ADnEV: Cross-Domain Schema Matching using Deep Similarity Matrix Adjustment and Evaluation

[Technical Report]

Roee Shraga, Avigdor Gal
Technion – Israel Institute of Technology
Haifa, Israel
{shraga89@campus.,avigal@}technion.ac.il

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

ABSTRACT

Schema matching is a process that serves in integrating structured and semi-structured data. Being a handy tool in multiple contemporary business and commerce applications, it has been 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 algorithmic 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 algorithmic matchers. ADnEV learns two models that iteratively adjust and evaluate the original similarity matrix. We empirically demonstrate the effectiveness of the proposed algorithmic solution for improving matching results, using real-world benchmark ontology and schema sets. We show that ADnEV can generalize into new domains without the need to learn the domain terminology, thus allowing cross-domain learning. We also show ADnEV to be a powerful tool in handling schemata which matching is particularly challenging. Finally, we show the benefit of using ADnEV in a related integration task of ontology alignment.

PVLDB Reference Format:
Roee Shraga, Avigdor Gal, and Haggai Roitman. ADnEV: Cross-Domain Schema Matching using Deep Similarity Matrix Adjustment and Evaluation [Technical Report]. *PVLDB*, (): xxxx-yyyy, .
DOI:

1. INTRODUCTION

The rapid growth in data source volume, variety, and veracity increases the need of *schema matching*, a data integration task that provides correspondences between concepts describing the meaning of data in various heterogeneous, distributed (structured or semi-structured) data sources. Examples include SQL and XML schemata, entity-relationship diagrams, ontology descriptions, and Web forms [24],[56]. The need arises in a variety of domains including data warehouse loading and exchange, linking datasets and entities for data discovery [27, 40], integrating displays in interactive data analysis [49], aligning ontologies for the Semantic Web [26], and business document format merging (*e.g.*, orders and invoices in e-commerce) [56]. 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 Tokyo in September." Such an app requires integrating and matching several data sources of product purchase orders and airfare Web forms.

Originated in the database community [56], research into schema matching has been ongoing for more than 30 years now, focusing on designing high quality matchers, automatic tools for identifying correspondences among database attributes. It has also been a focus for other disciplines as well, from artificial intelligence [19, 34] to Semantic Web [26] to data mining [30]. Numerous algorithmic attempts were suggested over the years for handling the problem (*e.g.*, COMA [20], Similarity Flooding [48], and BigGorilla [14]). Theoretical grounding for schema matching [12, 22, 28] have shown that schema matching is inherently an uncertain decision making process due to ambiguity and heterogeneity of structure, semantics, and forms of representation of identical concepts. Despite the increased necessity of schema matching, the development of advanced schema matching techniques is stagnating, revisiting existing heuristics that rely on string matching, structure, and instances.

The quality of automatic schema matching outcome is usually assessed using some evaluation metric (*e.g.,* Precision, Recall, F1, *etc.*) Applying such metrics requires human involvement to validate the decisions made by automatic schema matchers [74]. Yet, human validation of schema matching requires domain expertise [23] and may be laborious [74], biased [10], and diverse [60, 66]. This in turn, limits the amount of qualitative labels that can be provided for supervised learning, especially when new domains are introduced. Schema matching predictors [29, 64] have been proposed as alternative evaluators for schema matching outcome, opting to correlate well with evaluation metrics created from human judgment. Previous works have been focused so far on manually designed features and their combination [30]. Furthermore, trying to "adjust" (improve)

Proceedings of the VLDB Endowment, Vol. , No.
ISSN 2150-8097.
DOI:

schema matching outcome, previous works have utilized several human crafted rules and heuristics [14, 21, 27].

In this work, we offer a method to improve the outcome of automatic schema matchers without human support. We do so by offering a novel post processing step based on deep learning. Training data for supervised learning is created from existing reference models, while no human involvement is required during the matching process itself. Furthermore, the proposed method performs cross-domain matching effectively, learning using whatever domains are available and still performing well on new domains, without any need to inject domain-specific matching information.

The proposed method uses a novel mechanism of *similarity matrix adjustment* to automatically calibrate a matching result, conceptualized in a similarity matrix. Our tool of choice makes use of *deep neural networks*, providing a data-driven approach for extracting hidden representative features for an automatic schema matching process, removing the requirement for manual feature engineering. To this end, we first learn two conjoint neural network models for adjusting and evaluating a similarity matrix. We then propose the ADnEV algorithm, which applies these models to iteratively adjust and evaluate new similarity matrices, created by state-of-the-art matchers. With such a tool at hand, we enhance the ability to introduce new data sources to existing systems without the need to rely on either domain experts (knowledgeable of the domain but less so on the best matchers to use) or data integration specialists (who lack sufficient domain knowledge). Having a trained ADnEV model also supports systems where human final judgement is needed by regulation, *e.g.,* healthcare, by offering an improved matching recommendation. Our contribution is therefore threefold:

- A novel framework for schema matching using automatic *similarity matrix adjustment* and *evaluation* with performance guarantees (Section 2).
- A learning methodology using similarity matrices based on deep neural networks and an algorithm (ADnEV) to improve the matching outcome (Section 3).
- A large-scale empirical evaluation, using real-world benchmark ontology and schema sets, to support the practical effectiveness of the proposed algorithmic solution for improving matching results. In particular, we show that the ADnEV algorithm has strong cross-domain capabilities, can improve performance of a challenging matching problem, and serves in solving a related problem of ontology alignment (Section 4).

We conclude the paper with related work (Section 5) and concluding remarks (Section 6).

2. MODEL

We position the schema matching task as a similarity matrix adjustment process. By doing so, we enable a view of schema matching as a machine learning task, leading to a natural formulation using deep learning models (Section 3). This in turn, supports the significant improvement in our ability to correctly match real-world schemata (Section 4). We begin by introducing a schema matching model (Section 2.1), followed by a definition of two post processing steps (evaluation and adjustment) for a matching outcome. Similarity matrix evaluation (Section 2.2) assesses a matching outcome in the absence of a reference match while similarity matrix adjustment (Section 2.3) calibrates a matching result based on the evaluation. We conclude the section with a formal specification of the relationships between the two post processes (Definition 2). Throughout, we shall use the following illustrative example.

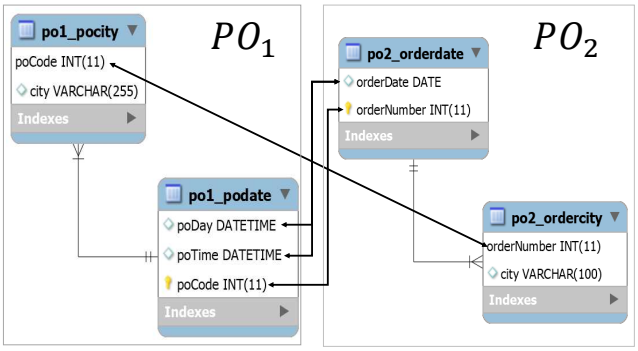


Figure 1: Schema Matching example

EXAMPLE 1. *Figure 1 presents two simplified purchase order schemata [20]. PO_1 has four attributes (foreign keys are ignored for simplicity): purchase order's number (*poCode*), timestamp (*poDay *and* poTime*) and shipment city (*city*). PO_2 has three attributes: order issuing date (*orderDate*), order number (*orderNumber*), and shipment city (*city*). A matching process aims to match the schemata attributes, where a match is given by double-arrow edges.*

2.1 Schema Matching Model

The presented schema matching model is mainly based on [28]. Let S, S' be two schemata with the unordered sets 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 is conceptualized as a similarity matrix, denoted hereinafter $M(S, S')$ (or simply M), with M_{ij} (typically a real number in $[0, 1]$) representing similarity of $a_i \in S$ and $b_j \in S'$. Matrix M is defined as *binary* if for all $1 \leq i \leq n$ and $1 \leq j \leq m$, $M_{ij} \in \{0, 1\}$. $\mathcal{M} \subseteq [0, 1]^{n \times m}$ is the set of all possible similarity matrices. A *match* between S and S' comprises all M's non-zero entries.

Let $f(M)$ denote a schema pair similarity function, assigning an overall value to a similarity matrix, M. Typically, these functions are additive, *e.g.*, $f(M) = \sum_{i=1}^{n} \sum_{j=1}^{m} M_{ij}$.

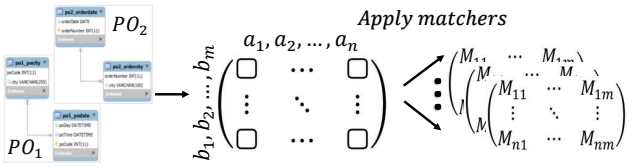


Figure 2: Similarity matrix generation

EXAMPLE 1 (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 *[28], a string-based matcher. The projected match includes all correspondences besides* {(city, orderNumber), (poCode, city)} *with* $f(M) = 2.19$.

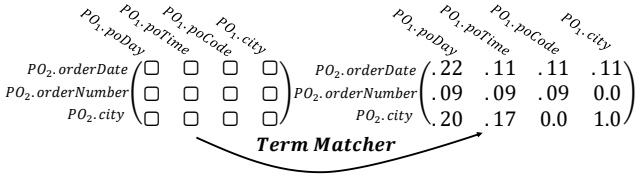


Figure 3: Similarity matrix example

2.2 Similarity Matrix Evaluation

Let M^e 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, $E_{M^e} : \mathcal{M} \rightarrow [0, 1]$, 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_{M^e} simply as E.

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

$$P(M) = \frac{\mid M^+ \cap M^{e+} \mid}{\mid M^+ \mid}, \quad (1) \qquad R(M) = \frac{\mid M^+ \cap M^{e+} \mid}{\mid M^{e+} \mid}, \quad (2)$$

where M^{e+} and M^+ represent the non-zero entries of M^e and M, respectively. Using $P(M), R(M)$, the F1 measure, $F(M)$, is calculated as their harmonic mean.

