# Deep Reinforcement Learning for Entity Alignment

**Anonymous ACL submission**

## Abstract

Embedding-based methods have attracted increasing attention in recent entity alignment (EA) studies. Although great promise they can offer, there are still several limitations. The most notable is that they identify the aligned entities based on cosine similarity, ignoring the semantics underlying the embeddings themselves. Furthermore, these methods are shortsighted, heuristically selecting the closest entity as the target and allowing multiple entities to match the same candidate. To address these limitations, we model entity alignment as a sequential decision-making task, in which an agent sequentially decides whether two entities are matched or mismatched based on their representation vectors. The proposed reinforcement learning (RL)-based entity alignment framework can be flexibly adapted to most embedding-based EA methods. Our experiments demonstrate that it consistently advances the performance of several state-of-the-art methods, with a maximum improvement of 31.1% on Hits@1.

## 1 Introduction

Entity alignment (EA) is one of the most crucial tasks in knowledge graph (KG) studies. It aims to seek the potentially aligned entity pairs between two KGs, such that distributed knowledge can be linked for better supporting downstream applications. Generally, a fact in a KG can be represented by a triplet $(e_x^1, r^1, e_y^1)$, where $e_x^1$, $e_y^1$ denote the head and tail entities in the first KG $\mathcal{G}_1$. $r^1$ is the relation connecting them. With a small number of known alignment pairs as anchors, embedding-based entity alignment (EEA) methods can learn the representations of entities belonging to respective KGs in a unified space and exploit the underlying aligned pairs based on the embedding distance. For example, $e_x^2$ will be chosen as target entity for $e_x^1$ if its embedding is closest to the embedding of $e_x^1$ in vector space.

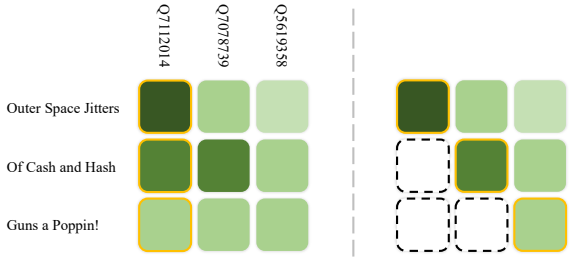


Figure 1: Different evaluation strategies. The ranking strategy (left) heuristically selects the candidate with the largest similarity. The sequential strategy (right) allows each candidate to be matched only once. Deeper color indicates higher similarity. Diagonals are correct matches. Cells with yellow borders are the selected entities, while those with dotted borders denote the excluded entities.

Although recent EEA methods (Chen et al., 2017; Sun et al., 2017, 2018; Guo et al., 2019; Wu et al., 2019) have made great performance improvement, they rarely consider the evaluation process. For example, in Figure 1, all three films are directed by Jules White and have similar casts. This makes the EEA methods confused to discriminate the true aligned entities from other candidates. Current ranking strategy heuristically chooses the nearest entities without considering that some entities have already been matched before. An entity with the largest similarity is not always the true target, especially when this candidate has been matched with other entities. In contrast, we can model entity alignment as a sequential decision-making task, where the agent sequentially decides whether a candidate embedding is aligned with the input one. Then, the environment will exclude the matched candidates in the subsequent decisions.

One issue with the sequential strategy is the accumulated errors. Due to the heterogeneity of KGs, a pair of underlying aligned entities may not share an identical neighborhood. This makes their embeddings not as similar as desired with each other (e.g., "Guns a Poppin!" in Figure 1). But seman-

tics in the embeddings may still indicate the actual target. It is worth estimating the alignment score directly from their embeddings. It is also important to refuse a most likely candidate for maximizing the long-term rewards.

