Deep SMAnE - Deep Similarity Matrix Adjustment and Evaluation to Improve Schema Matching

Roee Shraga
shraga89@campus.technion.ac.il
Technion – Israel Institute of Technology

Avigdor Gal
avigal@technion.ac.il
Technion – Israel Institute of Technology

Haggai Roitman
haggai@il.ibm.com
IBM Research - AI

ABSTRACT

Schema matching is at the basis of integrating structured and semi-structured data, serving as a handy tool in multiple contemporary business and commerce applications. Being investigated in the fields of databases, AI, semantic Web and data mining for many years, the core challenge still remains the ability to create quality matchers, automatic tools for identifying correspondences among data concepts (e.g., database attributes). In this work, we offer a novel post processing step to schema matching that improves the final matching outcome without human intervention. We present a new mechanism, similarity matrix adjustment, to calibrate a matching result and propose an algorithm (dubbed ADnEV) that manipulates, using deep neural networks, similarity matrices, created by state-of-the-art matchers. ADnEV learns two models that iteratively adjust and evaluate the original similarity matrix. We show conditions for dominance and convergence of ADnEV and demonstrate empirically the effectiveness of the proposed algorithmic solution for improving matching results, using real-world benchmark ontology and schema sets.

ACM Reference Format:

1 INTRODUCTION

The problem of schema matching is at the basis of integrating structured and semi-structured data. The schema matching task revolves around providing correspondences between concepts describing the meaning of data in various heterogeneous, distributed data sources, such as SQL and XML schemata, entity-relationship diagrams, ontology descriptions, interface definitions, and forms format [35]. The rapid growth in data source volume, variety, and veracity increases the need of schema matching in many applications in a variety of domains including data warehouse loading and exchange, integrating displays in interactive data analysis [30], aligning ontologies for the semantic web, and business document format merging (e.g., orders and invoices in e-commerce) [35]. As an example, consider a shopping comparison app, answering queries to find "the cheapest computer among retailers" or "the best rate for a hotel in Alaska in August." Such an app requires integrating and matching several data sources of product purchase orders and airfares Web forms.

Schema matching research originated in the database community [35] and has been a focus for other disciplines as well, from artificial intelligence [17] to semantic web [13] to data mining [19]. This research field has been going on for more than 30 years now, focusing on designing high quality matchers, automatic tools for identifying correspondences among database attributes.

Schema matching is challenging. Even when schemata describe identical concepts, there may still be structural and semantic differences, not to mention heterogeneous forms of representation. Over the years, numerous heuristic attempts were suggested for handling the problem (e.g., COMA [5] and Similarity Flooding [29]) followed by theoretical grounding (e.g., [1, 9]), showing that schema matchers are inherently uncertain [6, 9] mainly due to the ambiguity and heterogeneity of data description concepts.

Various mechanisms were proposed for uncertainty management in the matching process. Matching predictors [42] assess the quality of matchers without a reference match. Human experts are presented with a top-K ranked list of matches [34]. A recent work combined both approaches, using human engineered features to learn the ranking of better matches at the top of a list [12].

An alternative approach, widely used, involves human validation of algorithmic matchers [50], assuming human matcher superiority over algorithmic ones. Such an assumption is at best naïve, partially because different human matchers may differ in the way they match schemata [44]. Also, evaluation typically partitions the schema matching task into small-sized tasks to suit crowd sourcing, exposing the results to spamming [47] and misunderstandings mainly caused by the variability among workers [39].

In this work, we offer a novel post processing step to schema matching that improves the final matching outcome without human support, whether as validators or as matchers. We present a new mechanism, similarity matrix adjustment, to automatically calibrate a matching result conceptualized in a similarity matrix. Then, aided by deep neural networks, we propose the ADnEV algorithm to iteratively adjust and evaluate similarity matrices, created by state-of-the-art matchers. It learns two parallel models for adjusting and evaluating a similarity matrix, and uses them to manipulate the original matrix. With such a tool at hand, data integration practitioners can enhance their ability to introduce new data sources to existing systems without the need to rely on domain experts and data integration specialists. Our main contribution is threefold:

- A novel framework for schema matching using similarity matrix adjustment and monotonic evaluators (Section 2).
• Similarity matrices serve as learning features in an algorithmic solution (ADnEV) based on deep neural networks to improve the matching process outcome. Conditions for dominance and convergence of ADnEV are provided (Section 3).

• A large-scale empirical evaluation, using real-world benchmark ontology and schema sets, supports the practical effectiveness of the proposed algorithmic solution for improving matching results (Section 4).

In addition, related work is given in Section 5 and concluding remarks are provided in Section 6.

2.2 Similarity Matrix Adjustment
Recall that a schema matcher’s result is represented as a similarity matrix. A similarity matrix adjustment is a process that uses a mapping SMA : M → M, which transforms a similarity matrix into a (potentially) better adjusted similarity matrix (with respect to some evaluation criteria). Unlike matchers (Section 2.1), which operate over the schemata themselves, adjustment solely operates in the similarity matrix space. In the context of schema matching, such an adjustment process is typically referred to in the literature as a second line matcher (2LM) [9]. 2LMs may come in two flavors, namely decisive or non-decisive. The former manipulates the similarity matrix to determine which of the similarity matrix entries remain non-zero (and hence part of a match). The latter are meant to improve the matrix with respect to some evaluation criterion. Non-decisive matchers are usually used in tasks like pay-as-you-go schema matching [4] and schema cover [10].

Example 2. We now present five SMA, based on known 2LMs. Threshold(ν) and Max-Delta(δ) [5] apply selection rules to eliminate background noise in a similarity matrix. Threshold(ν) keeps those entries (i,j) having $M_{ij} \geq \nu$ while nullifying the others. Max-Delta(δ) selects those entries that satisfy: $M_{ij} + \delta \geq \max(\delta, M_{ij}, \max_j M_{ij})$. Maximum weighted bipartite graph match (MWBG) [15] and stable marriage (SM) [27] use well-known matching algorithms, given a score or ordering over elements. Finally, Dominants [9] selects correspondences that dominate (i.e., have the maximal similarity value) all entries in their row and column.