Sagi and Gal proposed methods for evaluating non-binary similarity matrices using a matrix-to-vector transformation of a matrix M into an $(n \times m)$ size vector, given by $v(M)$ [65]. Such a representation becomes handy when introducing a similarity matrix as input to a Recurrent Neural Network (RNN), capturing the memory needed when comparing the value of a specific entry in the similarity matrix to those of its preceding neighbors (Section 3.1). Using such a transformation, cosine similarity with respect to a reference match is defined as follows:

$$Cos(M) = \frac{v(M) \cdot v(M^e)}{||v(M)|| \cdot ||v(M^e)||} \quad (3)$$

Continuing Example 1, let {(poDay, orderDate), (poTime, orderDate), (poCode, orderNumber), (city, city)} be the reference match. Then, we have $P(M)$= .40, $R(M)$= 1.0, $F(M)$= .57. A vector representation of Figure 3 similarity matrix: (.22, .11, .11, .11, .09, .09, .09, .00, .20, .17, .00, 1.0), yields $Cos(M)$ = .65.

It is noteworthy that even a matcher's best match may not align perfectly with a reference match and match evaluation depends by-and-large on the preferred evaluation measure. For example, in Figure 3, since the first three attributes in PO_1 are equally likely to match orderNumber, a matcher may include all of them to gain maximum recall, by covering all likely-to-be correct correspondences and include none to avoid losing in precision.

In many real-world scenarios, there is no reference match against which evaluation is performed. A *predictor* $\hat{E} : \mathcal{M} \rightarrow [0, 1]$ uses human intuition to evaluate a match in the absence of a reference match [64]. Generally, matching predictors operate in an unsupervised manner, except an initial attempt to supervisedly learn a predictor [30]. We characterize next a monotonic similarity matrix evaluator (predictor), which "behaves" approximately the same as the evaluation function it aims to estimate.

Definition 1. *Let E be an evaluation function and $\hat{E}$ a predictor. $\hat{E}$ is* monotonic *w.r.t. E if for any two similarity matrices $M, M' \in \mathcal{M}$,*

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

2.3 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 $SMA : \mathcal{M} \rightarrow \mathcal{M}$, 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) [28]. 2LMs may come in two flavors, namely *decisive* or *indecisive*. The former manipulates the similarity matrix to determine which entries remain non-zero (and hence part of a match). The latter is meant to improve the matrix using some heuristic reasoning. Indecisive 2LMs are typically used in tasks such as pay-as-you-go schema matching [18].

Example 2. *We present five SMAs, based on known 2LMs.* Threshold(ν) *and* Max-Delta(δ) *[20] apply selection rules, eliminating background noise in a similarity matrix.* Threshold(ν) *keeps entries (i, j) having $M_{ij} \geq \nu$ while nullifying others.* Max-Delta(δ) *selects entries that satisfy $M_{ij} + \delta \geq \max\{\max_{j'} M_{ij'}, \max_{i'} M_{i'j}\}$.* Maximum weighted bipartite graph match (MWBG) *[31] and* stable marriage (SM) *[46] use well-known matching algorithms, given a score or ordering over elements.* Dominants *[28] selects correspondences that dominate (*i.e., *have the maximal similarity value) all entries in their row and column.*

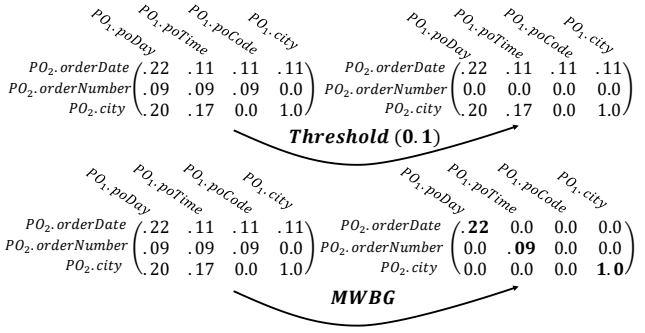


Figure 4: Similarity matrix adjustment example

Figure 4 provides two examples of SMAs over the similarity matrix of Figure 3. Threshold(0.1) *reduces noise by nullifying the matrix row of* orderNumber *with* $f(M)$=1.92. MWBG *selects for each attribute in PO_2 the most similar attribute from PO_1, while satisfying a $1:1$ matching constraint. The result is the match* {(poDay, orderDate), (poTime, orderNumber), (city, city)} *with* $f(M)$=1.31. *Recalling the reference match* {(poDay, orderDate), (poTime, orderDate), (poCode, orderNumber), (city, city)} *and dubbing the corresponding matrices of* Threshold *and* MWBG *as M^t and M^m, respectively, we have $P(M^t)$= .43, $R(M^t)$= .75, $F(M^t)$= .55 and $P(M^m)$= .67, $R(M^m)$= .50, $F(M^m)$= .57.*

The similarity matrix abstraction captures rich structures (including taxonomies, ontologies, and others, see [28] for details) that can be employed as part of an adjustment process. Typically, however, SMAs in the literature are limited in the way they adjust a similarity matrix, using some human-guided rules, *e.g.* 1 : 1 matching as in MWBG and SM or limiting the space of the output matrix values to the original similarity values (*e.g.*, Threshold, Max-Delta, and Dominants). Contemporary SMAs are also limited in that they do not take into account the evaluation measure (see Section 2.2). We hypothesize that a similarity matrix contains more than meets the eye, with hidden information that is not captured by human designed rules. In addition, to support evaluation-conscious SMAs we aim at *consistent similarity matrix adjustment*, with respect to an evaluation measure E, as follows.

DEFINITION 2. *Let E be an evaluation function and $\varepsilon > 0$ an improvement factor. A similarity matrix adjustment mapping SMA is* consistent *w.r.t. E if for any similarity matrix $M \in \mathcal{M}$, it holds that:* $\min(E(M) + \varepsilon, 1) \leq E(SMA(M))$.

A consistent SMA ($CSMA$) assures quality improvement of the original matrix with respect to an evaluation function. By Definition 2, $CSMA$ can identify perfect matches, given sufficient time to improve. In what follows, predictors (Section 2.2) can be used to assess our ability to adjust similarity matrices, which will be a main component of the proposed algorithm (Section 3.3).

3. DEEP SIMILARITY MATRIX ADJUSTMENT AND EVALUATION

Equipped with predictors and understanding their possible role in improving similarity matrices we are now ready to introduce our approach for deep similarity matrix adjustment and evaluation. Our tools of choice are deep neural networks (DNNs, see Section 3.1). The advantage of using DNN models is that we no longer need to hand-craft features when designing adjustors, nor do we need to decide a-priori which predictors would work well for a specific domain. DNNs assist in capturing complex relationships among similarity matrix elements, as detailed in Section 3.2. The ADnEV algorithm (Section 3.3) utilizes two interacting DNNs (adjustor and evaluator), incrementally modifying an input similarity matrix. We provide an illustrative example of the algorithm and a methodology to utilize ADnEV when the input consists of multiple matrices (Section 3.4).

3.1 Neural Networks for Similarity Matrices

Unlike most state-of-the-art SMAs operate, DNNs were shown to capture relationships in structured data automatically [33, 44]. The basic idea of deep learning is to perform non-linear transformations of an input data (using activation functions, *e.g., Sigmoid* or *ReLU* [32]) to produce an output. Specifically, in this work, we use two DNN types, namely Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). CNN is rooted in processing grid-like topology data [43], with applications in Image and Video Processing [42], Recommender Systems [68], *etc.* RNN is rooted in processing sequential data [63], with applications in Natural Language Processing [15], Time Series Prediction [17], and Entity Resolution [37, 50]. Next, we focus on the application of DNNs to similarity matrices. A broader description of the foundations of DNNs is given in Appendix A and may be found in [33, 44].

We utilize both CNN and RNN to capture (hidden) data patterns in similarity matrices. CNNs can identify spatial patterns between correspondences by learning matrix features using small subareas of the input data (convolutional and pooling layers). Several SMAs in the literature are heuristically designed with manually-crafted feature extraction rules that capture grid-like dependencies within the matrix. Recalling Example 2, both Dominants and Max-Delta(δ) essentially use a max-pooling approach when they make a match decision based on a correspondence respective row or column. We believe that CNNs can learn such spatial dependencies automatically.

RNNs can sequentially process the similarity matrix, allowing the model to make a decision regarding a single correspondence based on previous selections, and therefore, in a sense, imitating a human-like sequential schema matching decision process. Finally, by combining both CNN and RNN (into CRNN), we exploit the benefits of both DNN types (an empirically validated choice, see Section 4.3).

3.2 Learning to Adjust and Evaluate Similarity Matrices

The input for an adjustor and an evaluator is a set of K similarity matrices $M^{(K)} = \{M_k\}_{k=1}^{K}$. An adjustor applies an SMA (Section 2.3), returning a matrix in $\mathcal{M}$ (space of similarity matrices) and an evaluator returns a value in $[0, 1]$.

In this paper we adopt a supervised learning methodology. As such, we assume that (only) during training the input similarity matrices $M^{(K)}$ are available with their respective reference matrices $\{M_k^e\}_{k=1}^{K}$. Reference matrices were created over years with the assistance of multiple domain experts. They are only used during training, enabling a "human-less" matching process after a model is trained. We note that while it is not realistic to have a reference match for each domain, it is reasonable to assume that some human labeled data exists (in our case using existing benchmarks, see Section 4.1) to learn a model that can be applied in scenarios where no reference match exists. We show empirically (see Section 4.5.2) that the ADnEV algorithm (Section 3.3) provides effective cross-domain learning.

The adjustor, AD, receives similarity matrices $\{M_{ij} \in [0, 1]\}_{i=1,j=1}^{n,m}$ as training examples ($M \in \mathcal{M}$) and their respective labels: $l(M_{ij}) = 1$ if $M_{ij} \in M^e$; otherwise, $l(M_{ij}) = 0$. Spatial and sequential relationships among similarity matrix entries, as captured by the network, essentially allowing each entry to be represented not only by its own value but also by its *context* within the similarity matrix. Adjustors solve a binary classification problem for each entry (independently of other entries), trained using a binary cross entropy (CE) loss defined over each train matrix M:

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

where $\hat{M}_{ij}$ denotes the predicted value of $l(M_{ij})$. Note that $\hat{M}_{ij}$ basically represents the probability of $l(M_{ij})$ to be assigned with a value of 1, and its value is therefore in $[0, 1]$.