In this paper, we draw on the insights of reinforcement learning (RL) that has recently received great attention in many fields (Mnih et al., 2015; Lillicrap et al., 2016; Silver et al., 2016). With the trained embeddings of any existing EEA models as raw input, we train an agent to find as many alignment pairs as possible to maximize the reward. Meanwhile, we adopt a curriculum learning (Elman, 1993) strategy for the environment to provide candidate entity pairs as observations of increasing difficulty. In sum, our contributions are three-fold:

- We propose to model entity alignment as a sequential decision-making task. To the best of our knowledge, this is the first method that provides a general solution to improve the evaluation strategy for the EEA task.
- We implement an end-to-end RL-based entity alignment (RLEA) framework to solve the sequential EEA problem. We elaborate an entity alignment environment to sample candidate pairs as observations efficiently. Besides, we design a policy network that takes self-embedding, neighborhood, and long-term rewards into account.
- We conduct extensive experiments to show that RLEA can significantly and consistently improve the state-of-the-art EEA methods.

## 2 Related Work

### 2.1 Embedding-based Entity Alignment

We divide the exiting EEA methods into two categories. The first is based on the well-known KG embedding method TransE (Bordes et al., 2013). TransE models a triplet $(e_x^1, r^1, e_y^1)$ as $\mathbf{e}_x^1 + \mathbf{r}^1 \approx \mathbf{e}_y^1$, with the boldfaced as the corresponding embeddings. Many methods use TransE as the KG embedding model for the EA task: MTransE (Chen et al., 2017) sets a learnable matrix to project the entity embeddings from the source KG to the space of the target KG. Then, the distance among entity embeddings from different KGs can be used to estimate the similarity. This idea is extended by later works, e.g., KDCoE (Chen et al., 2018), SEA (Pei et al., 2019a), and OTEA (Pei et al., 2019b). Specifically, KDCoE learns the triplet embedding model and the description embedding model in a co-training fashion. SEA leverages adversarial learning to learn better projection matrix. It also considers the attribute information. OTEA makes use of optimal transport theories to advance the learning process of MTransE. On the other hand, JAPE (Sun et al., 2017) and IPransE (Zhu et al., 2017) adopt a mapping strategy that utterly different from MTransE. They directly set two entities in a known alignment pair to one embedding vector. Therefore, the vector spaces of two KGs are naturally connected. For example, given a known alignment $(e^1, e^2)$, $e^1$, $e^2$ will be mapped to one embedding vector $\mathbf{e}$.

The other line of EEA research focuses on the design of embedding models. Great efforts were put into graph convolutional networks (GCNs) (Kipf and Welling, 2017), e.g., GCN-Align (Wang et al., 2018), RDGCN (Wu et al., 2019), and graph attention networks (GATs) (Velickovic et al., 2018), e.g., MuGNN(Cao et al., 2019), AliNet (Sun et al., 2020a). Most of them adopt the mapping strategy to map entities in each known pair to one vector to connect two KGs. Therefore, these methods center on the design of different graph network structures, which is out of the discussion of this paper. We refer the readers to (Sun et al., 2020b; Wang et al., 2017) for details.

One unique method, BootEA (Sun et al., 2018), iteratively labels likely entity alignment as training data. BootEA is a powerful method that greatly improved the performance of the basic AlignE model. This bootstrapping method is closely related to RLEA as it also assumes that a candidate entity should not be matched more than once. However, BootEA does not have a learning process. The entity alignment pairs are computed based on the cosine similarity and further threshed by a hyper-parameter to filter out those with low similarity. On the other hand, the bootstrapping algorithm must run with the embedding model iteratively, making BootEA more sensitive to parameter settings. Nevertheless, there is no contradiction in integrating RLEA with BootEA to achieve better performance (please see Section 4.6 for details).

The above methods have different objectives and investigate diverse techniques. However, RLEA only needs their trained embeddings as input data, which is sufficient to achieve much better performance on several datasets.

### 2.2 Reinforcement Learning for Knowledge Graphs

One most relevant work to this paper is CEAFF (Zeng et al., 2021), which also leverages RL algorithms and believes in 1-to-1 alignment. But CEAFF focuses more on generating and integrating different entity features. The RL part is less explored. From its experimental results (Zeng et al., 2021), we can find that RL-based CEAFF only outperformed its heuristic version slightly. Moreover, CEAFF does not provide a general solution for sequential EEA task. It is not applicable to most existing EEA methods.