Example 2 (cont.). Figure 2 illustrates the general matching process, resulting in a similarity matrix, and Figure 3 provides an example of a similarity matrix over the two purchase order schemata from Figure 1. The similarity matrix is the outcome of Term [9], a string-based matcher. The projected match includes all correspondences besides $(S2.orderNumber, S1.city), (S2.city, S1.poCode))$ with $f(M) = 2.19$. 2.2.1 Schema Matching Model
The schema matching model, presented next, is mainly based on [9]. Let $S, S'$ be two schemata with the unordered set of attributes $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_m)$, respectively. A matching process matches schemata by aligning their attributes using matching algorithms (matchers for short), which deduce similarity using data source characteristics, e.g., attribute labels and domain constraints.

A matcher’s output can be conceptualized as a similarity matrix, denoted hereinafter $M(S, S')$ (or simply $M$), having entry $M_{ij}$ (typically a real number in $[0, 1]$) represents a degree of similarity between $a_i \in S$ and $b_j \in S'$. $M$ is a binary similarity matrix if for all $1 \leq i \leq n$ and $1 \leq j \leq m, M_{ij} \in \{0, 1\}$. $M \in \mathbb{R}^{n \times m}$ is the set of all possible similarity matrices. A match between $S$ and $S'$ consists of all non-zero entries of $M$.

Let $f(M)$ denote a schema pair similarity function, assigning an overall value to a similarity matrix, $M$. Typically, matching similarity functions are monotone submodular, e.g., $f(M) = \sum_{i=1}^{n} \sum_{j=1}^{m} M_{ij}$.
2.3 Similarity Matrix Evaluation

Let $M^*$ be a binary matrix, which represents a reference match such that $M_{ij} = 1$ whenever the correspondence $(a_i, b_j)$ is part of the reference match of the schema pair $(S, S')$ and $M_{ij} = 0$ otherwise. Reference matches are typically compiled by domain experts over the years in which a dataset has been used for testing. Given a reference match, similarity matrices can be measured using an evaluation function, $E_{Me} : M \rightarrow [0, 1]$, assigning a score to a similarity matrix according to its ability to identify correspondences in the reference match matrix. Whenever the reference match is clear from the context, we shall refer to $E_{Me}$ simply as $E$.

The most common evaluation functions in schema matching are precision ($P$) and recall ($R$), defined as follows:

$$P_M(M) = \frac{|M^* \cap M^+|}{|M^+|} \quad (1) \quad R_M(M) = \frac{|M^* \cap M^+|}{|M^*|} \quad (2)$$

where $M^+$ and $M^*$ represent the non-zero entries of $M$ and $M^*$, respectively. Using precision and recall, the F1 measure, $F^1$, is calculated as the harmonic mean of $P_M(M)$ and $R_M(M)$. We refrain from mentioning the reference match whenever the context is clear, referring to $P(M), R(M)$, and $F(M)$.

Sagi and Gal proposed methods for evaluating non-binary similarity matrices using a matrix-to-vector transformation of a matrix $M$ into a $(n \times m)$ size vector, given by $v(M)$ [43]. Using such a transformation, cosine similarity with respect to a reference match is defined as follows:

$$\cos_M(M) = \frac{v(M) \cdot v(M^*)}{||v(M)|| \cdot ||v(M^*)||} \quad (3)$$

Continuing with Example 1, let the reference match be $\{(order-Date, poDay), (orderDate, poTime), (orderNumber, poCode), (city, city)\}$ and let $M^*$ and $M^{**}$ be the corresponding matrices of Threshold and MWBG, respectively; then we have $P(M^*) = .43, R(M^*) = .75, F(M^*) = .55$ and $P(M^{**}) = .57, R(M^{**}) = .5, F(M^{**}) = .57$ respectively. Transforming the matrix of Figure 3 into a vector representation: $(.22, .11, .11, .09, .09, .00, .20, .17, .00, 1.0)$, further yields $\cos_M(M) = .65$.

2.4 Similarity Matrix Adjustment & Evaluation

Typically, SMAs in the literature (see Section 2.2) are limited in the way they transform a similarity matrix, not taking into account evaluation while performing adjustment. Therefore, SMAs may assume constraints such as 1 : 1 matching (e.g., MWBG and SM) or limiting the space of the output matrix values to the original similarity values (e.g., Threshold, Max-Delta, and Dominants). Some researchers also proposed to provide additional information as part of the adjustment process. He and Chang [19] suggested an holistic approach by matching more than a pair of schemata while others suggested to use multiple matrices input to the matching system [26, 27]. What is common to all these efforts is that they determine matching rules that reflect the way humans believe schema matching decisions should be made. Our hypothesis is that a similarity matrix contains more than meets the eye, with additional, hidden information that is not captured by human designed rules. The similarity matrix abstraction captures rich structures (including taxonomies, ontologies, and others, see [9] for details) that can be employed as part of the adjustment process.

Different evaluation functions emphasize different elements of the matching results, and SMAs should take that into account when aiming at optimized matrices. Consistent SMAs, therefore, should aim at improving matching outcome with respect to an evaluation function.

Definition 1 (CSMA: Consistent Similarity Matrix Adjustment). Let $E$ be an evaluation function. A similarity matrix adjustment mapping SMA is consistent if for any similarity matrix $M \in M$, it holds that: $E(M) \leq E(SMA(M))$.

Clearly, a consistent SMA does not decrease the quality of the original matrix with respect to an evaluation function. While this is beneficial, assuring we do not harm the initial performance still does not guarantee an improvement. In what follows, we therefore define strictly consistent SMAs to assure performance improvement.

Definition 2 (SCSMA: Strictly Consistent Similarity Matrix Adjustment). Let $E$ be an evaluation function and $\epsilon > 0$ an adjustment factor. A similarity matrix adjustment mapping SMA is strictly consistent if for any similarity matrix $M \in M$, it holds that: $\min(E(M) + \epsilon, 1) \leq E(SMA(M))$.

