Improving the Efficiency and Effectiveness for BERT-based Entity Resolution

Bing Li[1], Yukai Miao[1], Yaoshu Wang[2], Yifang Sun[3], Wei Wang[4,1]*

[1]School of Computer Science and Engineering, University of New South Wales, Australia
[2]Shenzhen Institute of Computing Sciences, Shenzhen University, China
[3]School of Computer Science and Engineering, Northeastern University, China
[4]Dongguan University of Technology, China
{bing.li, yukai.miao, weiw}@unsw.edu.au, sunyifang@cse.neu.edu.cn, yaoshuw@sics.ac.cn

Abstract

BERT has set a new state-of-the-art performance on entity resolution (ER) task, largely owed to fine-tuning pretrained language models and the deep pair-wise interaction. Albeit being remarkably effective, it comes with a steep increase in computational cost, as the deep-interaction requires to exhaustively compute every tuple pair to search for co-references. For ER task, it is often prohibitively expensive due to the large cardinality to be matched. To tackle this, we introduce a siamese network structure that independently encodes tuples using BERT but delays the pair-wise interaction via an enhanced alignment network. This siamese structure enables a dedicated blocking module to quickly filter out obviously dissimilar tuple pairs, and thus drastically reduces the cardinality of fine-grained matching. Further, the blocking and entity matching are integrated into a multi-task learning framework for facilitating both tasks. Extensive experiments on multiple datasets demonstrate that our model significantly outperforms state-of-the-art models (including BERT) in both efficiency and effectiveness.

Introduction

The task of entity resolution aims at identifying co-referent tuples that refer to the same real-world entity from different data sources. Consider the two tables in Fig. 1, each table is a set of tuples about products gathered from Google and Amazon, respectively. Entity resolution identifies pairs of tuples that refer to the same product, *e.g.*, tuples 587 from Google and 2816 from Amazon are identified as a co-reference, and the same goes for tuples 1213 and 2837.

As a fundamental essence for data cleaning and data integration (Dong and Srivastava 2013), entity resolution has been widely applied in knowledge graph construction (Chen et al. 2015), e-commerce (Gokhale et al. 2014), *etc*. It has been extensively studied by means of various methodologies such as declarative rules (Hernández and Stolfo 1995), crowd-sourcing (Wang et al. 2012), and machine learning (Faruqui et al. 2015). Over the past few years, deep learning (DL) based ER models (Mudgal et al. 2018; Fu et al. 2019) have become the de-facto standard. They typically present a *representation-then-interaction* scheme: first-

---

Figure 1: An example of entity resolution: identifying co-referent tuples from two tables.

ly, encode each individual tuple (or attribute) into a semantic *representation* using deep neural networks (*e.g.*, RNN, LSTM, and GCN); then, make an *interaction* by comparing the representations of two tuples via similarity functions (*e.g.*, cosine, dot-product) or learnable classifiers (*e.g.*, MLP, SVM) to make final ER decision.

Recently, deep pre-trained language models, *e.g.*, ELMo (Peters et al. 2018) and BERT (Devlin et al. 2019), are promoting fast-paced advances for many NLP tasks, of course, no exception for entity resolution: BERT gains significant improvements and has set new state-of-the-art performances on many ER benchmarks. In contrast to the shallow and asynchronous interaction of the prior DL-based models, BERT makes a *deep and synchronous* interaction: the representation and interaction are performed simultaneously on a packed tuple pair (instead of on an individual tuple) to yield deeply-contextualized cross-encodings of the two input tuples. The deep-interaction is more effective in bridging the pervasive vocabulary mismatch (Khattab and Zaharia 2020), and accounts for its superior performances.

Despite being remarkably effective, such a deep-interaction comes with a poor scalability that become a formidable impediment for applying BERT in real ER scenario. This is mainly caused by the quadratic searching space as the deep-interaction requires to be fed with every tuple pairs. Even worse, it cannot integrate blocking techniques (*e.g.*, product quantization (Ge et al. 2013), local sensitive hash (LSH) (Ebraheem et al. 2018)) to effectively reduce the search space, since there is no representation for individual tuple on which the blocking relies. Thus, BERT has to exhaustively search all quadratic pairs for co-references, which is prohibitively expensive due to the large cardinality. To tackle this problem, some attempts (Reimer-

s and Gurevych 2019; Khattab and Zaharia 2020) regress to traditional representation-then-interaction scheme, while differently, use BERT as the encoder. Albeit being able to reduce time cost, these methods suffer from substantial performance declines comparing to BERT, as the most powerful deep-interaction is discarded.

To improve BERT-based entity resolution in both efficiency and effectiveness, we propose a novel ER model *BERT-ER*. We make three major contributions: (1) we show a formal analysis that BERT's representation and interaction processes can approximately be decomposed. The interaction part can be judiciously delayed yet enhanced to achieve an even better results. Also, with the delayed interaction, the model actually presents a siamese network structure that makes it able to integrate blocking modules. (2) For blocking, to better leverage the expressiveness of BERT, we propose an adaptive blocking model. Different from other learning-to-hashing methods, we propose a SVD-based hyperplanes orthogonalization to make BERT dominate the similarity measurement. (3) Existing ER models regard blocking and matching as two isolated tasks. We made the first effort to integrate the two tasks into a multi-task learning (MTL) framework. As such, the error can be back propagated to generalize better on both tasks. Further, the proposed framework is independent of a specific LM, and could be applied not only on BERT, but also on other LMs such as RoBERTA and ALBERT.

The extensive experiments on various datasets demonstrate that, comparing to BERT, BERT-ER has a significantly better effectiveness (1.5 pts in F1), while at the same time, improves the empirical efficiency by two orders of magnitude ($219\times \sim 304\times$).