DeepPath (Xiong et al., 2017) and its followers (Das et al., 2018; Wan et al., 2020) are also well-known RL-based KG embedding methods. They leverage RL agents to continually extend paths for multi-hop reasoning. There are two major differences. First, DeepPath focuses more on the design of the reward function. It takes accuracy, diversity, and efficiency into consideration when estimating an action's reward. However, the design of the environment is relatively straightforward, as the next state is certain after receiving an edge as action. By contrast, the reward in our sequential EA task can be simply assigned by comparing the output action (i.e., *match* or *mismatch*) with the actual label; nevertheless, any valid entity pair can be set as next state. Therefore, we focus more on formalizing this problem and building a proper environment where the agent can explore efficiently.

Additionally, some methods like KAGAN (Qu et al., 2019) only use the policy gradient algorithm (Williams, 1992) to update their network parameters. They do not really learn a policy to solve a sequential decision-making problem. Therefore, we do not review them in this paper.

## 3 Methodology

### 3.1 Preliminaries

Let $\mathcal{G}_1 = \{\mathcal{E}_1, \mathcal{R}_1, \mathcal{T}_1\}$ and $\mathcal{G}_2 = \{\mathcal{E}_2, \mathcal{R}_2, \mathcal{T}_2\}$ be the source and target KGs, with $\mathcal{E}$, $\mathcal{R}$, $\mathcal{T}$ denoting the entity, relation, and triplet sets respectively. The proposed RL-based framework consists of two modules, i.e., the agent and the environment. We use the trained entity embeddings $\mathbf{E}_1$, $\mathbf{E}_2$ of any EEA models as input for the agent. The training set, same to the existing works, is still a small number of known entity alignment $\mathcal{S} \subset \mathcal{E}_1 \times \mathcal{E}_2$ provided by the dataset.

In each training episode, the states and actions are generated by the environment and the agent in an alternative order, i.e., $s_1, a_1, s_2, a_2, ..., s_i, a_i$. We define a state $s$ as a pair of arbitrary entities $[e_x, e_y]$ belonging to respective KGs (we rewrite $[e_x^1, e_y^2]$ as $[e_x, e_y]$ for readability, the same below). An action $a \in \{0, 1\}$ represents the decision of the agent that indicates *match* or *mismatch* for $[e_x, e_y]$. Each state also has a label $l \in \{0, 1\}$, implying the right decision. It is worth noting that an action may still have a positive effect even if it is not equal to the label. For example, an incorrect *match* action can also exclude two wrong entities correctly.

In the following sections, we call the case of $a = 0 \wedge l = 0$ a *true mismatch*, $a = 0 \wedge l = 1$ a *false mismatch*, $a = 1 \wedge l = 0$ a *false match*, and $a = 1 \wedge l = 1$ a *true match*. Therefore, the number of correct aligned entity pairs equals to that of *true match*, which is proportional to the Hits@1 result in the conventional EEA task.

### 3.2 Agent

We start by introducing the agent module, which is modeled by neural networks.

**State** A state $s = [e_x, e_y]$ is given by the environment. We take the following features into consideration: (1) the embeddings of two entities $\mathbf{e}_x$, $\mathbf{e}_y$; (2) the neighbor embedding sets of two entities $\mathbf{N}_x$, $\mathbf{N}_y$; (3) the opponent entity embedding set $\mathbf{O}_y$ of $e_y$. We term the $k$-nearest candidates to $e_x$ except $e_y$ "opponent entities", as they are also possible aligned entities to $e_x$. These entities can provide additional information for refusing or accepting the input entity pair $[e_x, e_y]$.

**Action** An action $a$ is a binary number that represents the agent's choice. The binary schema has two advantages. First, the corresponding best policy can be an easier function to be approximated. Selecting one entity from multiple candidates is significantly more complex than judging a pair of entities, especially for the case of existing thousands of different candidates. On the other hand, the binary schema enables the agent to suspend the current candidate pair. For example, if the agent chooses *mismatch*, the source entity still has a chance to be correctly matched in the following interactions. By contrast, in the classification schema, the agent must select one entity as the final choice.