By Definition 2, strictly consistent SMAs can detect perfect matches. Note that while a reference match always provides a perfect match, the opposite may not necessarily hold. For example, if a matrix covers all correct correspondences but also includes some incorrect ones, it is perfect with respect to recall but not with respect to precision. To illustrate, consider the similarity matrix from Figure 3. For $S2.orderNumber$, the first three attributes in $S1$ are equally likely to match. Focusing on recall, one might choose all first three candidate attributes to be included in the final match whereas for precision none may be chosen.

To assess our ability to adjust similarity matrices, we characterize next a well-behaved similarity matrix evaluation with respect to a specific evaluation function it aims to maximize, as the matrix is being adjusted.

Definition 3 (Monotonic Similarity Matrix Evaluation). Let $E$ be an evaluation function. $\hat{E}$ is monotonic (with respect to $E$) if for any two similarity matrices $M, M' \in M$,

$$E(M) \leq E(M') \iff \hat{E}(M) \leq \hat{E}(M')$$

A monotonic evaluator, according to Definition 3, “behaves” approximately the same as the evaluation function it estimates.

3 ADNEV: ADJUST & EVALUATE

The ADnEV algorithm tackles a common limitation of most state-of-the-art matchers, which transform a local similarity value $M_{ij}$ to a match decision (i.e., whether $a_i \in S$ and $b_j \in S'$ correspond), disregarding the broader scope of this correspondence in a matching system. Those SMAs that aim at broadening the matching scope use human designed rules such as 1 : 1 matching and dominance...
relationship (see Example 2). The ADnEV algorithm utilizes deep neural networks to capture global relationships between elements of the similarity matrix, enriching both adjustment and evaluation. To this end, ADnEV combines two interacting networks, one for adjusting and the other for evaluating similarity matrices. Given a strictly consistent adjustor and a monotone evaluator, ADnEV satisfies two essential properties: Dominance and Convergence.

3.1 Background: Neural Networks (NN)

The basic model in deep learning is a fully connected layer, which takes as an input a vector $X$ and performs a non-linear transformation $wX + b$ of the input using activation functions to produce an output (e.g., $ReLU(wX + b)$). Deep NNs (DNN) feed the output of one layer to another. The non-input/output layers consist of hidden units which can be seen as transforming the input in a non-linear fashion and extract hidden features such that the last layer can separate the input into categories (e.g., match/non-match). Multilayer NNs can be trained by simple stochastic gradient descent and, as long as the activation functions are relatively smooth with respect to the outputs and internal weights, one can compute the gradients using a backpropagation procedure. For more information see [16, 24].

Next we present two main classes of NNs, namely Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs).

The RNNs class is rooted in processing of sequential data [41] with applications in Natural Language Processing [2], Time Series Prediction [3], and Entity Resolution [8, 31]. RNNs process the input sequentially, one element at a time, maintaining hidden units that implicitly contain the history of past elements. The outputs of the hidden units at different time steps can be seen as hidden units in a DNN in which all the layers share the weights. Gated mechanisms have been developed to compensate some limitation of the basic RNN. Following [48], we use a Gated Recurrent Unit (GRU)-based RNN [2] in this work. A GRU uses a reset gate and an update gate. The update gate decides how much of the information of past states should pass through and the reset gate decides how much of past information should be disregarded.

The second class of NN is CNN, which is rooted in processing grid-like topology data [23] with applications in Image and Video Processing [22], Recommender Systems [46], and more. Convolution preserves the spatial relationship between cells by learning matrix features using small subareas of the input data. The first and main layer of a CNN is the convolutional layer, which detects conjunction of features from the previous layer. This layer uses filters to detect local features, which results in a feature map where each unit is connected to the previous layer through a set of weights. The pooling layer merges semantically similar features by computing maximum (or average) of a local patch of units into one feature map that can later connect to a fully connected layer. The convolution layer and the pooling layer can be fine-tuned with respect to hyperparameters such as the size of filtering, the strides in which the number of neighboring cells are considered and padding each side of the boundaries of the input.

3.2 Learning to Adjust & Evaluate

Both RNN and CNN may capture (hidden) structural properties of the similarity matrix. RNNs process the data sequentially, allowing the model to maintain a sequential memory and recognize patterns in the similarity matrix that may affect both local (adjustor) and overall (evaluator) decisions. As the examined similarity matrices are usually big, gates (like in GRU), enable models to learn indirect matrix dependencies. CNNs identify grid-like patterns in a matrix, making the model aware of spatial dependencies within the matrix. The convolution can highlight areas in a matrix that are important in determining the overall evaluation of the matrix and discover interacting sections within the matrix. Finally, we suggest the use of CRNN, combining convolutional and recurrent layers as a part of the same network (empirically validated choice, see Appendix A.1).

We present next a unified approach for Deep SMAnE (deep similarity matrix adjustment and evaluation), taking advantage of NNs flexibility in the way output layers are defined. Using supervised learning, the input for the adjustor and the evaluator is a set of $K$ similarity matrices $M^{(K)} = \{M_k\}^K_{k=1}$ and their respective reference matches $\{M_k^E\}^K_{k=1}$. An adjustor applies an SMA mapping, returning a matrix in $\hat{M}$ (the space of similarity matrices). The evaluator returns a value in $[0, 1]$.

The adjustor receives similarity matrices $\{M_{ij} \in [0, 1]\}^{n \times m}_{i=1, j=1}$ as training examples ($M \in M$) and their respective labels: $l(M_{ij}) = 1$ if $M_{ij} \in M^E$ (otherwise $l(M_{ij}) = 0$). Adjustors solve an entry-wise binary classification problem and are trained using a binary cross entropy (CE) loss defined over each train matrix $M$, as follows:

$$CE(M) = -\sum_{i=1}^{n} \sum_{j=1}^{m} (l(M_{ij}) \cdot \log(\hat{M}_{ij}) + (1 - l(M_{ij})) \cdot \log(1 - \hat{M}_{ij})) \quad (4)$$

where $\hat{M}_{ij}$ denotes the predicted value of $l(M_{ij})$.