Method

Model Overview

The general architecture of our model is summarized in Fig. 2. The model has a MTL framework that consists of three major components (denoted as boxes with grey-, blue-, and red-dotted border): BERT encoder, blocking decoder, and entity matching decoder. The base is the BERT encoder, which is shared by two task-specific decoders, namely, blocking and entity matching.

BERT Encoder. Given a pair of tuple $\langle u1, u2\rangle$, the BERT encoder uses BERT (Devlin et al. 2019), a powerful pretained language model, to independently encode the tuples into vector representations. Specially, to leverage the schema information of tuples (*e.g.*, table-attribute-token hierarchy), we add learned table embeddings (reuse the segmentation embeddings of BERT), and a set of attribute embeddings to enable the model to differentiate the attributes and tables.

Blocking Decoder. Blocking aims to speed-up entity resolution by grouping potentially co-referent tuples into blocks such that the fine-grained matching are exclusively performed within blocks. To this end, the blocking decoder employs learned hash functions to map each tuple into k-bits binary hash codes. Tuples with similar hash codes are assigned into the same hash bucket (*i.e.*, block). The blocking
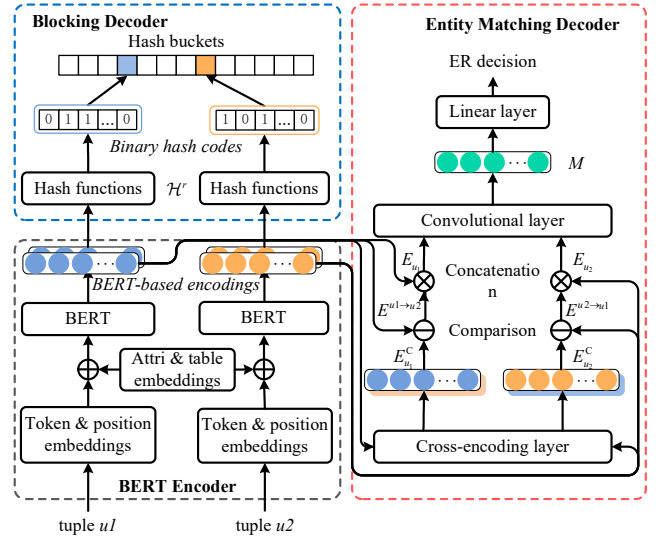

Figure 2: The general architecture of our model.

decoder is supposed to drastically reduce the number of pairs to be matched , with least amount of sacrifice on the recall.

Entity Matching Decoder. The entity matching task aims to compare each of the tuple pairs to decide whether they are co-referent. The decoder receives the BERT-based encodings of the two tuples, and compare them via an enhanced deep alignment module, which consists of three cascaded layers: a cross-encoding layer, a set of comparison and concatenation operations, and a convolutional layer. The alignment module outputs matching features, which are then fed into a simple linear layer to get the final ER decision. The goal of entity matching task is to achieve an F1 score (*i.e.*, in terms of both precision and recall) as high as possible.

We will elaborate on the two task-specific decoders and the joint learning framework in the following subsections.

Entity Matching via Delayed and Enhanced Alignment

BERT is a multi-layer transformer (Vaswani et al. 2017). Each layer consists of a multi-head self-attention mechanism and a position-wise feed-forward network.

Assume a tuple pair $\langle u1, u2\rangle$ is packed and tokenized to feed into BERT. The self-attention mechanism first projects each token into queries, keys, and values (Vaswani et al. 2017). Let K and V be keys and values matrices for the packed sequence, and K_{u1} and V_{u1}, K_{u2} and V_{u2} be the keys and values regarding tuple $u1$ and $u2$, respectively. For the i-token t_i (assume $t_i \in u1$), its encoding s_i after self-attention mechanism can be rewritten as:

$$
\begin{aligned}
s_i &= \text{softmax}(q_i \cdot K)V \\
&= \underbrace{\eta_1\text{softmax}(q_i \cdot K_{u1})V_{u1}}_{\text{inner-encoding } s_i^I} + \underbrace{\eta_2\text{softmax}(q_i \cdot K_{u2})V_{u2}}_{\text{cross-encoding } s_i^C},
\end{aligned}
$$
(1)

where q_i is the query vector of token t_i, η_1 and η_2 are scaling-factors.

$u1$:	micro	##so	##ft	word	2007	version	upgrade	micro	##so	##ft	109	.	95
$u2$:	micro	##so	##ft	money	2007	home	business	(	p	##c	)	89	.	99

Figure 3: The attention distribution between the decision vector (encoding of the final layer [CLS] token) and the tokens of the two input tuples (tuples 587 and 2837 of Fig. 1). Using a well-trained BERT model on *Amazon-Google* dataset.

In Eq. 1, s_i can be decomposed into two parts: an inner-encoding s_i^I that only focuses on the tuple that t_i belongs to; and a cross-encoding s_i^C that incorporates information from the other tuple. After applying position-wise feed-forward network (PFFN), the final encoding e_i can be approximated as the sum of applying PFFN on s^I and s^C, separately:

$$e_i = \text{PFFN}(s_i^I + s_i^C) \approx \underbrace{\text{PFFN}(s_i^I)}_{\text{(a) representation}} + \underbrace{\text{PFFN}(s_i^C)}_{\text{(b) interaction}} \quad (2)$$

According to Eqs. 1 and 2, each layer of BERT can functionally be decomposed into two parts: (a) one for generating *representation* using the contextual information within respective tuple; and (b) the other one performs a deep-*interaction* between tuples. For ER task, we believe part (a) mainly works for capturing the contextualized features of token itself and part (b) works for implicitly (and effectively) aligning the two tuples. This can be illustrated using the example in Fig. 3, where the final attentions focus on informative yet unmatching tokens (*e.g.*, "word" and "upgrade").

