For the record linkage task, we compare our approach with the reported performance metrics of the DeepMatcher system, and therefore use the same datasets as Mudgal et al. [41] (see Table 1). Though DeepMatcher can be configured with a number of different architectural variants, their comparison shows that, apart from the hybrid variant, the RNN variant is superior to the other architectural variants in terms of performance. When comparing the RNN with the hybrid variant, it turns out that the former is on average only 1.5% F-measure points worse, which does not constitute a clear superiority of the hybrid variant. For this reason, and because the RNN architecture of DeepMatcher comes closest to our approach, we focused on comparing our approach to the RNN variant of DeepMatcher. We do not compare our approach on their *Home*, *Electronics*, and *Tools* datasets, as those are not publicly available. All other datasets can be obtained from the Magellan Data Repository [16], which contains not only statistics about each dataset but also a detailed description of how they were created.

Except for the BeerAdvo-RateBeer and iTunes-AMA datasets, which already contained non-duplicate pairs, we had to generate them for all other datasets according to the process outlined in Section 5.3. Statistics about the number of duplicate (*#dpl*) and non-duplicate (*#ndpl*) pairs, the total number of records (*#records*) as well as a categorization into small and large datasets (*class*) can be found in Table 1. We categorized datasets that contain fewer than 4,500 labeled tuple pairs (*#dpl* + *#ndpl*) as small datasets and those with more than 4,500 labeled tuple pairs as large datasets.

5.2 Data Preprocessing

We kept preprocessing to a minimum to not alter the input data too drastically from its original raw state. As such, we first lower-cased all input characters, which not only helped the network to generalize better but also reduced the size of the embedding layers.

We then normalized any Unicode characters so that, e.g., accents and umlauts are translated into their respective ASCII characters, using the *normalize* function of the Python *unicodedata* package. For example, the German ö was replaced by the combination oe, whereas other characters, such as é or ç, were simply replaced with the letters e and c. This normalization further reduced the size of the embedding layers.

In a final step, we translated each ASCII character into its corresponding ASCII numeric representation, which served as input for the network. To speed up network training, we truncated the values of the description attribute of the AMA-GOOG dataset after 1,000 characters.

5.3 Selecting Duplicate and Non-duplicate Pairs

The gold standards that are available for most of the datasets in Section 5.1 each contain a list of record pairs that uniquely identify *duplicate record pairs*. In case this set is not transitively closed, we additionally create all transitive pairs and call this extended set *DPL*. For most datasets, the number of existing duplicates represents only a small subset of all possible tuples. To train and test a classifier, we also require a number of negative examples, i.e., non-duplicate pairs. Unfortunately, a random pairing of tuples most likely results in non-duplicate tuple pairs that exhibit a very low similarity in their attribute values, which in turn makes them easily distinguishable from true duplicates. This does not reflect the challenges of a real-world matching process, hence it is necessary to create non-duplicate pairs of greater similarity, making it harder for the model to classify them. Since training and testing with hard to distinguish non-duplicates comes much closer to a realistic matching scenario, we rely on blocking techniques [14, 21] to generate non-duplicate pairs that are harder to distinguish from real duplicates. In practice, to speed up duplicate detection, blocking methods are typically used to divide datasets into disjoint subsets, called blocks, according to a predefined partitioning key. To avoid false negatives, usually multiple partition keys are defined. It is important to select the partition key with great care, as it controls not only the size and number of blocks but also how similar the individual data records within each block are.

We perform a simple blocking by using each attribute of the currently processed dataset as a partitioning key. In case an attribute is multi-valued, we first divide it into its individual values and use those for blocking. Very unique attributes, which would lead to many single-record blocks, are split into n-grams of 6 to 10 characters, depending on the length of the attribute, and then used as partition keys. As a result, we obtain several blocks, within which we create the cross-product of all contained data records, and thus form data record pairs that are then combined into a common dataset. After making sure that there are no known duplicate pairs in the resulting dataset, we refer to it as *NDPL*.

The NDPL dataset is, therefore, the set of all record pairs that can be formed within all blocks and does not include duplicate pairs. To achieve a reasonably balanced training set, we ensured that the ratio of DPL to NDPL is 1:10 by randomly selecting pairs from the NDPL dataset. An exception to this is the Cora dataset, where we were unable to maintain the ratio of 1:10 without overly relaxing the blocking strategy.