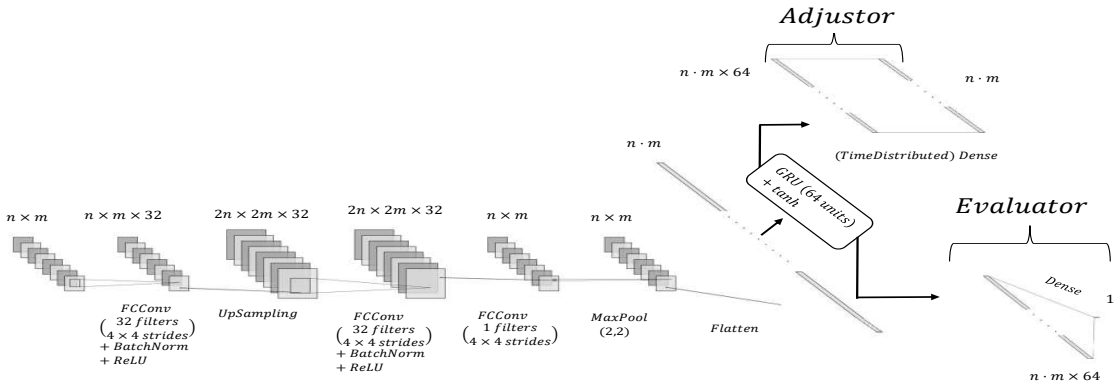


Figure 5: Adjust and Evaluate network architecture.

The evaluator, EV, also receives similarity matrices $\{M_{ij} \in [0,1]\}_{i=1,j=1}^{n,m}$ as training examples ($M \in \mathcal{M}$) together with their overall evaluation function value, $E(M)$, calculated using a reference match (see Section 2.2). The evaluator solves a regression problem, tuned using a mean squared error (MSE) loss computed, given $M^{(K)}$, as follows:

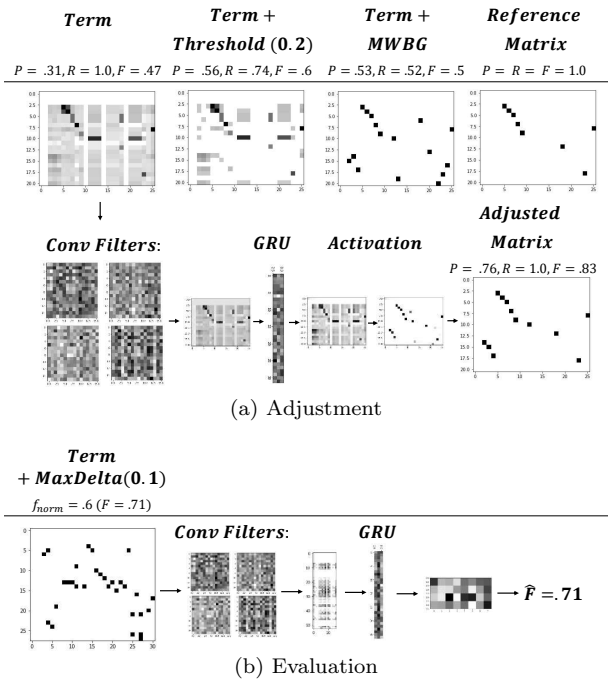
$$MSE(M^{(K)}) = \frac{1}{K}\sum_{k=1}^{K}\left(\hat{E}(M_k) - E(M_k)\right)^2, \qquad (5)$$

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

As a design of the overall neural network, we suggest the use of a *multi-task NN* [13, 62] with two objectives, one corresponds to SMA (AD) and the other corresponds to matrix evaluation (EV). This allows the network to learn a joint representation of the input matrix while updating the weights with respect to both objectives. The novelty of the proposed network architecture is in making the SMA *evaluation-aware* (and vice versa). Multi-task NN linearly combines both models via a joint loss based on CE (Eq. 4) and MSE (Eq. 5) using uniform weights.

Figure 5 illustrates the network architecture using both convolutional and recurrent layers (denoted as CRNN). To handle variable size input matrices, we begin with a fully convolutional network (FCN), following [45], which is independent of the input shape and does not affect weights and biases of layers (noticing input shape is larger than the filter size). For readability, $n \times m$ in Figure 5 represents variable size input matrices.

Convolutional and pooling layers can be fine-tuned using hyperparameters such as filtering size and the strides in which the number of neighboring cells are considered, as illustrated in Figure 5. Each of the two first convolutional layers are followed by batch normalization[1] and ReLU activation[2]. Between these layers we perform upsampling[3] to increase resolution and reduce noise. Then, we scale back to the input matrix dimension using a max-pooling layer that semantically merges back the upsampled surroundings of each entry. The convolution part of the network helps to represent each correspondence by its context in the matrix, similar to the way pixels are represented in image-to-image frameworks, *e.g.,* for image super-resolution [71].



(a) Adjustment

(b) Evaluation

Figure 6: AD and EV Illustration over Web-forms Schema Pairs (see Section 4.1). Entries are marked by a gray scale color map.[4]

Next, we flatten the output and feed it into the recurrent part of the network. Following [72], we use a Gated Recurrent Unit (GRU)-based RNN [15]. GRU uses a reset gate to decide how much of past information should be disregarded, and an update gate to decide how much of past states information should pass through. Finally, we obtain a 64-dimensional context vector for each correspondence that can be used for independent classification (adjustor, top part of the final layer) as well as for overall regression based on the representations of all the entries (evaluator, bottom part of the final layer).

We empirically validated that CRNN performs better than GRU and CNN independently. Dropout did not improve

[1]batch normalization is a technique for improving the stability of a neural network by normalizing its layers [36]

[2]ReLU is a widely used activation function defined as follows; $f(x) = max(0, x)$ [32]

[3]upsampling simply repeats entries in the input in order to change its dimension [71]

results, thus not included in the networks. We have also trained standalone versions for AD and EV, by omitting the bottom and top two layers of the network respectively, to assess the value of a multi-task network (see Section 4.3).

EXAMPLE 3. *Figure 6 presents a gray scale color map*[4] *of adjusting and evaluating similarity matrices from the* Web-forms *dataset (see Section 4.1). It visualizes the two parts of the network architecture (Figure 5) using trained layers with respect to a dataset on an input similarity matrix.*

Figure 6a illustrates the adjustment process, the convolutional filters highlight prominent areas in the matrix which is than fused using max pooling. Then, the GRU injects sequential knowledge, nullifying the left and top parts of the matrix and the activation projects the final matrix. Figure 6b demonstrates how the network managed to predict accurately the F1 value of an input matrix ($\hat{F} = F = 0.71$).

3.3 Iterative Adjust & Evaluate (ADnEV)

We now suggest an iterative Adjust and Evaluate (ADnEV) algorithm. In addition to an evaluation-aware adjustor training, using a hill-climbing approach, the ADnEV algorithm attempts to employ the best out of an input similarity matrix by performing iterative adjustments. Also, we show that the algorithm converges to the optimal matrix given well-behaved adjustors (Definition 2) and evaluators (Definition 1).

Algorithm 1 The *ADnEV* algorithm

1: **Input:** Similarity matrix M, trained adjustor AD, and trained evaluator EV
2: **Output:** An Adjusted Similarity Matrix
3: $M^0 \leftarrow M$, $\hat{E}(M^0) \leftarrow EV(M)$, $t \leftarrow 0$
4: **do**
5: $\quad t \leftarrow t + 1$
6: $\quad M^t \leftarrow AD(M^{t-1})$
7: $\quad \hat{E}(M^t) \leftarrow EV(M^t)$
8: **while** $\hat{E}(M^t) > \hat{E}(M^{t-1})$
9: **Return:** M^{t-1}

The algorithm uses pre-trained adjustor AD and evaluator EV as described in Section 3.2. Given an input similarity matrix, AD returns an adjusted similarity matrix and EV returns an evaluation value. Figure 7 illustrates the main idea of the ADnEV algorithm and the pseudocode is given in Algorithm 1. ADnEV begins with a new unseen similarity matrix (test phase) (line 3) and readjusts it (line 6) as its evaluation value continues to improve. 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 (line 8).

The adjustment process aims to improve the matching outcome, thus the algorithm should not return matching results that are inferior to the initial match. The main component of the algorithm in charge of assessing the matching outcome is the evaluator. While it is immediate from the stopping condition (line 8) that the predicted evaluation value of the output similarty matrix will be at least as good as the inputs, ADnEV *dominance* (Proposition 3.1) assures that given a well-behaved evaluator the true evaluation will behave the same. Using a well-behaved adjustor,

[4] 0 and 1 entries are colored white and black, respectively. Entries closer to 1 are darker than those closer to 0.

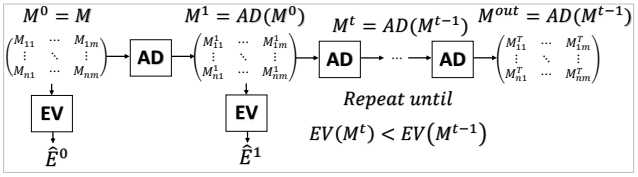


Figure 7: ADnEV algorithm illustrated

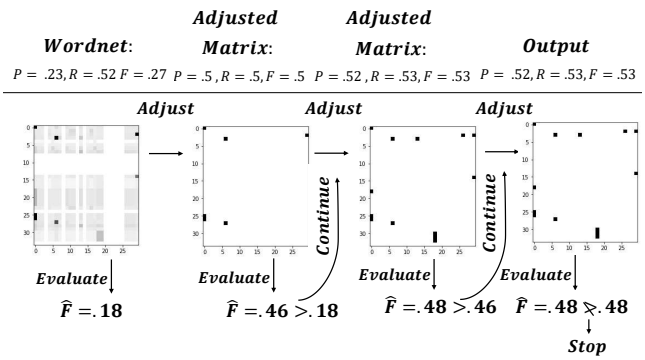


Figure 8: ADnEV Illustrative Example over a Web-forms Schema Pair (see Section 4.1). Entries are marked by a gray scale color map[4]

ADnEV is also guaranteed to *converge* (Proposition 3.2) to an ideal matrix with respect to an evaluation function.