The evaluator also receives similarity matrices $\{M_{ij} \in [0, 1]\}^{n \times m}_{i=1, j=1}$ as training examples ($M \in M$) together with their overall evaluation function value, $E(M)$, calculated using a reference match (see Section 2.3). The evaluator solves a regression problem. Evaluators are tuned using a mean squared error (MSE) loss that, given $M^{E(K)}$, is computed as follows:

$$MSE(M^{E(K)}) = \frac{1}{K} \sum_{k=1}^{K} (\hat{E}(M_k) - E(M_k))^2 \quad (5)$$

where $\hat{E}(M_k)$ denotes the predicted value of $E(M_k)$.

Schema matching is a challenging task, thus it is really rare to get an ideal training example to help the model achieve optimum in terms of both adjustment and evaluation. Therefore, to enhance our models, we also include the reference matrices as examples during training. A reference matrix obtains a perfect score with respect to both adjustor (all labels are correct by definition) and the evaluator (all evaluation functions will return a value of 1 given the reference matrix). Adding a reference matrix as a training example helps the model to maintain a sequential memory and recognize patterns in the similarity matrix that may affect both local (adjustor) and overall (evaluator) decisions. As the examined similarity matrices are usually big, gates (like in GRU), enable models to learn indirect matrix dependencies. CNNs identify grid-like patterns in a matrix, making the model aware of spatial dependencies within the matrix. The convolution can highlight areas in a matrix that are important in determining the overall evaluation of the matrix and discover interacting sections within the matrix. Finally, we suggest the use of CRNN, combining convolutional and recurrent layers as a part of the same network (empirically validated choice, see Appendix A.1).

We present next a unified approach for Deep SMAnE (deep similarity matrix adjustment and evaluation), taking advantage of NNs flexibility in the way output layers are defined. Using supervised learning, the input for the adjustor and the evaluator is a set of $K$ similarity matrices $M^{(K)} = \{M_k\}^K_{k=1}$ and their respective reference matches $\{M_k^E\}^K_{k=1}$. An adjustor applies an SMA mapping, returning a matrix in $\hat{M}$ (the space of similarity matrices). The evaluator returns a value in $[0, 1]$.

The adjustor receives similarity matrices $\{M_{ij} \in [0, 1]\}^{n \times m}_{i=1, j=1}$ as training examples ($M \in M$) and their respective labels: $l(M_{ij}) = 1$ if $M_{ij} \in M^E$ (otherwise $l(M_{ij}) = 0$). Adjustors solve an entry-wise binary classification problem and are trained using a binary cross entropy (CE) loss defined over each train matrix $M$, as follows:

$$CE(M) = -\sum_{i=1}^{n} \sum_{j=1}^{m} (l(M_{ij}) \cdot \log(\hat{M}_{ij}) + (1 - l(M_{ij})) \cdot \log(1 - \hat{M}_{ij})) \quad (4)$$

where $\hat{M}_{ij}$ denotes the predicted value of $l(M_{ij})$.

The evaluator also receives similarity matrices $\{M_{ij} \in [0, 1]\}^{n \times m}_{i=1, j=1}$ as training examples ($M \in M$) together with their overall evaluation function value, $E(M)$, calculated using a reference match (see Section 2.3). The evaluator solves a regression problem. Evaluators are tuned using a mean squared error (MSE) loss that, given $M^{E(K)}$, is computed as follows:

$$MSE(M^{E(K)}) = \frac{1}{K} \sum_{k=1}^{K} (\hat{E}(M_k) - E(M_k))^2 \quad (5)$$

where $\hat{E}(M_k)$ denotes the predicted value of $E(M_k)$.

Schema matching is a challenging task, thus it is really rare to get an ideal training example to help the model achieve optimum in terms of both adjustment and evaluation. Therefore, to enhance our models, we also include the reference matrices as examples during training. A reference matrix obtains a perfect score with respect to both adjustor (all labels are correct by definition) and the evaluator (all evaluation functions will return a value of 1 given the reference matrix). Adding a reference matrix as a training example serves as a desiderata for the best similarity matrix for a given schema pair and assist in homing in quickly on the appropriate model weights.

Given an input similarity matrix $M$, the adjustor, $AD(M)$, returns another similarity matrix $\hat{M}$ and the evaluator, $EV(M)$, returns an evaluation value $\hat{E}(M)$.

$^1$E.g., looking at the “easiest” dataset (see Section 4.1), Web-Forms, a-priori, only one preference similarity matrix out of 147 pairs was created with 18 state-of-the-art matcher combinations applied.
3.3 Adjust & Evaluate (ADnEV) - Algorithm

We present a combined Adjust and Evaluate (ADnEV) algorithm that performs iterative adjustments to a similarity matrix, given as input. The novelty underlying the ADnEV algorithm is to make the SMA evaluation aware. The algorithm wraps two pre-trained networks (the adjustor AD and the evaluator EV) with an iterative algorithm that, given a new unseen similarity matrix (test phase), attempts to adjust (and readjust) the output of an AD. The algorithm evaluates (using EV) the output at each step to decide whether the adjusted matrix is better than its predecessor until no improvement is achieved.

Figure 5 illustrates the ADnEV algorithm, ADnEV readjusts an input similarity matrix (M) as long as it continues to improve its evaluation value (as determined by EV). We set $M^0 = M$ and at each step $t = 1, 2, \ldots$ ADnEV assigns $M^t = AD(M^{t-1})$ and checks the following conditions to decide whether to continue to step $t+1$:

$$\begin{cases} 
\text{continue to step } t+1, & \text{if } EV(M^t) > EV(M^{t-1}) \\
\text{return } M^{t-1}, & \text{otherwise} 
\end{cases}$$

(6)

The adjustment process aims to improve the matching outcome, thus the algorithm should not weaken the initial matching result. The main component of the algorithm in charge of assessing the matching outcome is the evaluator. Given a well-behaved evaluator, the ADnEV Dominance assures that the output of the similarity matrix will be at least as good as the outcome of the input similarity matrix.

**Theorem 3.1 (ADnEV Dominance).** Let $EV(\cdot)$ be a monotonic evaluator (Definition 3) and let $M^*$ be the output of ADnEV, then the following holds: $E(M^*) \geq E(M^*)$.