6 EXPERIMENTS

This section presents the results of our experiments. In Section 6.1, we discuss the results achieved by training the SNN from scratch, while in Section 6.2, we present how transfer learning helped to further improve the results. For details about the training process and its parameters, please see Section 3.4.

6.1 Learning from Scratch

As a first experiment, we train the network on each of the datasets listed in Table 1 and report the achieved precision, recall, and F-measure values. To see how our approach performs against deduplication systems that use manual feature engineering, we compare ourselves to the SVM-based system of Christen [13]. As pointed out by Ebraheem et al., DeepMatcher is an extension of DeepER [19], hence in the record linkage case we compare ourselves with DeepMatcher.

6.1.1 Duplicate Detection. As seen in Table 2, we surpass the SVM-based approach in all cases, sometimes even dramatically as in the case of the Cora dataset where we reach an F-measure of 99.24 percent. On average, we manage to exceed the SVM approach by +5.9 and +23.7 percentage points in precision and recall, corresponding to an average improvement of +15.8 percentage points in F-measure. We attribute the increase in performance to the possibility of training the network in an end-to-end fashion. For example, the character embeddings are trained together with the actual task, and, in contrast to manually created features, can be adapted according to the propagated

Table 2. Performance Comparison for Deduplication Datasets

Dataset	**SNNDedupe**			**SVM** [13]		
	P	**R**	**F_1**	**P**	**R**	**F_1**
Restaurants	96.5	100	**98.2**	97.3	75.8	84.1
Census	88.7	94.6	**91.5**	87.5	75.1	80.8
Cora	99.3	99.2	**99.2**	82.2	71.8	76.4
CD	91.8	84.0	87.4	–	–	–
Movies	93.1	85.7	89.2	–	–	–

Table 3. Performance Comparison for Record Linkage Datasets

Dataset	**SNNDedupe**			**DeepMatcher** [41]		
	P	**R**	**F_1**	**P**	**R**	**F_1**
BeerAdvo-RateBeer	74.9	74.3	**73.1**	59.1	92.9	72.2
iTunes-AMA	93.8	93.0	**92.9**	92.0	85.2	88.5
DBLP-Scholar	94.8	91.9	**93.3**	93.2	92.7	93.0
DBLP-ACM	98.4	98.0	98.2	97.1	99.5	**98.3**
WMT-AMA	61.6	60.8	61.2	70.9	64.6	**67.6**
AMA-GOOG	90.6	82.1	**86.1**	69.5	52.6	59.9

error. In this way, the entire network is tuned to the task at hand, which is difficult to accomplish with an SVM approach that works with a number of predefined features.

An interesting observation during evaluation was the analysis of the causes for misclassification: We found that when the network generated classification errors, they were often accompanied by missing data in at least one of the entity's attributes. This is a well-known problem in the area of duplicate detection, and we noticed that it negatively affects the performance of the neural network. In short, if we decide to replace an empty attribute field with a specific value, then we change the entities so that they become either more similar or dissimilar. In the worst case, this process makes the correct classification considerably more difficult. To avoid this problem, we decided to exclude attributes with missing values from the entity representation that is passed to the network. Technically, we achieve this by replacing both attribute values in their respective entities with a fixed value, e.g., zero, and use a corresponding masking layer within the network to make these values invisible to the network.

6.1.2 *Record Linkage.*

In the case of record linkage, Table 3 shows that we were able to outperform our competitor on four out of six datasets. The largest improvement was achieved on the AMA-GOOG dataset, where we measured an improvement of +26.2 percentage points in F-score over DeepMatcher. We suspect this large performance increase to be caused by the truncation of the description attribute during the preprocessing phase, making it easier to learn a better representation of the description attribute. In most of the other cases, the performance increase is not as significant as with the deduplication task; however, we were also able to measure an improvement of +4.4 percentage points for the iTunes-AMA dataset.