PROPOSITION 3.1 (ADnEV DOMINANCE). *Let $EV(\cdot)$ be a monotonic evaluator (Definition 1) and let M and M^* be the input and output matrices of* ADnEV*, respectively. Then, $E(M^*) \geq E(M)$.*[†]

PROPOSITION 3.2 (ADnEV CONVERGENCE). *Let $AD(\cdot)$ be a CSMA (Definition 2) with $0 < \varepsilon \leq 1$, let $EV(\cdot)$ be a monotonic evaluator (Definition 1), and let M and M^* be the input and output matrices of* ADnEV*, respectively.* ADnEV *returns a matrix with an ideal evaluation ($E(M^*) = 1$) in less than $\lceil \frac{1}{\varepsilon} \rceil$ steps.*[†]

To intuitively explain the role of monotonic evaluation, recall that ADnEV iterates until its halting condition is met. At each iteration t the adjusted matrix is estimated to be better than its predecessor ($\hat{E}(M^t) > \hat{E}(M^{t-1})$) and thus better in terms of the true evaluation ($E(M^t) > E(M^{t-1})$) and the algorithm converges to an optimal output similarity matrix.

EXAMPLE 3 (CONT.). *Figure 8, provides a gray scale color map*[4] *example of using* ADnEV *over a challenging schema pair to obtain the output similarity matrix. The input matrix has a fairly low precision due to many non-zero entries. After applying an adjustment step, most low-similarity entries (light gray) are removed from the matrix, boosting precision and slightly reducing recall. As the estimated value is improved the algorithm continues with another adjustment, which spots 7 additional correspondences, improving all quality measures. Finally, the algorithm terminates as the next adjustment fails to improve the predicted quality of the matrix.*

[†]Proposition proofs are given in Appendix B

3.4 ADnEV with Multiple Matrices

Until now, we treated ADnEV as a single-input-single-output algorithm. However, having a trained adjustor and a trained evaluator opens the door for additional alternatives. Explicitly, let $M(S,S')^{(K)}$ be a set of K similarity matrices, generated with respect to a specific schema pair S, S'. Using ADnEV, we can obtain an improved set, $M(S,S')^{(K)*}$, where each matrix $M^* \in M(S,S')^{(K)*}$ is an adaptation of an initial input matrix. Accordingly, we have gathered a set of $2K$ matrices addressing the schema pair S, S'. Next, having a trained evaluator EV, we can select the best matrix for this pair aiming at the best possible outcome, *i.e.,*

$$\underset{M \in M(S,S')^{(K)} \cup M(S,S')^{(K)*}}{\operatorname{argmax}} EV(M).$$

This approach corresponds to earlier works on matcher ensembles [20]. A matcher ensemble considers multiple matchers input as it computes a weight for each one (which can be learned, *e.g.,* using a boosting method [47]) to create a matching decision. Our method, instead, learns a model based on an individual similarity matrix, which is the focus for this work. The learned models can be applied to select a top similarity matrix out of a set. We leave the design of ensemble deep framework for future work.

4. EMPIRICAL EVALUATION

We now present an empirical evaluation of the ADnEV algorithm, evaluating each component (adjustor and evaluator) separately and jointly using real-world datasets. We establish the applicability of the proposed algorithm by experimenting with a variety of application datasets, from small scale Web form schemata, extracted from diverse domains such as flights, dating, booking, *etc.* to large purchase orders used by retailers, to bibliographic references ontologies from various resources. Code repository is available. [1] In our empirical analysis we demonstrate the following three properties of our proposed algorithm:

- **Cross Domain:** ADnEV generalizes beyond a single domain. Using a network that was pre-trained on one domain (*e.g.*, flights), the algorithm can significantly improve the performance of matchers, applied to a different domain (*e.g.*, job finding).
- **Challenging Matches:** ADnEV significantly improves the performance of state-of-the-art matchers when applied to a large, particularly hard to solve benchmark (purchase orders).
- **Agility:** ADnEV is shown to perform well on varying data integration tasks, including ontology alignment, making use of either a single or multiple matchers simultaneously.

4.1 Datasets

Table 1 provides details of the datasets used in the experiments. All datasets offer a schema matching challenge. In addition, each dataset allows us to investigate a different facet of the proposed algorithm, aiming to illustrate its variability. The Purchase Order [20] dataset represents a classic large-scale schema matching problem between XML documents of purchase orders, extracted from various systems. This dataset has been a focus of interest to multiple schema matching research, including [10, 11, 21, 30, 67]. The OAEI dataset represents a slightly different matching problem, where instead of matching schema attributes, we aim to match ontology elements. OAEI contains ontologies from the bibliographic references domain, introduced in the OAEI 2011 and 2016 competitions [7], matching 100 ontologies against a reference ontology. Finally, the Web-forms [28] dataset represents a cross-domain matching problem and contains schemata of varying sizes that were automatically extracted from Web forms of diverse domains. Web forms present an ongoing matching challenge to Web interface integration [24, 70]. Cross domain Web form matching was also a focus for holistic schema matching, integrating multiple-source schemata rather than pairwise matching [16].

Table 1: Datasets used in the evaluation

Dataset	Purchase Order	OAEI	Web-forms
#Elements	50-400	80-100	10-30
#Pairs	44	100	147
#Augs	10	5	4
#Matrices	17,424	21,600	26,460
#App. Domains	1	1	18
Task Type	large-scale schema matching	ontology matching	cross-domain schema matching

The three benchmark datasets represent 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 of matchers. Reference matches for these dataset were manually constructed by domain experts over the years and considered as ground truth for our purposes.

4.2 Generating Similarity Matrices

Next we describe how we generate an initial set of similarity matrices as input for our algorithm. First, we describe how we derive similarity matrices from state-of-the-art schema matchers (Section 4.2.1). Then, we describe an augmentation methodology to expand a given (training) set of similarity matrices inspired by the effectiveness of data augmentation in computer vision, see for example, Perez and Wang [53] (Section 4.2.2).

4.2.1 Schema Matchers

We used three matchers (Term, WordNet, and Token Path) to generate our baseline set of similarity matrices. The first two, Term and WordNet, are implemented in the OntoBuilder Research Environment [8] (ORE) – a research prototype for large scale matching experiments. Term [28] compares attribute names to identify syntactically similar attributes (*e.g.*, using edit distance and soundex). WordNet [59] uses abbreviation expansion and tokenization methods to generate a set of related words for matching attribute names. The third matcher, Token Path, is based on SAP's Auto-Mapping Core (AMC) [54], which provides an infrastructure and a set of algorithms to match business schemata. Token Path integrates node-wise similarity with structural information by comparing the syntactic similarity of full root-to-node paths. We used ORE's embedded implementation of AMC's Token Path algorithm.

Each matcher's similarity matrix is used in its original form. In addition, we apply to the similarity matrix the five state-of-the-art SMAs that were presented in Section 2.3. Among those SMAs, both Threshold(ν) and Max-Delta(δ) are based on COMA [3], a state-of-the-art schema matching research tool, whose details are given in Example 2. Following [29, 30] we set $\nu = 0.5$ and $\delta = 0.1$, respectively.

4.2.2 Similarity Matrix Augmentation

We created 18 similarity matrices per schema pair,[3] 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, which enables increasing the train set size, 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 $a \in \{1, \dots A\}$, the following operations are applied:

- Randomly select i, j from $\{1, 2, ..., n\}$ and t, k from $\{1, 2, ..., m\}$
- Replace the i'th row with the j'th row and the t'th column with the k'th column in both M and M^T.
- Add the augmented versions to the training set.

To enhance reproducibility, an illustrative example is given in Figure 9 using the similarity matrix of Example 2.1. First, M^T is generated (top right). Then, for each matrix, we randomly select two rows and columns to swap (bottom).

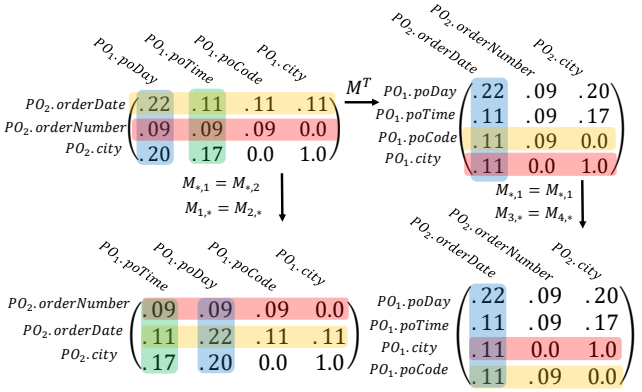


Figure 9: Similarity matrix augmentation example. Swapped rows/columns are colored respectively.

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 Experimental Setup

We now turn our attention to specifying hardware, implementation, methodology, and baselines.

4.3.1 Hardware and Implementation

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 gtx 2080 Ti, a second server contains two Nvidia GeForce gtx 1080 GPUs, while the others contain a single NVIDIA Tesla K80 GPU. The 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.

All networks were implemented using Keras [6] with a tensorflow backend and are available online [1]. The Adam optimizer [38] was used, with a learning rate of 0.001 and $\beta_1 = 0.9, \beta_2 = 0.999$, following [61]. We also trained separately CNN, RNN and a simple fully connected DNN, which gave inferior results. Hence, due to space considerations, we only focus on our suggested CRNN architecture (see Section 3.1). Dropout did not improve results, thus not used. We fed the network with each matrix independently ($batchsize = 1$), going over the dataset (including the augmented versions, see Section 4.2.1) only once ($epochs = 1$).