**Policy** The policy $\pi(a|s, \theta)$ is parameterized by graph neural networks (GNNs), where $\theta$ denotes

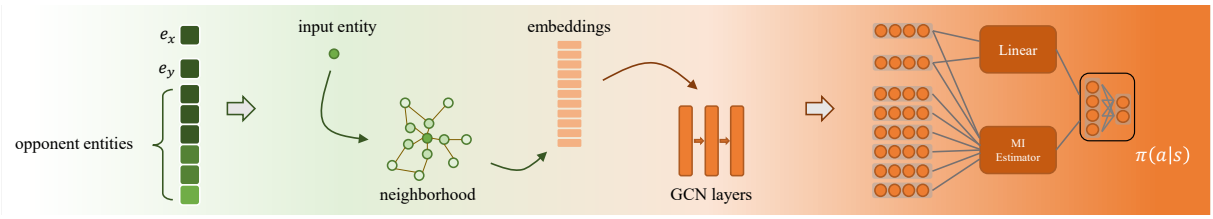


Figure 2: Overview of the policy network. We first use a GNN model to aggregate the neighbor embeddings of each entity. The output representations of $e_x$, $e_y$ are then fed into a linear layer that maps features to an unnormalized estimation of the alignment score. We also leverage a mutual information estimator, which takes the opponent entity representations as negative examples. We combine the output of two types of estimations to obtain the final action distribution.

the parameter set. We illustrate its architecture in Figure 2. Given a state $s = [e_x, e_y]$, we first extract the features by a multi-layer GNN. Here, we use vanilla GCN (Kipf and Welling, 2017) for graph convolution, but other GNN models like GATs (Velickovic et al., 2018) can also be employed. The output embedding of $e_x$ at layer $k$ is defined as:

$$\mathbf{g}_x^k = \sigma\Big(\sum_{e_i \in N(e_x) \cup \{e_x\}} \frac{1}{c_x} \mathbf{W}_g^k \mathbf{g}_i^{k-1}\Big) \tag{1}$$

where $g_x^k$ denotes the output hidden of layer $k$ for $e_x$. $c_x$ is the normalization constant. $\mathbf{W}_g^k$ is the weight matrix at layer $k$. $\sigma(\cdot)$ is the activation function (ReLU (Nair and Hinton, 2010) in our implementation). For the first layer, we set $\mathbf{g}_i^0 = \mathbf{e}_i$, where $\mathbf{e}_i$ denotes the input embedding of $e_i$. GCNs efficiently aggregate the neighborhood and self-information into a single vector, which is supposed to be more robust and informative than directly using the trained embeddings. Furthermore, GCNs also allow RLEA to reweight entity embeddings for sequential EEA. For simplicity, we denote the output of the last GCN layer by $\mathbf{g}_x$.

Next, we use a linear layer to combine the output embeddings $\mathbf{g}_x$, $\mathbf{g}_y$, which can be written as follows:

$$\mathbf{h}_{e_x,e_y} = \sigma(\mathbf{W}_h(\mathbf{g}_x || \mathbf{g}_y) + \mathbf{b}_h), \tag{2}$$

where $||$ is the concatenation operator to concat $\mathbf{g}_x$, $\mathbf{g}_y$ to one hidden vector. $\mathbf{W}_h$ and $\mathbf{b}_h$ are the weight matrix and bias vector, respectively.

We also take the mutual information $I(e_x, e_y)$ (Belghazi et al., 2018) as an additional feature. Unlike the cosine similarity that weights the difference of two vectors at each dimension, mutual information values more on the high-level correlations. Therefore, it is especially appropriate for the EEA task, where two aligned entities may not have identical neighborhoods due to the heterogeneity. Following (van den Oord et al., 2018), we leverage a neural function $f(\mathbf{g}_x, \mathbf{g}_y)$ to estimate the density ratio:

$$f(\mathbf{g}_x, \mathbf{g}_y) = \exp(\mathbf{g}_x^T \mathbf{W}_f \mathbf{g}_y), \tag{3}$$

where $\mathbf{W}_f$ is the weight matrix. As aforementioned, we consider opponent entities a kind of future information to aid the agent in making decisions. This idea can be naturally reified by viewing the opponents as negative examples:

$$\hat{I}_{e_x,e_y} = \frac{f(\mathbf{g}_x, \mathbf{g}_y)}{\sum_{e_i \in O_y \cup \{e_y\}} f(\mathbf{g}_x, \mathbf{g}_i)}. \tag{4}$$