Given a well-behaved adjustor, ADnEV is guaranteed to return an ideal matrix with respect to the evaluation function of interest.

**Theorem 3.2 (ADnEV Convergence).** Let $AD(\cdot)$ be a SCSMA (Definition 2) with $0 < \epsilon \leq 1$, let $EV(\cdot)$ be a monotonic evaluator (Definition 3), and let $M^*$ be the output of ADnEV. ADnEV returns a matrix with an ideal evaluation ($E(M^*) = 1$) in less than $\frac{1}{\epsilon}$ steps.*

ADnEV is an iterative algorithm, iterating until its halting condition is met, which in our case means the return of an adjusted matrix that is worse than its predecessor. An algorithm that always returns a better adjusted matrix will lead to convergence and an optimal output similarity matrix.

### 3.3.1 Multi-Task Adjust and Evaluate Learning

We also suggest the use of a multi-task NN (Multi) with two objectives, one corresponding to SMA (Multi.AD) while the other corresponding to matrix evaluation (Multi.EV), enabling the network to learn a joint representation of the input matrix while updating the weights with respect to both objectives (see architecture in Section 3.3.2). When Multi is used, we set $AD = Multi.AD$ and $EV = Multi.EV$. Multi combines both networks via a joint loss based on $CE$ (Eq. 4) and $MSE$ (Eq. 5) uniformly weighted, i.e., $0.5 \cdot CE(M^K) + 0.5 \cdot MSE(M^K)$.

### 3.3.2 ADnEV Network Architecture

For each of the networks (AD and EV) we use a CRNN, combining GRU- and CNN-based layers. We ran a validation phase over the Beta dataset (see Appendix A.1) and validated that CRNN performs better than GRU and CNN independently. Dropout did not improve results, thus not included in the networks.

With a variable size input matrices, we begin with a FCN (Fully Convolutional Network), following [25]. The rest is illustrated in Figure 6. For the single objective AD and EV we have omitted the bottom and top two layers of the network respectively. (see Appendix A.1 for more details)

## 4 EMPIRICAL EVALUATION

We now present an empirical evaluation of the ADnEV algorithm, evaluating each component separately (i.e., adjustor and evaluator) and jointly using a variety of real-world datasets. We establish the practical usage of the proposed algorithm by experimenting with a variety of application datasets, ranging from small scale web form schemata extracted from domains such as flights, dating, booking, etc. to large purchase orders used by retailers and bibliographic references ontologies from various resources.

### 4.1 Datasets

Table 1 summarizes the details of the datasets we used for our experiments. Web-forms [11] contains schemata of varying sizes from diverse domains, automatically extracted from Web forms using the OntoBuilder extractor. Reference matches were manually constructed by human judges. Purchase Order [5] contains XML documents describing purchase orders extracted from various systems. OAEI contains ontologies from the 2011 competition in the comparison track, using the bibliographic references domain, comparing 100 ontologies with a reference ontology. The three benchmark datasets were designed to present schema pairs with differing levels of difficulties, introducing alongside easy matches also complex relationships and attributes, which may yield low precision and recall levels, even when using the strongest matchers. We also created a synthetic dataset, following the model of Marie

---

*Theorem proofs are given in Appendix A.2

---

to perform a matchers quality independent evaluation, used for parameter tuning (see Appendix A.1).

### 4.2 Schema Matchers

Three schema matching tools were used, namely ORE, AMC, and COMA. OntoBuilder Research Environment\(^4\) (ORE) is a research prototype for large scale matching experiments. The Auto-Mapping Core (AMC)\(^3\), developed by SAP Research, provides an infrastructure and a set of algorithms to match business schemata. We used ORE’s embedded implementation of AMC’s Token Path algorithm. We also used two matchers from COMA 3.0,\(^5\) a state-of-the-art schema matching research tool, namely Threshold(\(\epsilon\)) and Max-Delta(\(\delta\)), whose details are given in Example 2. Together with five state-of-the-art SMAs (described in Section 2.2) we used three matchers (Term, WordNet, and Token Path) to generate our baseline set of similarity matrices. Term\(^9\) compares attribute names to identify syntactically similar attributes (e.g., using edit distance and soundex). WordNet\(^38\) uses abbreviation expansion and tokenization methods to generate a set of related words for matching attribute names. Finally, Token Path\(^33\) integrates node-wise similarity with structural information by comparing the syntactic similarity of full paths from root to a node.

#### 4.2.1 Similarity Matrix Augmentation

We created 18 similarity matrices per schema pair,\(^6\) yielding for example, 2,646 matrices for the Web-forms domain.

Using properties of schema attribute order independence and matching process symmetry, we also propose a similarity matrix augmentation method, as follows. Given a similarity matrix \(M\) and a requested number of augmentations \(A\), a transpose of \(M\), \(M^T\) is created and added to the training set. Then, for each iteration \(\alpha \in \{1, \ldots , A\}\), the following operations are applied:

1. Randomly select \(i, j\) from \(\{1, 2, \ldots , n\}\) and \(\ell, k\) from \(\{1, 2, \ldots , m\}\)
2. Replace the \(i\)th row with the \(j\)th row and the \(j\)th column with the \(k\)th column in both \(M\) and \(M^T\).
3. Add both \(M\) and \(M^T\) augmented versions to the training set.

To enhance reproducibility, we present an augmentation illustrative example (Example 2.1) in Appendix A.1.4.

Each similarity matrix is accompanied by \((2 + 2A)\) augmented matrices during the training phase. For example, with \(A = 4\), the Web-forms domain would have 26,460 matrices. Table 1 details the number of augmentations performed for each dataset and the total number of matrices used in the experiments. It is noteworthy that augmentation was applied only in the training phase.

### 4.3 Baselines and Tuning

We have implemented our networks using Keras\(^7\) with a tensorflow backend (see Section 3.3.2). The Adam\(^20\) optimizer with a learning rate of 0.001 and \(\beta_1 = 0.9, \beta_2 = 0.999\) was used, following\(^40\). Since the similarity matrices have varying sizes, we fed the network with each matrix independently (batchsize \(= 1\)); and because we perform augmentations over each matrix (see Section 4.2.1), we go over the dataset only once \((epochs = 1)\). Additional details, for reproducibility sake, are given in Appendix A.1.