In the case of the WMT-AMA dataset, however, our approach is 6.4 percentage points worse than DeepMatcher. As mentioned by Mudgal et al., it can sometimes occur that due to inaccurate extraction methods, some attribute values are not assigned to the intended columns, but to the

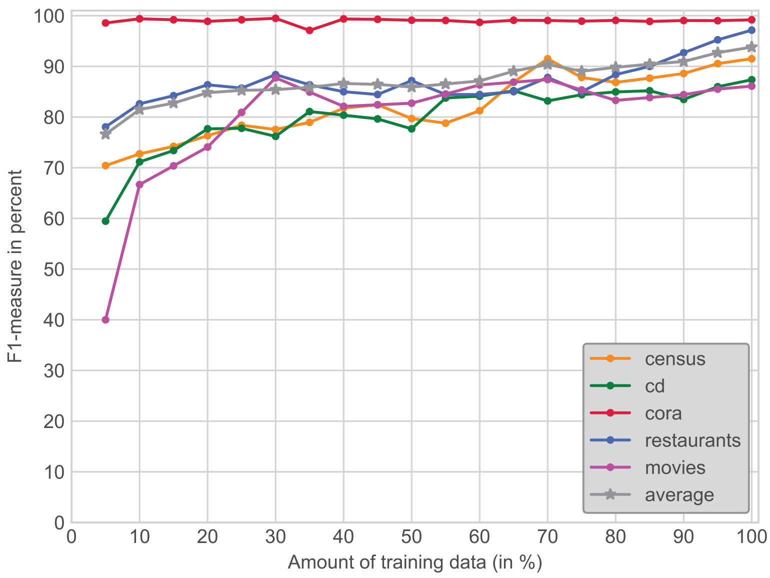

Fig. 2. Performance of SNNDedupe on deduplication datasets for increasing amount of training data (of a 3:1:1 split).

columns of other attributes [41]. Because the WMT-AMA records contain such impurities, our network architecture struggles to find a good representation for dirty attribute columns, which is reflected in weaker performance. To address this issue, the deduplication could be performed on a representation of the entire entity, created by concatenating the individual attribute values. Using this approach, the network would be able to determine the similarity of two entities in their entirety, eliminating the issue of incorrect attribute assignments, because entities are no longer deduplicated on attribute but on entity level. Unfortunately, leaving the attribute level would prohibit the formation of attribute embeddings during the deduplication process, which in turn would render the envisioned attribute-level knowledge transfer impossible. We have therefore decided to not proceed accordingly.

Our results are on par with those of DeepMatcher, justifying our design decisions, but in addition allowing us to perform knowledge transfer. In Section 6.2, we examine the effects of transfer learning on the networks performance.

6.1.3 Learning with Increasing Training Data. Due to the large number of record pairs that can arise during duplicate detection, a complete annotation of the entire dataset is usually very cost-intensive. This raises the question of how SNNDedupe behaves for different amounts of training data. To answer this question, we train SNNDedupe with an increasing amount of training data, starting at 5% of the corresponding dataset, and gradually increasing it by 5% increments until we arrive at utilizing the entire amount of training data. We do this for both the duplicate detection as well as the record linkage datasets, with results shown in Figures 2 and 3. As both figures show, the performance of SNNDedupe depends on the complexity of the dataset. While with relatively simple datasets, such as Cora or DBLP-ACM, an F-measure of 98.6% and 94.89% can be achieved with only 5–10% of the total training data, it is more difficult to achieve a satisfactory F-measure with more demanding datasets, such as WMT-AMA, where we obtain an F-measure of 61.2% even after utilizing all training data.

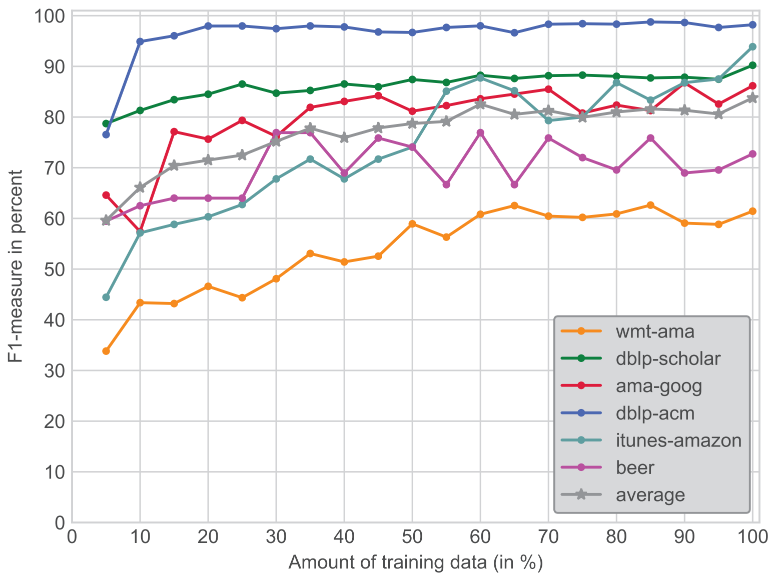

Fig. 3. Performance of SNNDedupe on record linkage datasets for increasing amount of training data (of a 3:1:1 split).