[3]Using 3 matchers without an SMA, and also applying each of the 5 SMAs.

4.3.2 Methodology

The experiments are performed per dataset. We use k-fold cross validation for evaluation. For Purchase Order and OAEI, we randomly split the schema pairs in each dataset into 5 folds and repeat an experiment 5 times, using 4 folds for training and the remainder fold for testing. For the Web-forms dataset, we split the schema pairs by domain, having distinct domains in each fold.

For each experiment, we first generate an initial training set of similarity matrices (Section 4.2.1) and expand this set using augmentation (Section 4.2.2). Then, we train the $CRNN$-based adjustor and the $CRNN$-based evaluators as described in Section 4.3. For each evaluation measure E, we train a different evaluator, which, in turn, represent four versions of ADnEV$_E$ (displayed in Table 2). Then, we use the ADnEV algorithm (Section 3.3) to adjust and evaluate the test set of similarity matrices (generated from the remaining fold of schema pairs). Model training of each epoch took ≈45 minutes on two Nvidia gtx 2080 Ti GPUs. Applying ADnEV during the test phase takes ∼2.5 seconds per matrix.

In addition, to enhance the learning ability of our model, we include the reference matrix as an example **during training only**. A reference matrix obtains a perfect score with respect to both adjustor (all labels are correct by definition) and evaluator (all evaluation functions will return a value of 1 given the reference matrix). Reference matrices have been complied over the years for the datasets used in our experiments and serve as a desiderata for the best similarity matrix of a given schema pair. This, in turn, assists in quickly "homing" on the appropriate model weights during training and allows the trained model to represent reality better. We emphasize here that reference matches are **not** included during the test phase.

Statistically significant differences in performance are marked with an asterisk using a paired two-tailed t-test with Bonferroni correction for 95% confidence level.

4.3.3 Baselines

We have tested our methods against several baselines. We provide next details on baselines for adjustors, evaluators and neural networks. All baselines are available online [1].

- **Adjustors** are compared to the similarity matrices that were generated by state-of-the-art schema matchers and employed as an initial input to the ADnEV algorithm (due to space considerations we aggregatively refer to them as $Orig$). In addition, we observe a resemblance of the adjustment problem to matrix factorization (MF) [41, 57, 58]. Using MF terminology, the original matrix M is the "user-item" matrix that represents the implicit "ratings" to be adjusted by MF. We use the following state-of-the-art MF methods as baselines: 1) SVD^{++} [9, 41], using singular value decomposition to learn latent representations of $a_i \in S$, $b_j \in S'$ and adjust a similarity matrix by multiplying

Table 2: Precision (P), Recall (R), F1 (F) of ADnEV with single input (a) and multiple inputs (b).

		Orig	*BPR*	*SVD++*	*FM*	*IR*	*DNN*	*CRNN AD*	*CRNN* **ADnEV**$_F$	*CRNN* **ADnEV**$_P$	*CRNN* **ADnEV**$_R$	*CRNN* **ADnEV**$_{Cos}$
Purchase Order	P	.44	.41	.50	.51	.49	.62	.64	.65	**.66**	.65	.64
	R	.39	.43	.50	.50	.50	**.69**	.66	.64	.65	.66	.59
	F	.35	.42	.49	.50	.49	.63	**.65**	**.65**	**.65**	**.65**	.58
Web-forms	P	.57	.41	.50	.51	.50	.54	.57	**.75***	.73*	.61	**.75***
	R	.51	.42	.50	.52	.50	.54	.55	.65	.61	**.68***	.65
	F	.48	.43	.49	.51	.50	.52	.55	.64	**.65***	**.65***	.64
OAEI	P	.50	.42	.51	.54	.50	.59	.60	.71	.71	.68	**.72***
	R	.44	.39	.50	.53	.51	.59	.60	.71	.69*	**.72***	.70
	F	.44	.32	.51	.53	.51	.54	.58	**.67***	.62	.65	.66

(a) Single Input ADnEV vs. Baseline Methods (Section 3.3)

		best (Orig)	*Maxf*	*MaxLRSM*	**ADnEV**$_F$	**ADnEV**$_P$	**ADnEV**$_R$	**ADnEV**$_{Cos}$	*best (Adjusted)*
Purchase Order	P	.56	.52	.56	.66*	**.80***	.68*	.68*	.92
	R	.81	.52	.54	.67*	**.77***	.72*	.68*	.88
	F	.63	.52	.53	.66*	**.78***	.70*	.65*	.90
Web-forms	P	.74	.50	.69*	.79*	.79*	.67*	**.81***	.81
	R	.86	.51	.78*	.66*	.69*	**.79***	.69*	.87
	F	.75	.50	.69*	.69*	.70*	.70*	**.72***	.78
OAEI	P	.76	.56	.59	.72*	**.78***	.73*	.75*	.83
	R	.74	.62	.60	**.82***	.76*	.77*	.77*	.86
	F	.75	.57	.59	.72*	**.76***	.72*	.74*	.85

(b) Multiple Inputs ADnEV vs. Baseline Methods (see Section 3.4). Best (*Orig*) and best (*Adjusted*) represent the best matrix per schema pair chosen in hindsight out of the original set and the improved set, respectively.

the representations per matrix entry; 2) *Bayesian Personalized Ranking* (BPR) [2, 58], which 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) [4, 57], which captures high order non-linear interactions; and 4) *Isotonic Regression* (IR), [5, 51] which learns a calibration function to order corresponding attribute pairs before non-corresponding ones.

- **Evaluators** were compared to $f(M)$ (see Section 2.1) and to a feature-based evaluation, where matching predictors [64] are used as features to learn an evaluation model [30] ($LRSM$). To this end, we used the state-of-the-art predictors proposed in [30], combined in $LRSM$ using a LambdaMART model. Baseline evaluators were also used in a multiple matrices input setting, where $f(M)$ ($Maxf$) and $LRSM$ ($MaxLRSM$) are used to select the best performing matrix. We further tested the following alternative regressors[4]: *Linear Regression*, *Lasso Regression*, *Ridge Regression*, *Theil-Sen Regression*, *Passive Aggressive Regression*, *SGD Regression*, and *Support Vector Regression* (SVR) [25]. We present the results of SVR, whose performance was superior.
- **Vanilla deep neural network (DNN)** was used as a baseline to our proposed deep neural network architecture $CRNN$. DNN is a fully connected network over the original similarity matrix values.

4.4 ADnEV with Single/Multiple Matcher Input

Table 2 reports on the average performance of ADnEV over all datasets, in terms of precision (P), recall (R), and F1 (F). Results of a single similarity matrix input (Section 3.3) are reported in Table 2a, where matching results of ADnEV are compared to a non-iterative adjustment step (applying only AD) and the baselines.Table 2b reports on the use of a set of similarity matrices for each schema pair (in the test set) as described in Section 3.4. Here, we compared our method with two selection baselines, $f(M)$ ($Maxf$) and $LRSM$ ($MaxLRSM$) [30].

ADnEV improves the quality of matching in each of the matching challenges it faced. Overall, a single adjustment step (AD) improves over the original matching ($Orig$) by 22%, 38% and 44% on average in terms of precision, recall, and F1, over all reported datasets, respectively. This validates that using $CRNN$ can improve matching quality. Even when compared to the top SMA (out of $Orig$) in hindsight (using an oracle to choose a matcher that yields the best quality, not shown explicitly in the table) AD achieves better results (*e.g.,* 5% improvement in F1 on average).

AD performs significantly better than BPR, SVD^{++}, FM, and IR, with 55%, 21%, 15%, and 19% better F1 score, respectively. In terms of neural models, the $CRNN$-based AD also performs mostly better than DNN, 6% improvement on average in F1 over all datasets (except the case of recall over the Purchase Order dataset). This indicates that the similarity matrices contain indirect entry dependencies that are captured by the $CRNN$ but not the DNN.

All deep neural models (DNN and $CRNN$) performed better than the original and matrix factorization baselines, demonstrating that similarity matrices feature higher-order (both spatial and sequential) relationships that can be utilized to improve the matching results via neural networks.

ADnEV performs significantly better than a single adjustment step (AD) as it takes into account an evaluation function. ADnEV$_P$ boosts precision by 17%, ADnEV$_R$ boosts recall by 15%, and ADnEV$_F$ boosts F1 by 11% on average over all datasets. When compared to $Orig$, the improvements are substantial, *e.g.,* average absolute gain of .22 in F1. Thus, we can empirically deduce that involving an eval-

[4]Implemented using `https://scikit-learn.org`.

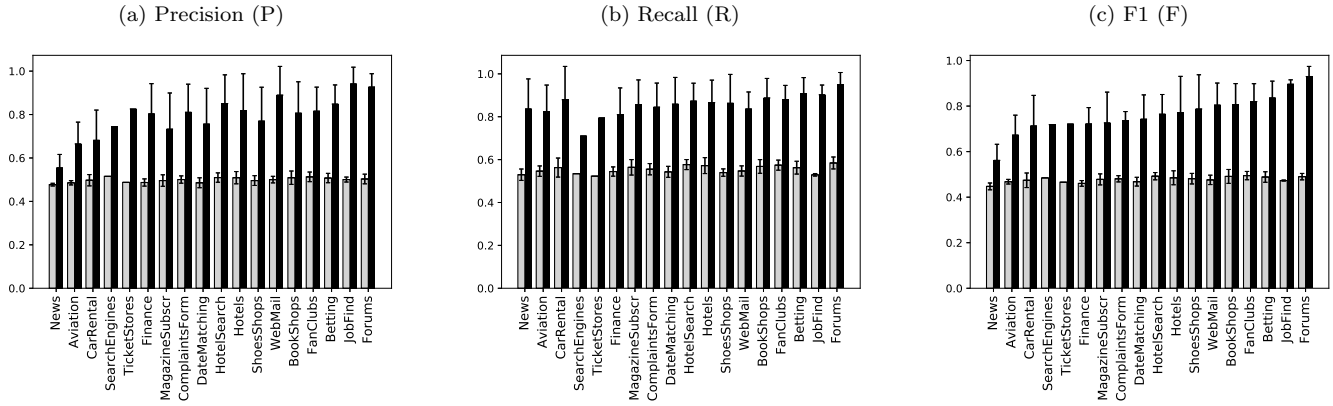


Figure 10: ADnEV$_{Cos}$ performance by domain compared to $Maxf$ (multiple matrices input)

uation (prediction) phase in the adjustment boosts desired performance.