We have tested our methods against several baselines. Evaluators were compared to \(f(M)\) (see Section 2.1) and to a feature based evaluation based on\(^14\), where we have used matching predictors\(^42\) as features to learn an evaluation model. We have also tested several state-of-the-art regression models using the scikit-learn\(^8\) package and present the results of Support Vector Regression (SVR)\(^7\), which obtained the best results.

Adjusters were compared to the original similarity matrices \((Ori)\). Given a resemblance of the adjustment problem to matrix factorization, we also compared our results to the following state-of-the-art methods: 1) \(SVD^{++}\)\(^{21}\), being a regular singular value decomposition method, \(SVD^{++}\) learns latent representations of \(a_i \in S, b_j \in S'\) and adjusts a similarity matrix by multiplying the representations per matrix entry 2) Bayesian Personalized Ranking (BPR)\(^{37}\) treats the corresponding pairs as implicit feedback and learns preference of candidate attributes with respect to a target attribute \((i.e., if M_{ij} > M_{ij}, then b_j \prec a_i, b_j)\). 3) Factorization Machines (FM)\(^{36}\) allows to capture high order non-linear interactions, and 4) Isotonic Regression (IR)\(^{32}\) learns a calibration function to order corresponding attribute pairs before non-corresponding attribute pairs. Finally, we have also compared the performance to a vanilla deep neural network (dubbed DNN), a fully connected network over the original similarity matrix values.

Evaluation was performed using 5-fold cross validation. We randomly split each dataset into 5 folds and repeat the experiment 5 times, using 4 folds for training and the remaining fold for testing. We report on the average performance over all folds.

### 4.4 ADnEV Algorithm Evaluation

We now present the main results of experimenting with the ADnEV algorithm. We start with the adjusters’ results, followed by the evaluators’ results, and lastly the combination of the two.

#### 4.4.1 Adjusters

We trained the CRNN-based adjuster as described in Section 4.3 and Appendix A.1 and compared it to the baselines described in Section 4.3. As an illustrative example, Figure 7 presents a gray scale color map\(^9\) of the deep adjustment process compared to an original similarity matrix generated by the Term matcher over a given Web-Forms schema pair. The figure visualizes the network architecture (see Figure 6) as it lays out actual trained layers with respect to a dataset on an input similarity matrix.

Table 2 reports the overall performance in terms of precision (P), recall (R), and F1 (F) obtained by the CRNN-based adjuster and other baseline SMAs. Statistically significant differences in

---

1. bitbucket.org/tomers77/ontobuilder-research-environment
3. Using the 3 matchers without an SMA, and also applying each of the 5 SMAs.
4. https://keras.io/
5. https://scikit-learn.org
6. 0 and 1 entries are colored white and black, respectively. Other entries are colored accordingly with entries closer to 1 colored darker than the ones closer to 0.
Figure 7: Adjustment Example. Examples are marked by a gray scale color map. 

Table 2: Precision (P), Recall (R), F1 (F) of Adjusted Matrices. Non-augmented results are given in parentheses.

Table 3: Correlation of the Evaluators with Precision (P), Recall (R), F1 (F), and Cosine (Cos). Non-augmented results are given in parentheses.

4.4.2 The Evaluators. We assess the quality of the suggested deep evaluator by comparing the Pearson’s-r correlation between the predicted values and the actual match quality, following previous works [12, 13, 42]. We expect monotone evaluators (Definition 3) to designate high correlation with respect to the evaluation measure they aim to estimate. For each given evaluation measure E, we have trained three different versions of each learning based evaluators, i.e., SVR (features based on [14] trained with SVR), DNN was, and finally our suggested deep evaluator CRNNe. To assist in reproducibility, an example of a deep evaluation process, similarly to Figure 7, is given in Appendix A.1.5.

Table 3 reports on the Pearson’s-r correlation obtained by the baselines and the newly proposed evaluator over the four match quality measures (see Section 2.3). A two-tailed t-test of significance at a 99% confidence level was performed against the null hypothesis of no-correlation (significant entries are denoted with an asterisk).

All our suggested evaluators have high statistically significant correlations with respect to their estimated binary evaluation function with an average of .78 for precision, .75 for recall, and .90 for F1 over all datasets. The non-binary evaluation, cos, seems to be the hardest to predict and especially when dealing with large matrices (Purchase Orders domain) demonstrates lower, yet significant, correlations. Moreover, CRNNe evaluators designated higher overall correlations (in absolute value) compared to all baselines. This may indicate that applying non-linear functions (as in CRNNe) evaluates a similarity matrix better than human feature engineering.

Yet, the feature based evaluators performed better than the DNN evaluators overall, indicating that a vanilla NN is not enough to outperform human knowledge. Evidently, our suggested evaluators behave approximately like monotone evaluators, validating their usage as a part of the ADnEv algorithm, to be described next.
the second combines both in one network (MULTIγ). An example ADnEV for a given schema pair is given in Appendix A.1.5.

Additionally, we have used our trained networks to seek the best similarity matrix possible for a given schema pair. To this end, we used ADnEV to create a set of similarity matrices for each schema pair out of which we selected the best similarity matrix possible according to our evaluator. We compared these results with two baselines; the first uses \( f(M) \) (see Section 2.1) and the second uses \( \text{LSRM} \) [14] over the same set of matrices to select the best one. This approach corresponds to earlier works on matchers ensemble [5]. A matcher ensemble considers multiple matchers as input and computes a weight for each one (which can be learned, e.g., using a boosting method [23]) to create a matching decision. Our method, instead, learns a model by considering individual similarity matrices, which can be then used to select a single similarity matrix.

Table 4 reports results in terms of precision (P), recall (R) and F1 (F). The upper side of the table reports the overall results averaged over all \textit{similarity matrices}, while the bottom reports average results of the \textit{best similarity matrix} for each schema pair in the dataset. Statistical significant differences in performance are marked with an asterisk using a paired two-tailed t-test with a Bonferroni correction for \textit{p-value} < .05.