Looking at the average performance, it can be seen that SNNDedupe is capable of achieving F-measure values between 77–81% for deduplication and 59–66% in case of record linkage using just 5–10% of training data. In addition, we observe that for every additional 5% of training data, the performance increase is significantly stronger in the beginning than later on. Thus, the average increase in F-Measure for adding 5% additional training data within the first 25% of the dataset amounts to 3.23% for record linkage and 2.16% for deduplication. In contrast, for every 5% increase in training data within the remaining 75% of the dataset, the average increase in F-measure measures only 0.56% (deduplication) and 0.75% (record linkage). This indicates that SNNDedupe is able to provide a good classification performance after using up to 25% of all available training data. In practice, however, this depends on the individual case as well as on the complexity of the underlying dataset, which can be seen in the case of iTunes-AMA, where a significant performance increase is also observed later during the transition from 50 to 55% of training data. Another interesting finding is that the amount of training data used also influences the stability of predictions. As such, the prediction behavior is relatively stable for datasets with many training examples, whereas in the case of the Beer dataset, which consists of only 450 training examples, strong fluctuations in the predictions of SNNDedupe can be observed. In general, SNNDedupe is able to achieve average F-values of 72.49% (record linkage) and 85.26% (deduplication) by using up to 25% of the respective training data.

It should be noted that for this experiment, the increase in training data is achieved by adding randomly selected training examples. The selection of informative training examples is at the core of active learning approaches and will be the subject of future research.

6.2 Transfer Learning

We examine the effects of transferring selected weight matrices, and thus the knowledge they contain, from one network to another. Our goal is to explore how knowledge transfer affects the training process and thus the performance of the network.

Table 4. Results of Transferring Different Weight Matrices from One Network to Another

Transfer of Embeddings	**P**	**R**	**F_1**
Movies.actors ⟶ CD.artists	87.1	94.5	90.4
Movies.title ⟶ CD.album	(−4.7)	(+10.5)	(+3.0)
WMT-AMA.title ⟶ AMA-GOOG.title	90.2	86.1	88.1
WMT-AMA.techdetails ⟶ AMA-GOOG.description	(−0.4)	(+4.0)	(+2.0)
WMT-AMA.brand ⟶ AMA-GOOG.manufacturer			
Transfer of BLSTMs			
Movies.actors ⟶ CD.artists	88.3	87.5	87.6
Movies.title ⟶ CD.album	(−3.5)	(+3.5)	(+0.2)
WMT-AMA.title ⟶ AMA-GOOG.title	89.8	84.1	86.8
WMT-AMA.techdetails ⟶ AMA-GOOG.description	(−0.8)	(+2.0)	(+0.7)
WMT-AMA.brand ⟶ AMA-GOOG.manufacturer			
Transfer of Embeddings & BLSTMs			
Movies.actors ⟶ CD.artists	91.9	92.5	92.1
Movies.title ⟶ CD.album	(+0.1)	(+8.5)	**(+4.7)**
WMT-AMA.title ⟶ AMA-GOOG.title	91.7	89.8	90.7
WMT-AMA.techdetails ⟶ AMA-GOOG.description	(+1.1)	(+7.7)	**(+4.6)**
WMT-AMA.brand ⟶ AMA-GOOG.manufacturer			

6.2.1 Experimental Setup. As explained in Section 3, the network architecture is structured so that each attribute is processed by a dedicated subnetwork. This allows us to carry out even partial knowledge transfers by transferring only the weight matrices for specific attributes. The general experimental setup can be described as a two-step process. First, we take a neural network (N_1) that we previously trained on a source dataset and transfer the knowledge contained in the weight matrices for specific attributes, to another, untrained network (N_2). After the weight transfer is complete, we train the network N_2, which already contains the knowledge of N_1 for the transferred attributes, on a destination dataset. During our experiments, we transfer knowledge between two network pairs, which each operate on one dataset pair. The first transfer takes place from a network that has been trained on the Movies dataset to a network that shall be trained on the CD dataset.