We wish to retain its representation part but delay and enhance its alignment. As such, we can leverage the expressiveness of BERT, while at the same time, being able to integrate blocking to speed-up the processing.

Delayed and Enhanced Alignment. The original alignment $\text{PFFN}(s_i^C)$ can be enhanced in three aspects: (1) cross-encoding may not be the most appropriate features since it is less-effective and implicit for aligning tuples; (2) instead of sharing parameters, token representation and alignment features should have separated parameters, since their semantic for ER is different; (3) multi-gram features have been proved to be effective for ER (Li et al. 2020), while the PFFN is only able to extract uni-gram features.

For aspect (1), we use bilateral comparison features, rather than solely relying on cross-encodings. Firstly, we define the cross-encodings E_{u1}^C and E_{u2}^C as:

$$E_{u1}^C = \text{softmax}(Q_{u1} K_{u2}^\mathsf{T}) E_{u2}^I$$
$$E_{u2}^C = \text{softmax}(Q_{u2} K_{u1}^\mathsf{T}) E_{u1}^I, \quad (3)$$

where $E_{u1}^I \in \mathbb{R}^{m \times d_B}$ and $E_{u2}^I \in \mathbb{R}^{n \times d_B}$ are BERT-based representations for $u1$ and $u2$ (dimensionality m and n are token sequence lengths, d_B are the output dimension of BERT). Q and K are queries and keys matrices, which are defined as

$$Q = E^I \mathcal{W}_Q, \quad K = E^I \mathcal{W}_K, \quad (4)$$

where $\mathcal{W}_Q \in \mathbb{R}^{d_B \times d_B}$ and $\mathcal{W}_K \in \mathbb{R}^{d_B \times d_B}$ are parameters for query and key projection, respectively.

We define two comparison functions, the first is the subtraction function (Wang and Jiang 2017; Li et al. 2020)

$$f_{sub}(E^I, E^C) = (E^I - E^C) \odot (E^I - E^C), \quad (5)$$

where $\odot$ denotes the Hadamard product. The other one is the multiplication function (Wang and Jiang 2017), which is defined as:

$$f_{mul}(E^I, E^C) = E^I \odot E^C. \quad (6)$$

The above two functions would be collapsed to L_2 distance and dot-product if we sum up the vector. By not summing up, they can keep more information to be trained by the following layers.

To make the comparison robust, we make a bilateral matching (Wang, Hamza, and Florian 2017) that compares $u1$ and $u2$ in both $u1 \rightarrow u2$ and $u2 \rightarrow u1$ directions, *i.e.*, compare $u1$ against $u2$, and compare u_2 against $u1$. Take $u1 \rightarrow u2$ direction as an example, we concatenate the subtraction and multiplication comparison features:

$$E^{u1 \rightarrow u2} = [f_{sub}(E_{u1}^I, E_{u1}^C); f_{mul}(E_{u1}^I, E_{u1}^C)] \quad (7)$$

Compared with cross-encoding, the enhanced comparison features $E^{u1 \rightarrow u2} \in \mathbb{R}^{m \times 2d_B}$ are more explicit and could provide more information to reason to what extent a token in $u1$ is matched with its counterpart in $u2$.

To incorporate representation features, $u1$'s encoding $E_{u1} \in \mathbb{R}^{m \times 3d_B}$ is represented as the concatenation (rather than sum) of $u1$'s representation E_{u1}^I and comparison feature $E^{u1 \rightarrow u2}$:

$$E_{u1} = [E_{u1}^I; E^{u1 \rightarrow u2}]. \quad (8)$$

The concatenation make the two terms use different weight parameters to enhance aspect (2).

To enhance aspect (3), we employ a convolutional layer (Conv) with multiple kernel sizes to effectively extract multi-gram features and aggregate E_{u1} into a fixed-size matching vector $M_{u1} \in \mathbb{R}^{1 \times gc}$:

$$M_{u1} = \text{Conv}(E_{u1}), \quad (9)$$

where $\text{Conv}(\cdot)$ is a composite function consisting of four cascaded operations: a set of convolutions $co_1, co_2, ..., co_g$, a batch normalization (BN) (Ioffe and Szegedy 2015), a rectified linear unit (ReLU), and a 1-max-over-time pooling (Kim 2014). The i-th convolution has learnable parameters $\mathcal{W}_i \in \mathbb{R}^{c \times h \times 3d_B}$ indicating that there are c kernels. Each kernel has size $h \times 3d_B$ convolving h adjacent vectors to capture h-gram matching features. The 1-max-pooling operation selects the largest value over the feature map of a particular kernel. The outputs are with fixed size $1 \times gc$.

The output matching vector $M \in \mathbb{R}^{1 \times 2gc}$ is the concatenation of the matching vectors of both $u1$ and $u2$ ($M = [M_{u1}; M_{u2}]$), which is then fed into a simple linear layer for ER decision.

Loss Function for Entity Matching. For entity matching task, we adopt the standard cross-entropy loss:

$$\mathcal{L}_M = \mathcal{C}(y, h(\mathcal{W}, M)), \tag{10}$$

where $h(\mathcal{W}, M)$ is the predicted distribution based on the final matching vector M, $\mathcal{C}(l, p)$ denotes cross-entropy function between l and p, and y is the annotated label.

Adaptive Blocking with Orthogonalized Hyperplanes

Majority of prior ER models regard blocking as an isolated process with the entity matching, and the blocking methods (*e.g.*, key-based (Papadakis et al. 2020) or LSH-based blocking (Ebraheem et al. 2018)) are designed to be data-independent and matching-unaware. As such, they cannot learn to flexibly fit the data, and utilise the results of matching to rectify blocking-incurred error.