Comparing the two ADnEV variants, MULTI performs slightly better than the straight forward utilization of both networks, i.e., CRNNγ. This fits the insight that combining the networks by their losses can help each network to collaborate with the other. ADnEV (MULTI) performs significantly better than the CRNN adjustor as it takes into account an evaluation function, MULTIγ boosts precision by 17%, MULTIγ boosts recall by 15%, and, MULTIγ boosts F1 by 11% on average over all datasets. Involving an emphasized evaluation function in the adjustment can boost the desired performance. It is also noteworthy that MULTIγcos had similar improvements, except for the Purchase Order domain, which actually fits the difficulty of the CRNN evaluator to correlate with cosine (see Section 4.4.2).

Finally, analyzing the best matching result for each schema pair, ADnEV is able to create (adjust) and detect (evaluate) very high quality results. Compared to selecting the best of the original matching results based on \( f(M) \), ADnEV improves the matching results by 51% on average precision (MULTIγp), by 39% on average recall (MULTIγr), and by 30% on average F1 (MULTIγ), over the datasets. This demonstrates the potential of using deep learning for resolving indirect relationships within a similarity matrix to generate better matches and to be able to know they are better.

### 5 RELATED WORK

Recent years have seen a big hype around deep learning, both theoretically [16] and applicatively [24]. Some problems in data integration, e.g., entity resolution [8, 31], have benefited from its use, yet schema matching thus far was not considered a good application domain for deep learning, due to insufficient amount of data. Existing work in data integration assume the use of additional or external information. For example, Mudgal et al. assume multiple matchers input, applying deep learning on top of the results [31]. Also, Ebraheem et al. assume a rich textual information (typical to the entity resolution domain but not to schema matching) to create word embeddings [8]. In this work, we allow the use of deep learning via the similarity matrix abstraction, capturing the result of matchers. Similarly to [12], similarity matrices serve as a basis for learning features and via deep learning we learn to adjust similarity matrices to obtain better match results.

Deep learning solutions were suggested for matching problems such as patch-based image matching [18] and graph matching [49]. These networks learn to compute a similarity matrix, given two elements (images/graphs). In this work the network input is a similarity matrix, to be evaluated and adjusted, posing different obstacles in achieving an effective solution.

Our work is the first to use deep learning for similarity matrix adjustment and evaluation and was shown here to give superior results on the tested datasets. Other methodologies were suggested in schema matching literature to achieve similar goals: 2LMs (SMAs) were used to adjust similarity matrices as introduced and discussed in Section 2.2. In this work, we define the notion of (strictly) consistent similarity matrix adjustment (Definitions 1 and 2) to set a desiderata for well-behaved adjuster and show empirically that the proposed deep adjuster approximate this property well. Moreover, we compared against state-of-the-art 2LM and show significant improvement using the proposed method.

Matching predictors were suggested to evaluate similarity matrices [42] in an unsupervised manner. A recent work used supervised learning, limited to F1 measure in the scope of top-K schema
matching based on a human engineered matching predictors as features [12]. We, instead, train a supervised deep learning evaluator with respect to an evaluation function of choice and show dominance (Section 4.4.2) over a feature-based evaluation. Moreover, we introduce the notion of a monotonic evaluator to characterize well-behaved evaluator (Definition 3) and empirically validated that our newly designed evaluator approximates well a monotonic evaluator, being highly correlated with a target evaluation measure.

6 CONCLUSIONS AND FUTURE WORK

In this work we proposed a post processing step to schema matching that uses deep neural networks to improve the final matching outcome. The ADnEV algorithm adjusts and evaluates input similarity matrices, as created by state-of-the-art matchers, in an iterative manner. We showed that using a strictly consistent similarity matrix adjustment and monotonic evaluation, the algorithm satisfies dominance and convergence. Finally, we empirically validated our algorithmic solution effectiveness using real-world benchmark ontology and schema sets.

We see this work as a proof-of-concept to using deep learning for schema matching, which fits well with the ongoing investigation of machine learning-based solutions to classical data management problems. Applying deep learning as a supervised learning tool, without the need of a human validator carries high promises for effective embedding of schema matching in contemporary applications that depend on continuous online large-scale integration of new data sources to remain competitive in dynamic markets.

In future work we intend to extend the suggested solution as we aim to address wider range of data integration problems and boost their performance.

REFERENCES

[18] Xufang Han et al. 2015. Matchnet: Unifying feature and metric learning for patch-based matching. In CVPR.
[37] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In AAM.
[50] DeepSMAnE Conference’17, July 2017, Washington, DC, USA
A SUPPLEMENT

In this supplement we provide additional information regarding hardware, implementation, and baselines. In addition, we provide a similarity augmentation example (see Section 4.2.1), deep similarity matrix evaluation example (see Section 4.4.2), and an example of applying ADnEV for a given schema pair. The supplement aims at achieving reproducibility of experiments and results.

A.1 Experimental Supplement

The main code repository for this project is available at https://github.com/shraga89/DSMA.

A.1.1 Hardware. Algorithms were implemented in Python and evaluation was performed using 8 GPU servers and 3 CPU servers. All have a CentOS 6.4 operating system. The main GPU server contains two Nvidia gtx2080 Ti, a second server contains two Nvidia GeForce gtx 1080 GPUs while the others contain a single NVIDIA Tesla K80 GPU. The main CPU server has 28 Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz, 128GB RAM cores while the others have 8 Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, 32GB RAM.