As a second transfer, we transmit knowledge between a network trained on the WMT-AMA dataset and a network that we then train on the AMA-GOOG dataset. As discussed in Section 4, the general assumption made for these transfers is that the data distribution of the transferred source attribute resembles the data distribution of the destination attribute and is therefore well suited for the knowledge transfer. With this in mind, we decided for the first transfer to map the weight matrices of the *actor* and *title* attributes of the Movie dataset to the *artist* and *title* attributes of the CD dataset, respectively. This concrete transfer is reflected in the notation used in Table 4: $dataset_{src}.attribute_{src} \longrightarrow dataset_{dst}.attribute_{dst}$ describes from which source attribute ($attribute_{src}$) of the source dataset ($dataset_{src}$) to which destination attribute ($attribute_{dst}$) of the destination dataset ($dataset_{dst}$) the knowledge transfer is performed. Although we perform the mapping manually, schema matching research has produced a number of techniques to automate this step both heuristically and using machine learning approaches [2, 36]. The exploration of these techniques lies outside the scope of this work.

6.2.2 What to Transfer? The next step is to determine which of the weights are transferred from one network to the other. As can be seen in Figure 1, each of the subnetworks consists of two layers, an embedding layer ($Emb_{1...|\mathcal{A}|}$) and a BLSTM layer ($BLSTM_{1...|\mathcal{A}|}$). For our experiment, this results in three possible transfer configurations, which are listed in Table 4. First, we transfer only the weight matrices of the embedding layers, in a second step only the matrices of the BLSTM layers, and finally, both matrices of the embedding as well as those of the BLSTM layers are transferred. We structure our experiments in this way, because it allows us to study how each weight transfer affects the overall result. Once the weight transfer is complete, we train the network on the destination dataset as described in the previous section. Table 4 reports on precision, recall, F-measure, and relative improvement over the baseline, which was trained from scratch in the previous section. The baseline values can thus be found in Tables 2 and 3. The relative improvements over the baseline are shown in parentheses below their corresponding precision, recall, and F-measure values.

Transfer Embeddings. Regarding the first experiment, in which only the embeddings are transferred, it is noticeable that an improvement of +3.0 and +2.0 percent compared to the respective baseline can be observed for both datasets. Upon closer inspection, it can be seen that this improvement is mainly driven by an improvement in recall (+10.5/+4.0). These results support our initial assumption that the network, due to its high flexibility, is actually capable of learning a similarity measure that correctly identifies more duplicate pairs. It also shows that the knowledge about certain attribute domains contained within the embeddings can be transferred to other attributes that possess similar domain properties. Attention should also be paid to the circumstance that in both cases the recall values increase significantly without a severe decrease in precision. This interaction ultimately leads to the aforementioned improvement of F-measure.

Transfer BLSTM weights. In a second experiment, we transfer only the weight matrices of the BLSTM layers. Compared to the transfer of the embeddings, there is only a relatively small improvement of +3.5 and +2.0 percent in the recall values, while the precision values decrease in both cases. On careful consideration, this was to be expected as the BLSTM layers were trained together with the embedding layers and therefore the values of their weight matrices are conditioned on the specific embeddings that were jointly trained. If the embeddings are not transferred, but initialized randomly, then the weight matrices of the BLSTM layer also lose a lot of their significance. Ultimately, this causes the network to perform hardly better than the baseline with +0.2 and +0.7 percent improvement in F-measure.

Transfer Embeddings and BLSTM weights. In the last transfer experiment, we measure the performance of the network after transferring both the weight matrices of the embedding and the BLSTM layers. With an improvement of +8.5 and +7.7 percent, the recall values show the greatest improvement in this experiment. In contrast to the experiment in which only the embeddings were transferred, however, an improvement in the precision values of +0.1 and +1.1 percent can also be observed. Even though the improvements in the precision values are small, in combination with the recall improvements, they ensure that the F-measure values improved by +4.7 and +4.6 percent compared to the baseline. These results lead us to conclude that the best results can be achieved when the weight matrices of both the embeddings and the BLSTM layer are transferred jointly.

Overall, the transfer enabled us to improve the total network performance by +4.7 percent from 87.4 to 92.6 percent F-measure for the CD dataset and by +4.6 percent F-measure from 86.1 to 90.7 percent for the AMA-GOOG dataset.