The above equation has a similar form to that used in InfoNCE (van den Oord et al., 2018). But from another aspect, $\hat{I}_{e_x,e_y}$ can be also understood as the probability of outputting the action *match* based on the mutual information estimator (MIE).

Finally, we concatenate all estimates to obtain the final action distribution:

$$\begin{aligned} \mathbf{p}_{e_x,e_y} &= \mathit{Softmax}(\mathbf{W}_p(\mathbf{h}_{e_i,e_j} || \hat{I}_{e_x,e_y})), \\ &= \pi(a|s, \theta) \end{aligned} \tag{5}$$

where $\mathbf{p}_{e_x,e_y}$ is the normalized action distribution . $\mathbf{W}_p$ is the weight matrix.

**Reward** We assign the reward for the given output action $a$ by the following equation:

$$r = \begin{cases} 1, & \text{a } \textit{true match}, \\ -10, & \text{a } \textit{false mismatch}, \\ 0, & \text{elsewise}. \end{cases} \tag{6}$$

The goal of the agent is to maximize the overall reward, i.e., output 1 as much as possible for aligned pairs. Therefore, we set a positive reward for $a = l$ when the input pair match and a severe penalty ($-10$ is most efficient in our implementation) for a *false mismatch*. For other cases, the agent will receive a reward 0, as they do not directly increase or decrease the number of alignment pairs.

**Optimization** We use the policy gradient algorithm REINFORCE (Williams, 1992) to find the parameters leading to a larger reward. To reduce the variance, we employ a baseline function for comparison. Therefore, the gradient at $i$-step in an episode is:

$$\nabla\theta = \alpha\gamma^i\delta\nabla\ln\pi(a|[e_x, e_y]), \tag{7}$$

where $\alpha$ is the learning step-size. $\gamma$ is the discount factor. $\delta$ is the relative advantage of policy $\pi(a|[e_x, e_y])$ than the baseline, i.e., how much better the output action $a$ is than mean or random. It can be defined as follows:

$$\begin{aligned}\delta &= G - \hat{v}([e_x, e_y]) \\ &= \sum_{k=i+1}^{T} \gamma^{k-i-1} r_k - \hat{v}([e_x, e_y]),\end{aligned} \tag{8}$$

where $G$ is the return based on the future rewards. $T$ denotes the episode length. The baseline function $\hat{v}([e_x, e_y])$ in this paper is an estimate of the state value.

### 3.3 Environment

Generally, the environment for an RL task should conform with three basic properties: dependency, dynamics, and difficulty.

**Dependency** The output action may change the later states. For sequential entity alignment, a *true match* will not only yield a correct alignment, but also exclude some plausible candidates for the following judgments, contributing to higher overall reward. Even a *false match* also has its value in filtering out two wrong entities. Therefore, we should consider the long-term dependencies.

To this end, for each entity $e_x$ in $\mathcal{G}_1$, its $k$-nearest entities $e_1, e_2, ..., e_k$ in $\mathcal{G}_2$ are selected as candidates. Those entities are then concatenated with $e_x$ to form $k$ candidate pairs $[e_x, e_1], [e_x, e_2], ..., [e_x, e_k]$. The environment maintains a sequence $c_1, c_2, ..., c_j$, in which each element is such a candidate pair. At the $i$-th step, the environment pops a candidate pair $[e_x, e_y]$ as $s_i$. If it receives an action $a_i = 1$ from the agent, all candidate pairs containing $e_x$ or $e_y$ will be removed from the sequence.