A.1.2 Implementation. Similarity matrices were generated using ORE. All networks were implemented using Keras with a tensorflow backend and are available online. Four variations of adjustment/evaluation were implemented and tested, namely, a vanilla version which is based on fully connected NN, an RNN only network based on GRU, a CNN only network, the CRNN network presented in the evaluation (Section 4), and the multi objective network that uses CRNN and two objectives based on adjustment and evaluation. We also created a synthetic dataset, following the model of Marie et al. [27], to perform an independent matcher quality evaluation, used for parameter tuning during validation. Similar to [12], we manipulated 147 reference match matrices from the Web-Forms dataset using two Beta distributions (which differ in their parameters), one for correspondences in the reference match and one for non-corresponding attributes. Corresponding attributes are assigned values that are closer to 1 and non-corresponding are assigned with values closer to 0 according to the respective Beta distribution. Over the resulted matrix, a 0.1 threshold was applied. Using the Beta dataset, we validated that CRNN layers perform better than RNN or CNN only. Therefore, their results were not reported. We also validated that Dropout (ε ∈ {0.1, 0.3, 0.5, 0.7}) did not improve results, thus not included in the networks. While only CRNN and Multi results were reported for the ADnEV algorithm, all evaluator-adjustor combinations were tested on the Beta dataset and can be ran using our implementation. Figures 8 and 9 present an adjustment only and evaluation only CRNN networks, respectively.

A.1.3 Baselines. All baselines are available online. For all evaluation baselines as well as the Isotonic Regression we used scikit-learn. A scikit-learn extension for recommender systems, was used for SVD++. We used BPR for the BPR implementation; and finally we used a python implementation of libFM, for FM. For evaluation baseline, we presented the Support Vector Regression (SVR) (Table 3). We have also tested the following alternative regressors: Linear Regression, Lasso Regression, Ridge Regression, Theil-Sen Regression, Passive Aggressive Regression, and SGD Regression, which performance was inferior to that of SVR.

A.1.4 Similarity Matrix Augmentation Example. In Section 4.2.1 we elaborate on similarity matrix augmentation. Given a similarity matrix M, we use its transpose MT and shuffle rows and columns. An illustrative example is given in Figure 10, using the similarity matrix of Example 2.1. First the transpose is generated (top right). Then, for each of the matrices we randomly select two rows and columns to swap (bottom).
A.2 Theorem Supplement

**Theorem A.1 (ADnEV Dominance).** Let EV(·) be a monotone evaluator (Definition 3) and let $M^*$ be the output of ADnEV then

$$E(M^*) \geq E(M)$$

**Proof A.1 (ADnEV Dominance).** Negatively assume that $E(M) > E(M^*)$ and let $EV(\cdot) = \hat{E}$

- If $\hat{E}(AD(M)) = \hat{E}(M^1) \leq \hat{E}(M)$ then:

  $$M^* = M^1 = M \text{ (the algorithm will terminate at the first step by Eq. 6)}$$

  $$\Rightarrow E(M) = E(M^1) = E(M^*) \Rightarrow \textbf{Contradiction}$$

- If $\hat{E}(AD(M)) = \hat{E}(M^1) > \hat{E}(M)$ then:

  let $M^T$ be the matrix in the $t$th step and let $T$ be the stopping iteration.

  $$M^t = M^{T-1} \text{ (by Eq. 6)}$$

  $$\Rightarrow \hat{E}(M^t) = \hat{E}(M^{T-1})$$

  Since the condition in Eq. 6 was satisfied for each $t \in \{1, \ldots, T-1\}$ step:

  $$\hat{E}(M^t) > \hat{E}(M^{t-1}) \Rightarrow \hat{E}(M^{t-1}) > \hat{E}(M^{t-2}) > \ldots \hat{E}(M^1) > \hat{E}(M^0) = \hat{E}(M) \Rightarrow$$

  $$\hat{E}(M^*) = \hat{E}(M^{t-1}) > \hat{E}(M) \Rightarrow$$

  Since $\hat{E}(\cdot)$ is a monotone evaluator (definition 3):

  $$E(M^*) > E(M) \Rightarrow \textbf{Contradiction}$$

**Theorem A.2 (ADnEV Convergence).** Let AD(·) be a SCMSA (Definition 2) with $0 \leq \varepsilon \leq 1$, let $E(\cdot)$ be a monotone evaluator (Definition 3), and let $M^*$ be the output of ADnEV. ADnEV algorithm returns a matrix with an ideal evaluation ($E(M^*) = 1$) in less than $\frac{T}{\varepsilon}$ iterations.

**Proof A.2 (ADnEV Convergence).** Let $M^t$ be the matrix in the $t$th step, let $M^*$ be the output of ADnEV and let $EV(M) = \hat{E}$.

Since $AD(\cdot)$ is a SCMSA:

$$E(M^{t-1} + \varepsilon) \leq E(M^t) = E(AD(M^{t-1})) \tag{7}$$

Note that since $\hat{E}(\cdot)$ is a monotone evaluator, we can assume that the algorithm will reach the $t$th iteration (based on Eq. 6) and since $\varepsilon \geq 0$:

$$E(M^{t-1} + \varepsilon) \leq E(M^t) \Rightarrow E(M^{t-1}) < E(M^t) \tag{8}$$

Eq. 7 holds for each iteration prior to $t < T$ and we obtain:

$$E(M^{t-2}) + \varepsilon \leq E(M^{t-1}) \Rightarrow E(M^{t-2}) + 2 \cdot \varepsilon \leq E(M^t) \ldots$$

Finally we get:

$$E(M^0) + t \cdot \varepsilon = E(M^t) + t \cdot \varepsilon \leq E(M^t) \tag{8}$$

Let $E(M) = 0$, representing the worst case, then we get:

$$E(M^t) \geq t \cdot \varepsilon \tag{9}$$

We denote $E(M^t)$ as a ideal evaluation matrix, i.e., $E(M^t) = 1$ and obtain:

$$E(M^t) \geq t \cdot \varepsilon \Rightarrow \exists t \cdot \varepsilon \Rightarrow t = \frac{1}{\varepsilon}$$

Running the algorithm for $\lceil \frac{T}{\varepsilon} \rceil$ steps (step number should be a discrete) we obtain (in the worst case) $M^* = M^\frac{T}{\varepsilon} \Rightarrow$ (Eq. 9)

$$E(M^*) = E(M^{\lceil \frac{T}{\varepsilon} \rceil}) \geq \left[ \frac{1}{\varepsilon} \right] \cdot \varepsilon \geq 1$$

Which concludes the proof since SCMSA returns the max($E(M) + \varepsilon, 1$) (in case we reach a ideal evaluation prior to the $\frac{T}{\varepsilon}$th iteration).