6.2.3 Data Reduction Through Knowledge Transfer. In a final experiment, we investigate to what extent transfer learning can be used to reduce training data. To this end, we trained the network on an increasing number of tuple pairs from the AMA-GOOG dataset and recorded the F-scores achieved with and without knowledge transfer. The knowledge transfer was carried out in the same way as in the previous transfer experiment (WMT-AMA ⟶ AMA-GOOG). We transferred

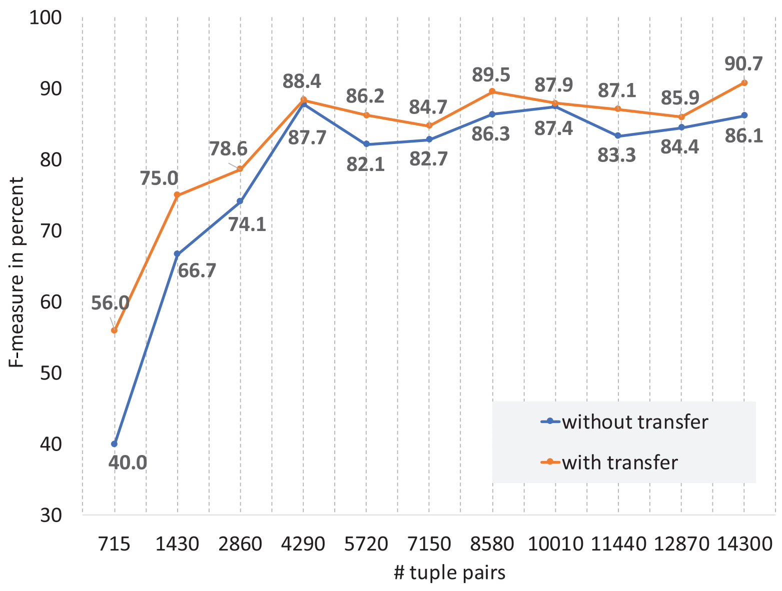

Fig. 4. Development of the F-measure with and without knowledge transfer.

both the weights of the embedding layers as well as those of the BLSTM layers. Figure 4 shows that knowledge transfer significantly improves performance when training with a smaller number of tuple pairs. This performance increase can be clearly observed up to a number of 4,290 tuple pairs and is less prominent afterwards. Nonetheless, it should be noted that the transfer values always remain above the values measured without transfer. Considering the graph, it can be observed that an F-score of 75.0 percent can be achieved when training with 1,430 tuple pairs, if a knowledge transfer has been carried out beforehand. To achieve the same F-score without knowledge transfer, one would have to use more than twice as many tuple pairs for training: the required amount of training data can be significantly reduced through a well-executed knowledge transfer.

7 CONCLUSION AND FUTURE WORK

We introduced a deep Siamese neural network capable of learning a similarity measure between tuple pairs of specific datasets that can then be used to detect duplicates. Thereby, we eliminate the manual feature engineering process and thus significantly reduce the effort required for model building. We compare our approach with two competitors, a more traditional, SVM-based duplicate detection approach and DeepMatcher, a neural network for entity linking. In duplicate detection, we were able to outperform our competitor in all cases with performance improvements of up to 22 percent. For entity-linking, we managed to outperform the DeepMatcher system on four out of six datasets and achieved improvements of up to 26 percent F-measure.

We conceived and implemented a knowledge transfer between two deduplication networks, which shows that knowledge that accumulates during the training in the weight matrices of one network can be transferred to another. By transferring the matrices of selected attributes, we succeeded in increasing the overall network performance by +4.7 and +4.6 percent. This is a significant performance increase when compared to the respective baselines. In addition, we showed in a subsequent experiment that it is possible to reduce the amount of training data by performing a knowledge transfer. We tested our method on multiple datasets and compare our approach to other state-of-the-art methods.

Although we have shown that weight matrices can be transferred from one attribute to another, attributes for which we do not have pre-trained weight matrices must be initialized randomly, which is not ideal. This problem becomes particularly serious if the target dataset has a large number of attributes, but only a few of them can be initialized with pre-trained weight matrices. In this respect, it would be interesting to investigate whether the knowledge for certain attributes can also be captured by applying sequence to sequence learning [54]. By doing so, large amounts of existing data could be leveraged to generate vectors containing information for specific attribute domains, such as addresses, names, or descriptions. The vectors trained in this way could then be used to, in an ideal case, initialize all attributes of the target dataset to further improve the performance of duplicate detection.