**Dynamics** The environment is usually dynamic. The state-action sequences are different in different episodes. A dynamic environment makes the agent capable of capturing the general rules of the game, which is crucial to avoid overfitting. For the EEA task, if the state sequence is constant at each training episode, the agent will fit this sequence. However, the states are entirely different at the testing phase.

To ensure the dynamic property, we set a skip rate $p_s$. The environment randomly skips a candidate pair with probability $p_s$ and then pops the next pair. Therefore, the length and elements of the state sequence change in each episode.

**Difficulty** Often, the difficulty of a game is improved gradually as step number grows. For example, the health and speed of enemies in video games usually increase over game time. On the other hand, it is also a general strategy to break down complex knowledge by a sequence of learning episodes of increasing difficulty, which is known as curriculum learning (Elman, 1993).

For sequential entity alignment, the difficulty of a candidate pair can be estimated based on the cosine similarity of the two entities and their label, which can be written as follows:

$$\begin{aligned}d(e_x, e_y) = &\, l(C_{e_x,e_{max}} - C_{e_x,e_y}) \\ &+ (1-l)(\tau - C_{e_x,e_{max}} + C_{e_x,e_y})\end{aligned} \tag{9}$$

where $C_{e_x,e_y}$ is the cosine similarity between $e_x$ and $e_y$. $e_{max}$ denotes the entity with the largest similarity to $e_x$. We use the difference between $C_{e_x,e_{max}}$ and $C_{e_x,e_y}$ as the basis to estimate the extent, and the label $l$ as the sign. When $l = 1$, i.e., the first term in Eq. (9), a large difference between $C_{e_x,e_y}$ and $C_{e_x,e_{max}}$ will result in high difficulty because $e_x$, $e_y$ may be too dissimilar with each other. The situation is reversed for $l = 0$. We add a hyper-parameter $\tau$ to balance difficulty scores between these two cases.

Then, we can sort candidate pairs by the difficulty in ascending order, such that the agent will always start from the relatively easier states. However, this operation is inapplicable to the testing set where the label information is unknown. To mitigate this problem, we propose a curriculum learning strategy. We do not directly sort the candidate pairs by difficulty score. Instead, we sort them based on the cosine similarity and re-weight the skip rate $p_s$ for each pair by its normalized difficulty score. Therefore, for each episode, the agent will start from pairs with high similarity, and the more difficult states will be skipped with larger

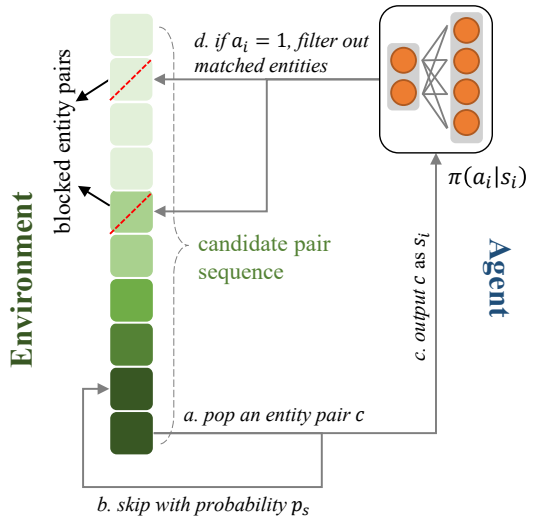


Figure 3: Illustration of how the environment interacts with the agent in RLEA.

probabilities. As the policy is optimized, we gradually decrease $p_s$ to approximate the testing environment. The final state sequence shall have a similar arrangement to that at the testing phase.

The skip rate $p_s^{i,t}$ at the $i$-th step in episode $t$ can be written as:

$$p_s^{i,t} = max(p_s^{min}, \eta^{t-1} p_s d_i), \tag{10}$$

where $p_s^{min}$ is the minimal skip rate to ensure the dynamic property. $p_s$ is the basic skip rate. $d_i$ denotes the difficulty of state $s_i$ at $i$-th step. As episode number grows, $p_s^{i,t}$ decreases with discount factor $\eta$ exponentially until it meets the lower bound $p_s^{min}$.