To tackle this, we use a learnable hash-based blocking method, which could be aware of the matching features from the shared BERT encoder.

Learnable Hash Model. The goal is to learn a mapping $\mathcal{H}$ from BERT encoding space $\mathbb{R}^d$ to k-bit binary hash space, namely, $\mathcal{H} : \mathbb{R}^d \rightarrow \{+1, -1\}^k$. Typically, the hash codes of co-referent tuples should be close in Hamming space, and the hash codes of dissimilar tuples should be far away.

The mapping $\mathcal{H}$ consists of k hash functions $h_1, h_2, ..., h_k$, each of which is a learnable hyperplane through the origin and used for generating a binary value. Formally, $\mathcal{H}$ can be defined as:

$$\mathcal{H}(t) = \text{sign}(t\mathcal{X}), \tag{11}$$

where $\mathcal{X} \in \mathbb{R}^{d \times k}$ is k learnable hyperplanes, and sign is the signum function to binarize real values.

L_2 **Relaxation for Training.** Directly optimizing Eq. 11 is infeasible since the signum function is not differentiable. Thus, we adopt the L_2 relaxation (Liu et al. 2016; Chen et al. 2018) as it is more training-efficient and has stellar results.

The main idea of L_2 relaxation is to move the binary constraint from hash functions to a regularizer in the loss function, and use L_2 distance in Euclidean space to approximate the Hamming distance. With L_2 relaxation, the mapping $\mathcal{H}$ is relaxed to $\mathcal{H}^r$:

$$\mathcal{H}^r(t) = t\mathcal{X}. \tag{12}$$

Given the encodings of two tuples t_i and t_j, and the label y ($y = 1$ if they are co-referent; otherwise, $y = 0$), the loss function is naturally designed to pull co-referent tuples closer, while pushing unmatching tuples away from each other:

$$\begin{aligned}
\mathcal{L}_B^r = &\frac{1}{2}y \, ||\mathcal{H}^r(t_i), \mathcal{H}^r(t_j)||_2 \\
&+ \frac{1}{2}(1-y)\max(m - ||\mathcal{H}^r(t_i), \mathcal{H}^r(t_j)||_2, 0) \\
&+ \gamma(|||\mathcal{H}(t_i)| - 1||_1 + |||\mathcal{H}(t_j)| - 1||_1),
\end{aligned} \tag{13}$$

where $||\cdot||_1$ and $||\cdot||_2$ are the L_1- and L_2-norm, respectively. $|\cdot|$ is the absolute operation, γ is the weight of the regularizer. Notably, the last term is the regularizer (Liu et al. 2016) of L_2 relaxation, which aims to push the values to either +1 or -1 to facilitate the binary constraint.

Hyperplanes Orthogonalization. The mapping $\mathcal{H}$ is supposed to faithfully preserve two tuples' similarity in the BERT encoding space (*i.e.*, isometry). To this end, the mapping should satisfy the following constraint:

$$\begin{aligned}
||t_i, t_j||_2 &= ||\mathcal{H}^r(t_i), \mathcal{H}^r(t_j)||_2 \\
&= \sqrt{(t_i - t_j)\mathcal{X}\mathcal{X}^\mathsf{T}(t_i - t_j)^\mathsf{T}}
\end{aligned} \tag{14}$$

A sufficient condition makes Eq. 14 hold is $\mathcal{X}\mathcal{X}^\mathsf{T} = \mathbf{I}$ ($\mathbf{I}$ is the identity matrix), indicating $\mathcal{X}$ is a unitary matrix. The orthogonality of unitary matrix also ensures independency of hash functions.

To orthogonalize hyperplanes, a straightforward solution is adding a regularizer term R_o (Muja and Lowe 2014):

$$R_o = ||\mathcal{X}\mathcal{X}^\mathsf{T} - \mathbf{I}||_F, \tag{15}$$

where $||\cdot||_F$ denotes the Frobenius-norm.

As the regularization is not strict, it may lead to suboptimal results. To keep $\mathcal{X}$ strictly unitary, we propose another SVD-based approach. This approach decomposes $\mathcal{X}$ using singular vector decomposition (SVD) (*i.e.*, $\mathcal{X} = USV^\mathsf{T}$) to get three decomposed matrices, U, S, and V, where U and V is $d \times d$ and $k \times k$ unitary matrix, respectively, and S is a $d \times k$ diagonal singular value matrix. Then, we the replace $\mathcal{X}$ with orthogonal matrix US. As we have

$$\begin{aligned}
||\mathcal{H}^r(t_i), \mathcal{H}^r(t_j)||_2 &= \sqrt{(t_i - t_j)\mathcal{X}\mathcal{X}^\mathsf{T}(t_i - t_j)^\mathsf{T}} \\
&= \sqrt{(t_i - t_j)USV^\mathsf{T}VS^\mathsf{T}U^\mathsf{T}(t_i - t_j)^\mathsf{T}} \\
&= \sqrt{(t_i - t_j)US(US)^\mathsf{T}(t_i - t_j)^\mathsf{T}},
\end{aligned}$$

the discriminative ability of learned hyperplanes will be kept, while at the same time, the L_2 distance remains unchanged. It is worth noting that, since $\mathcal{X}$ is not full rank, back propagation through SVD is intractable. To make it trainable, we assume $\mathcal{X}$ is a latent matrix, from which US are decomposed. Initially, S is assigned with an identity matrix[1], and U and V are randomly initialized. During training, gradients can be back-propagated on US. Then, we restore the latent matrix $\mathcal{X}$ by multiplying the V with US. At this step, the gradients are indirectly applied on $\mathcal{X}$. Finally, the new US and V can be fetched by decomposing $\mathcal{X}$.