As for the multiple input scenario, analyzing the best matching result for each schema pair (Table 2b), ADnEV creates (adjusts) and detects (evaluates) high quality results. Compared to selecting from the original matching results using $Maxf$, ADnEV improves, on average, matching results by 51% precision (ADnEV$_P$), 39% recall (ADnEV$_R$), and 30% F1 (ADnEV$_F$), over all datasets. When compared to a feature-based approach ($MaxLRSM$), ADnEV fuses higher quality matrices, lifting average absolute values from $P = 0.61, R = 0.64, F = 0.60$ to $P = 0.78, R = 0.74, F = 0.75$ (ADnEV$_P$). This demonstrates again the model's potential to resolve indirect relationships in a similarity matrix and generate better matches.

Analyzing the best similarity matrix in hindsight, we observe that the improved set (best (*Adjusted*)) contains better matrices than the original set (best ($Orig$)). This indicates that ADnEV creates a better pool of similarity matrices to choose from. Our evaluators were able to select high quality matrices, compare to best (*Adjusted*), ADnEV achieves 88% of the former's F1 (on average). However, it also reveals room for improvement when it comes to choosing the best similarity matrix out of a pool.

4.5 ADnEV Algorithm Evaluation

Next, we evaluate ADnEV performance, analyzing each of the three matching challenges separately (see Table 1).

4.5.1 ADnEV *for Large-Scale Schema Matching*

The Purchase Order dataset represents a large-scale matching problem, with similarity matrices reaching $\sim$60,000 entries. The original matching scores ($Orig$) are quite low with $P = 0.44, R = 0.39, F = 0.35$. Being a challenging matching task, several works (*e.g.,* [21]) have focused on this dataset developing domain specific methodologies to deal with its characteristics. Token Path[5] was a matcher that was designed to deal with this dataset, see Section 4.2.1 and improves over the general purpose matchers with $P = 0.55, R = 0.56, F = 0.55$. It is worth noting that for the Web-forms dataset, which introduces a cross-domain challenge (see Section 4.5.2), its performance is unsatisfying ($P = 0.29, R = 0.61, F = 0.38$), suggesting our flexible proposed adjustment methodology to be valuable.

We observe that for the Purchase Order dataset, a single adjustment (AD) is sufficient to achieve a statistically significant improvement over the original matching (Figure 2a). A possible explanation for that may be the structural nature [20] of this dataset, is successfully captured by the neural network. ADnEV did not manage to boost the performance over a single adjustment, indicating that the iterative adjustment may be redundant at times. It is noteworthy that even when adjustment is redundant, the evaluation step restricts quality reduction by rejecting poor adjustments. Still, ADnEV$_{Cos}$ reduce quality of results created by AD, which can be explained by the difficulty of the $CRNN$ evaluator to correlate with cosine (see Section 4.6.3).

When it comes to multiple matrices input, $Maxf$ and $MaxLRSM$ improve over $Orig$ as they consider multiple matching inputs, which manifests in improved quality reaching $P = 0.52, R = 0.52, F = 0.52$ and $P = 0.56, R = 0.54, F = 0.53$, respectively. ADnEV fuses an even higher matching quality, especially when optimizing precision (ADnEV$_P$), obtaining $P = 0.8, R = 0.77, F = 0.78$.

4.5.2 ADnEV *for Cross-Domain Schema Matching*

The Web-forms dataset represents a cross-domain matching problem, with fairly small matrices of $\sim$1,000 entries. Generally, the Web-forms dataset is considered easier to match than the Purchase Order dataset due to its scale, which is indicated by relatively higher original matching ($Orig$) quality. However, an absolute value of .48 F1 score is simply insufficient, especially in a cross-domain setting where acquiring domain expertise may be hard.

For the Web-forms dataset, while a single adjustment step (AD) slightly improved matching results, applying ADnEV significantly improves over these results no matter which quality measure is emphasized. This indicates that in a cross-domain setting it is insufficient to have an initial adjustment with knowledge about a specific domain. Rather, the adjustment process needs to be guided by evaluating (predicting) its performance in an iterative manner.

In a multiple matrices input scenario, we observe first that $Maxf$ performs poorly in choosing the best matrix for precision. Beyond this baseline, we see an increased performance of other methods with respect to all quality measures. Yet, when compared to learning an evaluator using human

[5] Token Path results are aggregated with the other baseline matchers under $Orig$ in Table 2

designed features ($MaxLRSM$), the performance boost is quite similar to the those of ADnEV. This implies that when the input is less sparse (compared to the other datasets) even human designed features capture essential properties of the matrix, particularly with respect to recall. Notably, in this case, the performance benefits from ADnEV$_{Cos}$. Here, capturing the similarity between the derived matrix and the reference matrix, allows selecting higher quality matrices, where for other datasets this strategy fails.

Figure 10 analyzes the quality per domain using a multiple matrices input. Light color represents the performance of $Maxf$ compared to dark color that represents the results of ADnEV$_{Cos}$. We observe variability across domains using ADnEV indicating that transferring a trained model is hard for some domains. Focusing on precision (Figure 10a), we see that the baseline values revolve around .50, indicating that in each domain, some correspondences, *e.g.,* a correspondence such as *E-mail* ↔ *Email*, are easy to identify. However, considering the full context of a match (as captured by a similarity matrix) allows a better adjustment to harder correspondences, *e.g.,* correspondences such as *doa_yy* ↔ *tn_checkin_year* and *Prev. Revenue* ↔ *Prior Year's Sales*, which are challenging and domain-specific. Other works also experimented with matching multi-domain Web forms, *e.g.,* [16, 70]. While in our case the extracted Web forms from the News and Aviation domains were the most challenging, others were challenged by domains of Jobs [16] and Airfare [70]. In the domains of JobFind and Forums, the proposed model obtains ∼90% F1 scores. This, in turn, emphasizes the difficulty variability of Web forms and the ability of ADnEV to handle effectively cross-domain matching.

4.5.3 ADnEV *for Ontology Matching*

The OAEI dataset represents a slightly different challenge. When it comes to aligning ontologies, some basic characteristics are relatively different having, for example, ontology elements rather than schemata attributes (see Section 5). ADnEV was successful at improving over state-of-the-art matchers. The improvement is split between the single adjustment step (AD), which significantly improves over initial results ($Orig$), and the iterative improvement of ADnEV. Additionally, when compared to the winners of the 2016 competitors of the ontology alignment competition, our methodology (which is generic and is not specifically tailored to this or that competition) achieves slightly better results by improving over recall, with $P = 0.81, R = 0.69, F = 0.72$ compared to $P = 0.81, R = 0.61, F = 0.67$.[6]

4.6 ADnEV Algorithm Properties

Having shown the algorithm effectiveness we next analyze ADnEV's properties. First, we investigate the effectiveness of multi-task network training compared to individually trained adjustor and evaluator. Next, we show that increasing the training set size using similarity matrix augmentation (see Section 4.2.2) improves the quality of the final outcome. Finally, we analyze the prediction capabilities of the suggested evaluators as a standalone, provising an empirical backing to the assumptions of Proposition 3.2.

4.6.1 *Multi-task Learning*

[6]see `http://oaei.ontologymatching.org/2011/results/benchmarks/index.html` and `http://oaei.ontologymatching.org/2016/conference/eval.html`

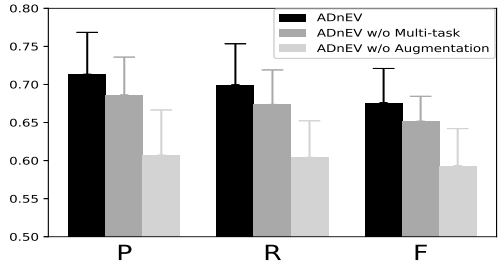


Figure 11: Effect of Multi-task Learning and Augmentation on ADnEV's performance: Precision (P), Recall (R), F1 (F)

To investigate the impact of multi-task learning, we trained two separate standalone versions of AD and EV (see Section 3.2) to be used in ADnEV.
Figure 11 compares aggregated performance of multi-task trained ADnEV (black) to the one of individually trained ADnEV (dark grey).

Multi-task ADnEV performs slightly better, yet not statistically significant. On average, multi-task training improves all quality measures by ∼4% overall measures. When zooming in on the different datasets, we identify a significant improvement with the Web-forms dataset, indicating that when a cross-domain matching challenge is on the line, multi-task learning becomes handy as it allows for better generalization of each of the networks. These results align well with the insight that combining the networks by their losses can help each network to collaborate with the other.

4.6.2 *Augmentation*

We analyze the effect of similarity matrix augmentation by training our neural adjustors and evaluators without expanding the train set. The performance of such networks as a part of the ADnEV (light grey) is compared to the full ADnEV in Figure 11. Recall that augmentation was aimed to overcome the scale of training data, which is a concern when training neural models for schema matching.

Overall, using augmentation during training increases the performance significantly by an average of +18%, +16%, +14%, in precision, recall, and F1, respectively, overall datasets. Specifically, we observe a boost of F1 performance by +10% over the Purchase Order dataset, +16% over the OAEI dataset, and +17% over the Web-forms dataset. This indicates that the augmentation improves generalization, enabling better performance over the test set.