While being one of the most widely used network designs, which in our case also resulted in good network performance, the BLSTM architecture design does not exhibit particularly good runtime performance. As runtime aspects were not the main focus of this work, it remains to be investigated whether it is possible to replace the LSTM by more performant GRU cells in the recurrent model to improve the runtime performance without negatively affecting the classification performance.

Another interesting question worth investigating would be to what extent multitask learning [9] can be used for duplicate detection. For example, it is conceivable that a network could be adapted to use multiple loss functions to train special similarity measures between the individual attribute pairs, while at the same time using these measures to optimize the classification of the individual entities into duplicate and non-duplicate pairs.

REFERENCES

[1] Asma Abboura, Soror Sahri, Mourad Ouziri, and Salima Benbernou. 2015. CrowdMD: Crowdsourcing-based approach for deduplication. In *Proceedings of the IEEE International Conference on Big Data*. 2621–2627.

[2] Zohra Bellahsene, Angela Bonifati, and Erhard Rahm (Eds.). 2011. *Schema Matching and Mapping*. Springer.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* 5 (1994), 157–166.

[4] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*. MIT Press, 2546–2554.

[5] Mikhail Bilenko and Raymond J. Mooney. 2003. Adaptive duplicate detection using learnable string similarity measures. *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD'03)* (2003), 39–48.

[6] Jens Bleiholder and Felix Naumann. 2008. Data fusion. *Comput. Surveys* 41, 1 (2008), 1–41.

[7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Trans. Assoc. Comput. Linguist.* 5 (2017), 135–146.

[8] Jane Bromley, James W. Bentz, Leon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak. Shah. 1993. Signature verification using Siamese time delay neural networks. *Int. J. Pattern Recogn. Artific. Intell.* (1993), 688–669.

[9] Rich Caruana. 1997. Multitask learning. *Mach. Learn.* 28 (1997), 41–75.

[10] Sung-Hyuk Cha. 2007. Comprehensive survey on distance/similarity measures between probability density functions. *Int. J. Math. Models Methods Appl. Sci.* 1 (2007), 300–307.

[11] Sumit Chopra, Raia Hadsell, and Yann LeCun. 2005. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 539–546.

[12] Peter Christen. 2007. A two-step classification approach to unsupervised record linkage. *Proceedings of the Australasian Conference on Data Mining and Analytics*. 111–119.

[13] Peter Christen. 2008. Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD'08)*. ACM, 151–159.

[14] Peter Christen. 2012. *Data Matching—Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer.

[15] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. 2003. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the International Workshop on Information Integration on the Web (IIWeb'03)*. AAAI Press, 73–78.

[16] Sanjib Das, AnHai Doan, Suganthan G. C. Paul, Chaitanya Gokhale, and Pradap Konda. [n.d.]. The Magellan Data Repository. Retrieved from https://sites.google.com/site/anhaidgroup/useful-stuff/data.

[17] Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. 2012. ZenCrowd: Leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *Proceedings of the International World Wide Web Conference (WWW'12)*. 469–478.

[18] Lee R. Dice. 1945. Measures of the amount of ecologic association between species. *Ecology* 26, 3 (1945), 297–302.

[19] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed representations of tuples for entity resolution. *Proc. VLDB Endow.* 11, 11 (2018), 1454–1467.

[20] Mohamed G. Elfeky, Vassilios Verykios, and Ahmed Elmagarmid. 2002. TAILOR: A record linkage tool box. In *Proceedings of the International Conference on Data Engineering (ICDE'02)*. 17–28.

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

[22] Raul Castro Fernandez and Samuel Madden. 2019. Termite: A system for tunneling through heterogeneous data. In *Proceedings of the International Conference on Management of Data (SIGMOD'19)*. 7:1–7:8.

[23] Donatella Firmani, Barna Saha, and Divesh Srivastava. 2016. Online entity resolution using an Oracle. *Proc. VLDB Endow.* 9 (2016), 384–395. Issue 5.

[24] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10)*. 249–256.