Hash Code for Prediction. In the predication phase, we use the original definition of $\mathcal{H}$ in Eq. 11 to generate a k-bit hash code for a tuple. To balance the trade-off between recall and the number of matchings, we merge q-nearest buckets ($0 \leq q \leq 2$ in common practice) into a block for searching co-references.

Joint Learning of Blocking and Entity Matching

The final training objective of the MTL framework is to minimize the following loss function:

$$\mathcal{L} = \alpha\mathcal{L}_B^r + (1-\alpha)\mathcal{L}_M, \tag{16}$$

where $\mathcal{L}_B^r$ (Eq. 13) and $\mathcal{L}_M$ (Eq. 10) is the loss for blocking and entity matching, respectively. α is the weight to balance the two tasks.

---

[1]We can keep S fixed to the identity matrix during training, but leave it as free parameters could slightly improve performance.

Datasets	Domain	Size	# Pos.
Amazon-Google[2]	software	11460	962
BeerAdvo-RateBeer[2]	beer	450	68
iTunes-Amazon[2]	music	539	132
DBLP-ACM[2]	scholar	12363	2220
Walmart-Amazon(B)[3]	electronic	56.4 million	1154
Amazon-Google(B)[4]	software	4.4 million	1300

Table 1: Evaluation datasets for our experiments.

Experimental Evaluation

We evaluated the effectiveness and efficiency of our model on both entity resolution and blocking tasks.

Evaluation Datasets

Entity Matching. We used four widely used benchmark datasets covering diverse domains such as products, music, and scholar. Table 1 lists some statistics (the first four datasets). Each dataset contains a list of *after-blocking* tuple pair followed with gold labels. All datasets have been split into train/dev/test subsets by Mudgal et al. (2018).

Blocking. We adopted two widely used datasets *Walmart-Amazon (B)* and *Amazon-Google (B)*. Their detailed statistics are listed in Table 1 (the last two datasets). Following Ebraheem (2018), the negative examples are randomly sampled with a positive to negative ratio 1:100. All positive and negative examples are shuffled and split into train/test with the ratio 4:1.

Training Settings

Our model was implemented using Pytorch 1.4 with Python 3.7, and ran on an Nvidia Titan V GPU. We used the popular *transformers*[5] library for the pre-trained BERT model.

BERT Encoder. The BERT was initialized using a standard $BERT_{BASE}$ model[6]. Each tuple was tokenized with pre-trained BERT tokenizer and packed with the form [CLS]*tuple tokens*[SEP], and padded the tokenized sequences to a max length of 120.

Blocking Decoder. The blocking decoder was on the last-layer [CLS] token. The hash bits k and tolerance threshold q were set to 8 and 1. Following Liu et al. (2016), m was set to $2k = 16$, and the regularizer weight γ was 0.01. We under-sampled negative instances to yield a 1:10 pos/neg rate.

Entity Matching Decoder. The entity matching decoder was performed on the output of the 2-nd layer of BERT[7].

---

[2] http://pages.cs.wisc.edu/~anhai/data1/deepmatcher_data/

[3] https://sites.google.com/site/anhaidgroup/useful-stuff/data

[4] https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution

[5] https://github.com/huggingface/transformers

[6] Compared with $BERT_{LARGE}$, $BERT_{BASE}$ has similar results while being more cost-efficient.

[7] Empirically, placing on 2nd layer has better results. The performance discrepancies are less than 3 pts for all 12 layers.

The weights of query and key projections, and the final linear layer were initialized using Xavier with gain 1. Kernel sizes of the convolutional layer was set to $[1, 2, 3]$, each has $c = 128$ kernels. The balance weight $\alpha = 0.2$.

For optimization, we used AdamW (Loshchilov and Hutter 2019) with an initial learning rate 10^{-5}, eps 10^{-8}, and the gradient clipping 5; the batch size is set to 32; all other hyper-parameters were their default values.

In each round, the model was run 10 times with a maximum of 50 epochs and reported the best performing models as the result.

Evaluating Entity Matching

Baselines and Metrics. We compared our model with seven state-of-the-art entity matching models, including a feature-based model Magellan, and four DL-based models RNN, Hybrid, MPM, and GraphER. Besides, we also compared with two BERT-based models: the standard interactive BERT, and a siamese BERT model SBERT (Reimers and Gurevych 2019).

Following common practices, we use the F1 score on test datasets as the metric.

Main Results. Table 2 presents the performance comparison of our BERT-ER model compared with baselines on the four benchmark datasets. We can see that:

1) Our model significantly outperforms all baselines, achieving new state-of-the-art results. On average, our model achieved a 1.45 pts improvement over the best baselines, and 1.5 pts improvement over BERT. This demonstrates our delayed and enhanced alignment paradigm is highly effective on alignment-focused tasks such as entity resolution: one can get a even better results by judiciously delaying and enhancing BERT's deep-interaction part.

2) Compared with the four non-BERT DL model (*i.e.*, RNN, MPM, Hybrid, and GraphER), the two BERT-based models (*i.e.*, BERT and BERT-ER) have an average 2.34 pts improvement. This demonstrates the deep pretrained LMs are more expressive than traditional DL-based models. Comparing deep-interactive model BERT-ER with shallow-interactive model SBERT, although they are both BERT-based, the performance gap is huge (over 25 pts). This indicates the deep-interaction is vitally important for performance improvements of ER tasks. Our enhanced alignment is more effective than the shallow-alignment of SBERT.

3) Our model has more advantages on *Amazon-Google* and *iTunes-Amazon*. While on *DBLP-ACM*, the performances for all models are similar. The main reason is that *Amazon-Google* and *iTunes-Amazon* are semantically deep, as they have more textual attributes, whereas *DBLP-ACM* is much cleaner and well-formatted. Finding co-references on semantically deep datasets hinges on aligning informative "key" tokens (*e.g.*, "upgrade" in Fig. 3) by bridging vocabulary mismatch. Thus, the models with deep-interaction (*e.g.*, BERT and BERT-ER) are desired.