4.6.3 *Prediction Monotonicity*

Finally, following previous works [29, 30, 64], we assess the quality of evaluators by comparing the Pearson's-r correlation between the predicted values and the actual match quality. We expect monotone evaluators (Definition 1) to demonstrate high correlation with the evaluation measure they aim to estimate. Recall that, for each evaluation measure E, we have trained an evaluator EV_E. Prediction performance was compared to $f(M)$ and two learning-based methods, namely, $LRSM_E$ [30] and DNN_E as described in Section 4.3.3. Table 3 reports Pearson's-r correlation over the four match quality measures (see Section 2.2).

All the evaluators proposed in this work 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 overall datasets. The

Table 3: Correlation of the Evaluators (Predictors) with Precision (P), Recall (R), F1 (F), and Cosine (Cos). Non-augmented are given in parentheses (see Section 4.6.2)

		$corr_P$	$corr_R$	$corr_F$	$corr_{Cos}$
Purchase Order	$f(M)$	.28*	-.24*	-.30*	-.21*
	$LRSM_P$	.28*	-.25*	-.29*	.04
	$LRSM_R$	.21*	.33*	.12*	.04
	$LRSM_F$	.24*	.03	.01	-.02
	$LRSM_{Cos}$	-.20*	-.10*	-.18*	-.11*
	DNN_P	.26* (.24*)	.10* (.10*)	.02 (.03*)	-.05* (.25*)
	DNN_R	.19* (.07*)	.28* (.07*)	.07* (.04*)	-.05* (.11*)
	DNN_F	.30* (.22*)	.06* (.10*)	.04 (.03)	-.03 (.22*)
	DNN_{Cos}	-.26* (-.11*)	-.21* (-.19*)	-.13* (-.10*)	.05 (-.17*)
	$\mathbf{EV}_P$	**.78*** (.72*)	.49* (.37*)	.72* (.78*)	.16* (.17*)
	$\mathbf{EV}_R$	.54* (.46*)	**.85*** (.62*)	.86* (.83*)	.06* (**.22***)
	$\mathbf{EV}_F$	.21* (.36*)	.54* (.56*)	**.95*** (.84*)	.13* (.15*)
	$\mathbf{EV}_{Cos}$	.41* (.31*)	.57* (.48*)	.75* (.66*)	.18* (.08*)
Web-forms	$f(M)$	-.08*	-.61*	-60*	-.27*
	$LRSM_P$	.76*	.34*	.36*	-.02
	$LRSM_R$	.35*	.76*	.36*	-.02
	$LRSM_F$	.68*	.31*	.32*	-.02
	$LRSM_{Cos}$	.76*	.35*	.36*	-.04*
	DNN_P	.36* (-.06*)	.36* (.07*)	.27* (.03*)	.42* (.06*)
	DNN_R	.34* (.20*)	.30* (.25*)	.22* (.21*)	.41* (.07*)
	DNN_F	.33* (.35*)	.42* (.32*)	.35* (.23*)	.20* (.42*)
	DNN_{Cos}	.23* (.19*)	.06* (.12*)	.01 (.09*)	.34* (.08*)
	$\mathbf{EV}_P$	**.82*** (.79*)	.20* (.42*)	.83* (.78*)	.31* (.36*)
	$\mathbf{EV}_R$	.45* (.20*)	.69* (.76*)	.87* (.72*)	.37* (.32*)
	$\mathbf{EV}_F$	.45* (.18*)	**.76*** (.66*)	**.87*** (.85*)	.35* (.30*)
	$\mathbf{EV}_{Cos}$	.48* (.35*)	.68* (.75*)	.71* (.75*)	**.50*** (.34*)
OAEI	$f(M)$	-.27*	-.66*	-.67*	-.30*
	$LRSM_P$	-.28*	-.11*	-.38*	-.01
	$LRSM_R$	-.17*	-.25*	-.32*	-.03
	$LRSM_F$	-.28*	-.14*	-.13*	-.02
	$LRSM_{Cos}$	-.28*	-.15*	-.13*	-.02
	DNN_P	-.45* (.01)	-.26* (.04)	-.23* (.02)	-.02 (.03)
	DNN_R	-.32* (-.08*)	-.26* (-.05)	-.21* (-.06)	-.02 (-.05)
	DNN_F	-.03 (-.02)	.12* (-.03)	-.01 (-.01)	-.06* (-.03)
	DNN_{Cos}	-.42* (-.05)	-.27* (-.07*)	-.22* (-.05)	-.05 (-.04)
	$\mathbf{EV}_P$	**.75*** (.69*)	.46* (.23*)	.85* (.84*)	.42* (.39*)
	$\mathbf{EV}_R$	.50* (.20*)	**.72*** (.35*)	.82* (.41*)	.46* (.16*)
	$\mathbf{EV}_F$	.31* (.48*)	.71* (.76*)	**.88*** (.87*)	.40* (.43*)
	$\mathbf{EV}_{Cos}$	.40* (.17*)	.65* (.50*)	.69* (.58*)	**.53*** (.43*)

non-binary evaluation, Cos, seems to be the hardest to predict and especially when dealing with large-scale matching problems (Purchase Order), demonstrating lower, yet still statistically significant, correlations.

Finally, EV evaluators display higher overall correlations (in absolute value) compared to all baselines. Evidently, applying data-driven feature extraction (as in $CRNN$) is better than human knowledge encoding as represented by $LRSM$, achieving better approximate evaluators, validating their usage in the ADnEV algorithm.

5. RELATED WORK

Some problems in data integration, *e.g.*, entity resolution [37, 50], ontology alignment [39], and dataset linking [27] have benefited from the recent advances, both in theory [33] and in the application [44] of deep learning. Mudgal *et al.* apply supervised deep learning using pre-trained character-level embedding to represent entities [50]. Ebraheem *et al.* [37], Kolyvakis *et al.* [39] and Fernandez *et al.* [27] assume a rich textual information (typically not the case in schema matching) and utilize word embeddings for better entity resolution, ontology alignment, and dataset linking. Recent attempts in schema matching assume the availability of instance data to apply supervised machine learning models [14]. However, schema matching by itself was not considered a good candidate for deep learning, due to insufficient amount of data. Our work is, to the best of our knowledge, the first to enable the use of deep learning via the similarity matrix abstraction, learning to adjust similarity matrices for improved match results.

Deep learning was used for other matching problems, *e.g.*, patch-based image matching [35] and graph matching [73], learning to compute a similarity matrix, given two elements (images/graphs). We train our networks with similarity matrices as inputs, capturing spatial and sequential dependencies between entries. We reuse the trained networks in an iterative manner to achieve an optimized output.

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. 2LMs (SMAs) were suggested in schema matching literature to achieve similar goals by adjusting similarity matrices, as introduced and discussed in Section 2.3. We further define the notion of consistent similarity matrix adjustment (Definition 2) to set a desiderata for well-behaved adjustors and show empirically that the proposed adjustor approximate this property well. Moreover, we compare against state-of-the-art 2LMs and show significant improvement using the proposed method.

Various mechanisms were proposed to manage matching uncertainty. Matching predictors were suggested to evaluate similarity matrices [64] in an unsupervised manner. Human experts are presented with a top-K ranked list of matches [55]. A recent work [30] combined both approaches using supervised learning, limited to $F1$ measure, proposing human engineered matching predictors as features to learn to improve the ranking of a top-K schema match list. We, instead, train a supervised deep learning evaluator with respect to an evaluation function of choice and show dominance (Section 4.6.3) over a feature-based evaluation. Moreover, we introduce the notion of monotonic prediction to characterize well-behaved predictors (Definition 1) and empirically validate that our newly designed evaluator approximates well a monotonic predictor, being highly correlated with a target evaluation measure.

Finally, we note that schema matching and ontology alignment [26] are closely related research areas, both aiming at finding matches between concepts. The two vary in their matching objects (schemata *vs.* ontologies), matching refinement (equivalence *vs.* richer semantics such as inclusion), and the underlying mathematical tools (*e.g.*, similarity matrix analysis *vs.* logic). They share many matching techniques (*e.g.*, [52]) and their benchmarks can be used interchangeably (we have evaluated our model on OAEI, an ontology dataset benchmark, as well). We believe this research can be readily applied to ontology alignment.

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 matching outcome. The ADnEV algorithm adjusts and evaluates similarity matrices, as created by state-of-the-art matchers, in an iterative manner. We showed that using a *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 and analyzed our various design choices.

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 human involvement once the model is learned, carries high promise for effectively utilizing schema matching in contemporary applications; specifically to those that depend on continuous online large-scale integration of new data sources to remain competitive in dynamic markets. In particular, the cross domain ability of the proposed algorithm allows the use of models, learned in a domain where human expertise exists, in new domains where expertise may be scarce.

As a possible future direction we intend to use pointer networks [69] to introduce constraints such as 1 : 1 matching into the learned network. Additionally, after showing the applicability over ontology alignment, tackling other data integration tasks such as entity matching and process matching using ADnEV is another direction for future work.

7. REFERENCES

[1] Adnev. https://github.com/shraga89/DSMA.

[2] Bpr. https://github.com/gamboviol/bpr.

[3] Coma. http://sourceforge.net/p/coma-ce/mysvn/HEAD/tree/coma-project.

[4] Factorization machines. https://github.com/coreylynch/pyFM.

[5] Isotonic regression, howpublished =.

[6] Keras. https://keras.io/.

[7] Oaei benchmark. http://oaei.ontologymatching.org/2011/benchmarks/.

[8] Ontobuilder research environment. https://bitbucket.org/tomers77/ontobuilder-research-environment.

[9] Surprise. http://surpriselib.com/.

[10] R. Ackerman, A. Gal, T. Sagi, and R. Shraga. A cognitive model of human bias in matching. In *Pacific Rim International Conference on Artificial Intelligence*, pages 632–646. Springer, 2019.

[11] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with coma++. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 906–908. Acm, 2005.

[12] Z. Bellahsene, A. Bonifati, and E. Rahm, editors. *Schema Matching and Mapping*. Data-Centric Systems and Applications. Springer, 2011.