[25] Karl Goiser and Peter Christen. 2006. Towards automated record linkage. In *Proceedings of the Australasian Conference on Data Mining and Analystics*. Australian Computer Society, Inc., 23–31.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[27] Yash Govind, Erik Paulson, Mukilan Ashok, Suganthan G. C. Paul, Ali Hitawala, AnHai Doan, Youngchoon Park, Peggy L Peissig, Eric LaRose, and Jonathan C. Badger. 2017. Cloudmatcher: A cloud/crowd service for entity matching. In *Proceedings of the KDD Workshop on Big Data as a Service (BIGDAS'17)*.

[28] Lifang Gu and Rohan A. Baxter. 2006. Decision models for record linkage. In *Data Mining—Theory, Methodology, Techniques, and Applications*. Springer, 146–160.

[29] Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR'06)*. 1735–1742.

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

[31] Paul Jaccard. 1901. Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles* 37 (1901), 241–272.

[32] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* 22, 10 (2010), 1345–1359.

[33] Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. 2019. Low-resource deep entity resolution with transfer and active learning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL'19)*. 5851–5861.

[34] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *Proceedings of the International Conference for Learning Representations (ICLR'15)*.

[35] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. In *Proceedings of the International Conference on Machine Learning (ICML'15) Deep Learning Workshop*.

[36] Prodromos Kolyvakis, Alexandros Kalousis, and Dimitris Kiritsis. 2018. DeepAlignment: Unsupervised ontology matching with refined word vectors. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT'18)*. 787–798.

[37] Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Phys. Doklady* 10, 8 (1966), 707–710.

[38] Iaroslav Melekhov, Juho Kannala, and Esa Rahtu. 2016. Siamese network features for image matching. In *Proceedings of the International Conference on Pattern Recognition (ICPR'16)*. 378–383.

[39] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS'13)*. 3111–3119.

[40] Alvaro E. Monge and Charles P. Elkan. 1996. The field matching problem: Algorithms and applications. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD'96)*. 267–270.

[41] 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 *Proceedings of the International Conference on Management of Data (SIGMOD'18)*. 19–34.

[42] Jonas Mueller and Aditya Thyagarajan. 2016. Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'16)*. 2786–2792.

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

[44] Azade Nazi, Bolin Ding, Vivek R. Narasayya, and Surajit Chaudhuri. 2018. Efficient estimation of inclusion coefficient using HyperLogLog sketches. *Proc. VLDB Endow.* 11, 10 (2018), 1097–1109.

[45] Paul Neculoiu, Maarten Versteegh, and Mihai Rotaru. 2016. Learning text similarity with siamese recurrent networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL'16).*

[46] Sahand Negahban, Benjamin I. P. Rubinstein, and Jim Gemmell. 2012. Scaling multiple-source entity resolution using statistically efficient transfer learning. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM'12).* 2224–2228.

[47] M. Odell and R. Russell. 1918. The soundex coding system. *U.S. Patents* 1261167 (1918).

[48] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2012. Understanding the exploding gradient problem. *CoRR* abs/1211.5063. http://arxiv.org/abs/1211.5063.

[49] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'14).* 1532–1543.

[50] Sunita Sarawagi and Anuradha Bhamidipaty. 2002. Interactive deduplication using active learning. *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD'02).* 269–278.

[51] Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.* 45 (1997), 2673–2681.

[52] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems.* MIT Press, 2960–2968.

[53] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (2014), 1929–1958.

[54] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems.* MIT Press, 3104–3112.

[55] Sheila Tejada, Craig A. Knoblock, and Steven Minton. 2002. Learning domain-independent string transformation weights for high accuracy object identification. *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD'02).* 350–359.

[56] Vasilis Verroios and Hector Garcia-Molina. 2015. Entity resolution with crowd errors. In *Proceedings of the International Conference on Data Engineering (ICDE'15).* 219–230.

[57] Jiannan Wang, Tim Kraska, Michael J. Franklin, and Jianhua Feng. 2012. CrowdER: Crowdsourcing entity resolution. *Proc. VLDB Endow.* 5 (2012), 1483–1494. Issue 11.

[58] William E. Winkler and Yves Thibaudeau. 1991. An application of the Fellegi-Sunter model of record linkage to the 1990 U.S. decennial census. *U.S. Bureau of the Census* (1991), 1–22.

[59] Chen Zhao and Yeye He. 2019. Auto-EM: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *Proceedings of the International World Wide Web Conference (WWW'19).* 2413–2424.

[60] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-scale domain search. *Proc. VLDB Endow.* 9, 12 (2016), 1185–1196.

Received December 2019; revised May 2020; accepted July 2020