Ablation Study. We conducted an ablation study to evaluate the contribution of each component. Table 3 shows the results. The biggest performance gaps happened in removing comparison functions. This indicates comparison fea-

Model	Amazon-Google	BeerAdvo-RateBeer	iTunes-Amazon	DBLP-ACM
Magellan (Konda et al. 2016)	49.1	78.8	91.2	98.4
RNN (Mudgal et al. 2018)	59.9	72.2	88.5	98.3
Hybrid (Mudgal et al. 2018)	69.3	72.7	88	98.4
MPM (Fu et al. 2019)	70.7	-	-	-
GraphER (Li et al. 2020)	68.1	79.7	-	-
SBERT (Reimers and Gurevych 2019)	44.8	42.1	74.1	94.3
BERT	73.1	**87.5**	93.1	98.2
BERT-ER	**75.3**	**87.5**	**96.4**	**98.7**

Table 2: F1 (%) of our model and baselines on entity matching task. Since the benchmark datasets only contains after-blocking instances, to be consistent, our model is trained only with entity matching decoder.

Model	Amazon-Google		BeerAdvo-RateBeer		iTunes-Amazon		DBLP-ACM	
	F1 (%)	ΔF1	F1 (%)	ΔF1	F1 (%)	ΔF1	F1 (%)	ΔF1
BERT-ER	**75.3**		**87.5**		**96.4**		**98.7**	
- Comparison functions	62.4	-12.9	61.1	-26.4	92.6	-3.8	98.3	-0.4
- Multi-gram kernel	73.9	-1.4	84.2	-3.3	94.3	-2.1	98.7	0.0
- Attribute embeddings	74.4	-0.9	83.3	-4.2	95.5	-0.9	97.3	-1.4
- Pre-training	65.8	-9.5	62.2	-25.3	87.1	-9.3	96.8	-1.9

Table 3: Ablation test for entity matching task. In each test, we remove a component from the full BERT-ER model.

tures are necessary for fine-grained comparison. Removing pre-training incurs a roughly 11.5 pts performance decline. This demonstrates that pre-training is vitally importance for improving representation ability, especially for datasets with small training instances, *e.g.*, *BeerAdvo-RateBeer*. The absence of multi-gram kernels leads to roughly 1.7 pts decline, which again demonstrates the usefulness of multi-gram features for ER tasks. Removing attribute embeddings leads to 1.85 pts decline, which indicates involving structural information could help identify co-references. From above we can conclude that these components are essential and contribute significantly to our model.

Evaluating Blocking

Baselines and Metrics. We tested two settings of our model: ABOH-regularizer (with the regularizer of Eq. 15) and ABOH-SVD (with SVD-based orthogonalization). We adopted LSH as the baseline, as it represents the *de-facto* blocking technique for DL-based ER systems (Ebraheem et al. 2018; Papadakis et al. 2020).

The evaluation is based on two metrics that are widely used by prior research (Michelson and Knoblock 2006; Papadakis et al. 2020): Pair Completeness (PC) and Reduction Ratio (RR). PC corresponds to recall, evaluating the ratio of co-references assigned with the same blocks against the total number of co-references. RR measures the reduction in the number of pairwise comparisons against the brute-force approach. Higher values for PC indicate higher effectiveness of the blocking method, while higher values for RR indicates higher efficiency.

Main Results. Figs. 4(a) and 4(b) show RRs and PCs under different bits size k on *Walmart-Amazon (B)* and *Amazon-Google (B)*, respectively. We can see that the general trends for both datasets are similar: with the increasing
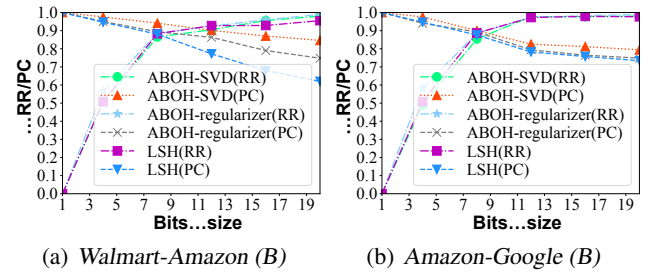

Figure 4: RRs and PCs of ABOH-SVD, ABOH-regularizer, and LSH against various bits size k.

of k, RRs quickly raise to a flat peak region and PCs gradually decreases. This is due to the fact that the increasing bit size exponentially increases the number of hash buckets, which makes more options of assignment. While at meantime, it reduces the likelihood of two true co-referent tuples being placed in the same block.

All the three models have similar RRs in both Figs. 4(a) and 4(b). With more than 8 bits, all models have a high RR (more than 0.85), indicating they avoid most of comparisons and only need to compare the remaining few. For PCs, the two variants of ABOH are better than LSH nearly in all cases. Take $k = 8$ as an example, ABOH-SVD is roughly 9 pts and 3 pts better on the two datasets, respectively. ABOH-regularizer is roughly 2 pts better than LSH. Further, the advantage of ABOH is more obvious on *Walmart-Amazon (B)*. We believe the main reason is that, the adaptive and orthogonalized hyperplane could better use the expressiveness of BERT. Compared with ABOH-regularizer, ABOH-SVD has a better PC (on average 3.75 pts) but slightly lower RR (2.1 pts). Since our entity matching decoder is efficient enough,

a model with better PCs (ABOH-SVD) is preferred. The experimental results demonstrate our blocking method could drastically reduce the number to be matched, in the meanwhile, keeps a high recall.