[13] R. Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[14] C. Chen, B. Golshan, A. Y. Halevy, W.-C. Tan, and A. Doan. Biggorilla: An open-source ecosystem for data preparation and integration. *IEEE Data Eng. Bull.*, 41(2):10–22, 2018.

[15] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[16] S.-L. Chuang and K. C.-C. Chang. Integrating web query results: holistic schema matching. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 33–42, 2008.

[17] J. T. Connor, R. D. Martin, and L. E. Atlas. Recurrent neural networks and robust time series prediction. *IEEE transactions on neural networks*, 5(2):240–254, 1994.

[18] A. Das Sarma, X. Dong, and A. Halevy. Bootstrapping pay-as-you-go data integration systems. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 861–874, New York, NY, USA, 2008. ACM.

[19] D. De Una, N. Rümmele, G. Gange, P. Schachte, and P. J. Stuckey. Machine learning and constraint programming for relational-to-ontology schema mapping. In *IJCAI*, 2018.

[20] H. H. Do and E. Rahm. Coma: a system for flexible combination of schema matching approaches. In *Proceedings of VLDB*, pages 610–621. VLDB Endowment, 2002.

[21] H.-H. Do and E. Rahm. Matching large schemas: Approaches and evaluation. *Information Systems*, 32(6):857–885, 2007.

[22] X. Dong, A. Halevy, and C. Yu. Data integration with uncertainty. *The VLDB Journal*, 18:469–500, 2009.

[23] Z. Dragisic, V. Ivanova, P. Lambrix, D. Faria, E. Jiménez-Ruiz, and C. Pesquita. User validation in ontology alignment. In *International Semantic Web Conference*, pages 200–217. Springer, 2016.

[24] E. C. Dragut, W. Meng, and C. T. Yu. Deep web query interface understanding and integration. *Synthesis Lectures on Data Management*, 7(1):1–168, 2012.

[25] H. Drucker, C. J. Burges, L. Kaufman, A. J. Smola, and V. Vapnik. Support vector regression machines. In *Advances in neural information processing systems*, pages 155–161, 1997.

[26] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag New York Inc, 2007.

[27] R. C. Fernandez, E. Mansour, A. A. Qahtan, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Seeping semantics: Linking datasets using word embeddings for data discovery. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 989–1000. IEEE, 2018.

[28] A. Gal. *Uncertain Schema Matching*. Morgan & Claypool Publishers, 2011.

[29] A. Gal, H. Roitman, and T. Sagi. From diversity-based prediction to better ontology & schema matching. In *Proceedings of the 25th International Conference on World Wide Web*, pages 1145–1155. International World Wide Web Conferences Steering Committee, 2016.

[30] A. Gal, H. Roitman, and R. Shraga. Learning to rerank schema matches. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, (PrePrint https://ieeexplore.ieee.org/document/8944172), 2019.

[31] Z. Galil, S. Micali, and H. Gabow. An o(ev\logv) algorithm for finding a maximal weighted matching in general graphs. *SIAM Journal on Computing*, 15(1):120–130, 1986.

[32] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the*

fourteenth international conference on artificial intelligence and statistics, pages 315–323, 2011.

[33] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[34] A. Y. Halevy and J. Madhavan. Corpus-based knowledge representation. In *IJCAI*, volume 3, pages 1567–1572, 2003.

[35] X. Han, T. Leung, Y. Jia, R. Sukthankar, and A. C. Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3279–3286, 2015.

[36] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

[37] M. E. S. T. S. Joty and M. O. N. Tang. Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment*, 11(11), 2018.

[38] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[39] P. Kolyvakis, A. Kalousis, and D. Kiritsis. Deepalignment: Unsupervised ontology matching with refined word vectors. In *NAACL*, 2018.

[40] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 2010.

[41] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *SIGKDD*, 2008.

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[43] Y. Le Cun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11):41–46, 1989.

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

[45] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[46] A. Marie and A. Gal. Managing uncertainty in schema matcher ensembles. In *International Conference on Scalable Uncertainty Management*, pages 60–73. Springer, 2007.

[47] A. Marie and A. Gal. Boosting schema matchers. 5331:283–300, 2008.

[48] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings 18th International Conference on Data Engineering*, pages 117–128. IEEE, 2002.

[49] T. Milo and A. Somech. Next-step suggestions for modern interactive data analysis platforms. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 576–585. ACM, 2018.

[50] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*, pages 19–34. ACM, 2018.

[51] A. Niculescu-Mizil and R. Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632. ACM, 2005.

[52] L. Otero-Cerdeira, F. J. Rodríguez-Martínez, and A. Gómez-Rodríguez. Ontology matching: A literature review. *Expert Systems with Applications*, 42(2):949–971, 2015.

[53] L. Perez and J. Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[54] E. Peukert, J. Eberius, and E. Rahm. AMC-a framework for modelling and comparing matching systems as matching processes. In *ICDE*, 2011.

[55] A. Radwan, L. Popa, I. R. Stanoi, and A. Younis. Top-k generation of integrated schemas based on directed and weighted correspondences. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 641–654, New York, NY, USA, 2009. ACM.

[56] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10(4):334–350, 2001.

[57] S. Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012.

[58] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

[59] P. Rodríguez-Gianolli and J. Mylopoulos. A semantic approach to xml-based data integration. In *International Conference on Conceptual Modeling*, pages 117–132. Springer, 2001.

[60] J. Ross, L. Irani, M. Silberman, A. Zaldivar, and B. Tomlinson. Who are the crowdworkers?: shifting demographics in mechanical turk. In *CHI'10 Extended Abstracts on Human Factors in Computing Systems*, pages 2863–2872. ACM, 2010.

[61] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[62] S. Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[63] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[64] T. Sagi and A. Gal. Schema matching prediction with applications to data source discovery and dynamic ensembling. *The VLDB Journal*, 22(5):689–710, 2013.

[65] T. Sagi and A. Gal. Non-binary evaluation measures for big data integration. *The VLDB Journal*, 27(1):105–126, Feb. 2018.

[66] R. Shraga, A. Gal, and H. Roitman. What type of a matcher are you?: Coordination of human and algorithmic matchers. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2018, Houston, TX, USA, June 10, 2018*, pages 12:1–12:7, 2018.

[67] N. T. Toan, P. T. Cong, D. C. Thang, N. Q. V. Hung, and B. Stantic. Bootstrapping uncertainty in schema covering. In *Australasian Database Conference*, pages 336–342. Springer, 2018.

[68] A. Van den Oord, S. Dieleman, and B. Schrauwen. Deep content-based music recommendation. In *Advances in neural information processing systems*, pages 2643–2651, 2013.

[69] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[70] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 95–106, 2004.

[71] W. Yang, X. Zhang, Y. Tian, W. Wang, and J.-H. Xue. Deep learning for single image super-resolution: A brief review. *IEEE Transactions on Multimedia*, 2019.

[72] W. Yin, K. Kann, M. Yu, and H. Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.

[73] A. Zanfir and C. Sminchisescu. Deep learning of graph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2684–2693, 2018.

[74] C. Zhang, L. Chen, H. Jagadish, M. Zhang, and Y. Tong. Reducing uncertainty of schema matching via crowdsourcing with accuracy rates. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

APPENDIX

A. 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 decent 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 [44, 33]. Next we present two main classes of NNs, namely Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs).

A.1 Recurrent Neural Network (RNN)

The RNNs class is rooted in processing of sequential data [63] with applications in Natural Language Processing [15], Time Series Prediction [17], and Entity Resolution [50, 37]. 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 [72], we use a Gated Recurrent Unit (GRU)-based RNN [15] 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.

A.2 Convolutional Neural Network (CNN)

The CNN class is rooted in processing grid-like topology data [43] with applications in Image and Video Processing [42], Recommender Systems [68], 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 filtering size, the strides in which the number of neighboring cells are considered and padding each side of input boundaries.

B. THEOREM PROOFS

PROPOSITION B.1 (ADnEV DOMINANCE). *Let $EV(\cdot)$ be a monotonic evaluator (Definition 1) and let M and M^* be the input and output matrices of* ADnEV*, respectively. Then, $E(M^*) \geq E(M)$.*

PROOF *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:

  $$M^* = M^1 = M(\text{the algorithm terminates, line 8, Alg 1})$$

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

- 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.

  $$M^* = M^{T-1}(\text{line 9 in Alg 1})$$

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

  Since the condition in line 8 was satisfied for each $t \in \{1, ..., T-1\}$ 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 1):

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

PROPOSITION B.2 (ADnEV CONVERGENCE). *Let $AD(\cdot)$ be a $CSMA$ (Definition 2) with $0 < \varepsilon \leq 1$, let $EV(\cdot)$ be a monotonic evaluator (Definition 1), and let M and M^* be the input and output matrices of* ADnEV*, respectively.* ADnEV *returns a matrix with an ideal evaluation ($E(M^*) = 1$) in less than $\lceil \frac{1}{\varepsilon} \rceil$ steps.*

PROOF *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 $CSMA$:

$$E(M^{t-1}) + \varepsilon \leq E(M^t) = E(AD(M^{t-1})) \tag{6}$$

Note that since $\hat{E}(\cdot)$ is a monotone evaluator, we can assume that the algorithm will reach the t'th iteration (based on the condition in line 8) and since $\varepsilon > 0$:

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

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

Eq. 6 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:

$$E(M^0) + t \cdot \varepsilon = E(M) + t \cdot \varepsilon \leq E(M^t) \tag{7}$$

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

$$E(M^t) \geq t \cdot \varepsilon \tag{8}$$

Denoting M^E as an ideal evaluation matrix ($E(M^E) = 1$), we obtain:

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

Running the algorithm for $\lceil \frac{1}{\varepsilon} \rceil$ steps (step numbers are integers) we obtain (in the worst case) $M^* = M^{\frac{1}{\varepsilon}} \Rightarrow$ (Eq. 8)

$$E(M^*) = E(M^{\lceil \frac{1}{\varepsilon} \rceil}) \geq \lceil \frac{1}{\varepsilon} \rceil \cdot \varepsilon \geq 1$$

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