Version	Walmart-Amazon (B)			Amazon-Google (B)		
	RR	PC	F1	RR	PC	F1
MTL	86.2	**96.1**	**94.9**	**88.7**	**91.2**	**96.0**
Matching	-	-	94.2	-	-	95.6
Blocking	**86.5**	94.1	-	85.4	90.0	-

Table 4: Performance (%) comparison of the multi-task and single-task BERT-ER.

Effectiveness of the MTL Scheme

To evaluate the effectiveness of the MTL scheme, we compared BERT-ER's MTL version with its single-task learning versions. Table 4 presents the results. From Table 4, we can see that by joint-learning blocking and entity matching, both tasks have considerable improvements comparing with solely learning one. On average, the F1 scores of entity matching improve 0.55 pt. For blocking, owing to being aware of fine-grained matching features, the improvements are more obvious: RR and PC improves 1.5 pts and 1.6 pts, respectively. This demonstrates that our MTL framework can effectively improve model's generalization ability to facilitate the performance improvements of both tasks.

Model	Phase	WA (B)	AG (B)	Speed
BERT-ER	Encoding	43.69 s	22.38 s	247.80 t/s
	Blocking	19.42 s	6.56 s	601.06 t/s
	Matching	18.48 m	98.16 s	9003.06 p/s
	Total	**19.54 m**	**127.1 s**	-
BERT	Total	99.18 h	7.73 h	157.90 p/s

Table 5: The empirical decoding time and speed of BERT-ER and BERT on full-size *WA(B)* and *AG(B)* datasets. t/s: # of tuples per second; p/s: # of pairs per second.

Empirical Efficiency

Table 5 presents the empirical time cost and decoding speed. We can see that:

1) Due to being able to integrate the blocking techniques, BERT-ER is far more efficient than BERT: decoding *WA (B)* and *AG (B)* requires $304\times$ less time (20 minutes v.s. 99 hours) and $219\times$ less time (3 minutes v.s. 7.7 hours), respectively. Moreover, the encoding phase of BERT-ER could be pre-computed offline to further accelerate the processing.

2) The decoding speed of BERT-ER's matching phase is $57\times$ faster than BERT's. This indicates our model will still have a better efficiency, even if BERT could be equipped with the same blocking module.

3) The matching phase is the most time-consuming one among the three phases of BERT-ER. This is due to the fact that, although the blocking could effectively reduce nearly

90% of comparisons, considering its enormous cardinality (*WA (B)* and *AG (B)* has 4.4 million and 56.4 million pairs, respectively), the remaining 10% could still be large. Fortunately, the matching phase is efficient enough, it will not incur too much overhead.

This evaluation demonstrates that our model is of high efficiency, and able to work in real ER scenarios.

Related Works

Entity Matching. Prior works can be classified as declarative rules based, crowd-sourcing-based, and machine learning (ML) based. The rule-based methods adopted declarative matching rules that are either pre-defined (Hernández and Stolfo 1995)or synthesized (Singh et al. 2017) for matching tuple pairs. The crowd-sourcing-based methods (Wang et al. 2012; Gokhale et al. 2014; Firmani, Saha, and Srivastava 2016) employ crowd-sourcing workers to manually annotate tuples. However, both methods highly rely on human efforts. ML-based methods train different classifiers, such as SVM (Bilenko and Mooney 2003), active learning (Sarawagi and Bhamidipaty 2002), MLP (Ebraheem et al. 2018), on manually collected (Konda et al. 2016) or deep neural features (Mudgal et al. 2018; Ebraheem et al. 2018; Fu et al. 2019; Li et al. 2020). Recently, BERT (Devlin et al. 2019) and DITTO (Li et al. 2021) deliver remarkable effectiveness on many ER datasets. However, they are computationally expensive as they need pair-wise deep interactions.

Blocking. Prior blocking methods could be classified as rule-based, sorting-based, and hash-based. Rule-based methods group tuples by static keys or decision rules that are derived by experts or from mere heuristics. Sorting-based methods (Papadakis et al. 2015; Kenig and Gal 2013) group tuples by efficiently sorting their textual similarities measured by various similarity functions. Hash-based approaches adopt hashing techniques (*e.g.*, Min-Hashing (Steorts et al. 2014; Wang, Cui, and Liang 2015) and LSH (Ebraheem et al. 2018)) to map tuples into hash buckets. Ebraheem *et al.* (Ebraheem et al. 2018) introduced LSH into deep-learning based ER system. Zhang *et al.* (Zhang et al. 2020) proposed tuple signatures and use LSH to perform fast NN search. Hashing-based methods can be applied on deep-neural representations. However, the current blocking techniques are matching-unaware, which cannot utilize matching features to further improve performances.

In contrast, our model integrates a learnable blocking module to improve the efficiency of BERT. And use a MTL framework to make the blocking matching-aware.

Conclusion

In this paper, we propose a novel BERT-based ER model. By delaying and enhancing BERT's interaction part, our model is able to integrate an adaptive blocking module. Further, the blocking and matching are integrated into a MTL framework to facilitate both tasks. Compared to a standard BERT, our model improves the effectiveness by 1.5 pts, while being $219\times \sim 304\times$ faster.

Acknowledgments

References

Bilenko, M.; and Mooney, R. J. 2003. Adaptive duplicate detection using learnable string similarity measures. In *SIGKDD*, 39–48. ACM.

Chen, Y.-N.; Wang, W. Y.; Gershman, A.; and Rudnicky, A. 2015. Matrix factorization with knowledge graph propagation for unsupervised spoken language understanding. In *ACL*, 483–494.

Chen, Z.; Yuan, X.; Lu, J.; Tian, Q.; and Zhou, J. 2018. Deep hashing via discrepancy minimization. In *CVPR*, 6838–6847.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT (1)*, 4171–4186.

Dong, X. L.; and Srivastava, D. 2013. Big data integration. In *ICDE*, 1245–1248. IEEE.

Ebraheem, M.; Thirumuruganathan, S.; Joty, S.; Ouzzani, M.; and Tang, N. 2018. Distributed representations of tuples for entity resolution. *VLDB* 11(11): 1454–1467.

Faruqui, M.; Dodge, J.; Jauhar, S. K.; Dyer, C.; Hovy, E.; and Smith, N. A. 2015. Retrofitting Word Vectors to Semantic Lexicons. In *NAACL-HLT*, 1606–1615.

Firmani, D.; Saha, B.; and Srivastava, D. 2016. Online entity resolution using an oracle. *VLDB* 9(5): 384–395.

Fu, C.; Han, X.; Sun, L.; Chen, B.; Zhang, W.; Wu, S.; and Kong, H. 2019. End-to-end multi-perspective matching for entity resolution. In *IJCAI*, 4961–4967. AAAI Press.

Ge, T.; He, K.; Ke, Q.; and Sun, J. 2013. Optimized product quantization. *IEEE TPAMI* 36(4): 744–755.

Gokhale, C.; Das, S.; Doan, A.; Naughton, J. F.; Rampalli, N.; Shavlik, J.; and Zhu, X. 2014. Corleone: hands-off crowd-sourcing for entity matching. In *SIGMOD*, 601–612.

Hernández, M. A.; and Stolfo, S. J. 1995. The merge/purge problem for large databases. In *ACM Sigmod Record*, volume 24, 127–138. ACM.

Ioffe, S.; and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*, 448–456.

Kenig, B.; and Gal, A. 2013. MFIBlocks: An effective blocking algorithm for entity resolution. *Information Systems* 38(6): 908–926.

Khattab, O.; and Zaharia, M. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *SIGIR*, 3948. New York, NY, USA: ACM.

Kim, Y. 2014. Convolutional Neural Networks for Sentence Classification. In *EMNLP*, 1746–1751.

Konda, P.; Das, S.; Suganthan GC, P.; Doan, A.; Ardalan, A.; Ballard, J. R.; Li, H.; Panahi, F.; Zhang, H.; Naughton, J.; et al. 2016. Magellan: Toward building entity matching management systems. *VLDB* 9(12): 1197–1208.

Li, B.; Wang, W.; Sun, Y.; Zhang, L.; Ali, M. A.; and Wang, Y. 2020. GraphER: Token-Centric Entity Resolution with Graph Convolutional Neural Networks. In *AAAI*, 8172–8179.

Li, Y.; Li, J.; Suhara, Y.; Doan, A.; and Tan, W. 2021. Deep Entity Matching with Pre-Trained Language Models. In *VLDB*.

Liu, H.; Wang, R.; Shan, S.; and Chen, X. 2016. Deep supervised hashing for fast image retrieval. In *CVPR*, 2064–2072.

Loshchilov, I.; and Hutter, F. 2019. Fixing weight decay regularization in adam. *ICLR* .

Michelson, M.; and Knoblock, C. A. 2006. Learning blocking schemes for record linkage. In *AAAI*, volume 6, 440–445.

Mudgal, S.; Li, H.; Rekatsinas, T.; Doan, A.; Park, Y.; Krishnan, G.; Deep, R.; Arcaute, E.; and Raghavendra, V. 2018. Deep learning for entity matching: A design space exploration. In *SIGMOD*, 19–34. ACM.

Muja, M.; and Lowe, D. G. 2014. Scalable nearest neighbor algorithms for high dimensional data. *IEEE TPAMI* 36(11): 2227–2240.

Papadakis, G.; Alexiou, G.; Papastefanatos, G.; and Koutrika, G. 2015. Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. *VLDB* 9(4): 312–323.

Papadakis, G.; Skoutas, D.; Thanos, E.; and Palpanas, T. 2020. Blocking and Filtering Techniques for Entity Resolution: A Survey. *ACM Computing Surveys* 53(2): 1–42.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. In *NAACL-HLT*, 2227–2237.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP-IJCNLP*, 3973–3983.

Sarawagi, S.; and Bhamidipaty, A. 2002. Interactive deduplication using active learning. In *SIGKDD*, 269–278.

Singh, R.; Meduri, V. V.; Elmagarmid, A.; Madden, S.; Papotti, P.; Quiané-Ruiz, J.-A.; Solar-Lezama, A.; and Tang, N. 2017. Synthesizing entity matching rules by examples. *VLDB* 11(2): 189–202.

Steorts, R. C.; Ventura, S. L.; Sadinle, M.; and Fienberg, S. E. 2014. A comparison of blocking methods for record linkage. In *International Conference on Privacy in Statistical Databases*, 253–268. Springer.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *NIPS*, 5998–6008.

Wang, J.; Kraska, T.; Franklin, M. J.; and Feng, J. 2012. Crowder: Crowdsourcing entity resolution. *VLDB* 5(11): 1483–1494.

Wang, Q.; Cui, M.; and Liang, H. 2015. Semantic-aware blocking for entity resolution. *IEEE TKDE* 28(1): 166–180.

Wang, S.; and Jiang, J. 2017. A compare-aggregate model for matching text sequences. In *ICLR*.

Wang, Z.; Hamza, W.; and Florian, R. 2017. Bilateral multi-perspective matching for natural language sentences. In *IJCAI*, 4144–4150. AAAI Press.

Zhang, W.; Wei, H.; Sisman, B.; Dong, X. L.; Faloutsos, C.; and Page, D. 2020. Autoblock: A hands-off blocking framework for entity matching. In *WSDM*, 744–752.