We illustrate how the environment collaborates with the agent in Figure 3: a. the environment pops an entity pair $c$ from the candidate pair sequence; b. this entity pair may be skipped with probability $p_s$ (Equation (10)); c. the non-skipped pair is outputted by the environment as $s_i$; d. the agent takes $s_i$ as input, and its output action changes the candidate pair sequence reversely. The detailed implementation can be found in Appendix A.

# 4 Experiment

We conducted experiments to verify the effectiveness of the proposed RL-based framework. The trained entity embeddings were obtained from the OpenEA project [1]. The source code was uploaded and will be available online.

[1] https://github.com/nju-websoft/OpenEA

## 4.1 Dataset Settings

We used the 15K benchmark proposed by OpenEA. It consists of four subsets: EN-FR, EN-DE, D-W, and D-Y. The former two are cross-lingual datasets, where EN, FR, DE denote English, French, and German versions of DBpedia, respectively. The latter two are cross-source datasets, where D, W, Y denote DBpedia (Auer et al., 2007), WikiData (Vrandečić and Krötzsch, 2014), and Yago (Fabian et al., 2007), respectively. We used "V1" subsets that has similar distributions to original KGs. Please refer to (Sun et al., 2020b) for detailed statistics.

## 4.2 Compared Methods

We select the following methods as baselines:

- JAPE (Sun et al., 2017), which learns attribute embeddings and relational embeddings jointly for EEA.
- SEA (Pei et al., 2019a), which adopts adversarial learning to learn the projection matrix.
- RSN (Guo et al., 2019), which leverages recurrent neural networks (RNNs) (Williams and Zipser, 1989) to learn KG embeddings.
- RDGCN (Wu et al., 2019), which uses GCNs to capture the neighborhood information into entity embeddings.

We also design a basic sequential strategy called *Seq* for comparison. We follow the algorithm used in BootEA (Sun et al., 2018) to implement it. Entity pairs with similarity above a predefined threshold are regarded as *match*, or the algorithm randomly chooses actions based on cosine similarity.

## 4.3 Main Results

The results on four datasets are shown in Table 1. Orig denotes the original results of the EEA methods. We can observe that RLEA significantly improved the performance of all baseline methods, including the best-performing one, RDGCN. Therefore, we believe that RLEA provides a better way to exploit aligned entity pairs from embeddings than the widely-used heuristic strategy.

Specifically, the performance improvement of JAPE is most notable, with 30.4% and 31.1% increases on EN-FR and D-Y, respectively. Although RSN has minimal performance increase, the difference is still significant. Similarly, RDGCN also achieved better performance, leading to a new state-of-the-art on the benchmark.

Table 1: Hits@1 results on four datasets (5-fold cross-validation).

| Methods | EN-FR | | | EN-DE | | | D-W | | | D-Y | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Orig | Seq | RLEA | Orig | Seq | RLEA | Orig | Seq | RLEA | Orig | Seq | RLEA |
| JAPE (Sun et al., 2017) | .247 | .291 | **.322** | .307 | .332 | **.336** | .259 | .279 | **.301** | .463 | .547 | **.607** |
| SEA (Pei et al., 2019a) | .280 | .317 | **.365** | .530 | .556 | **.571** | .360 | .359 | **.414** | .500 | .564 | **.643** |
| RSN (Guo et al., 2019) | .393 | .410 | **.429** | .587 | .614 | **.634** | .441 | .466 | **.493** | .514 | .546 | **.566** |
| RDGCN (Wu et al., 2019) | .755 | .801 | **.830** | .830 | .861 | **.878** | .515 | .517 | **.541** | .931 | .951 | **.974** |

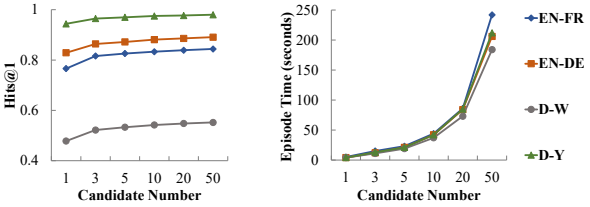


Figure 4: Hits@1 and episode time w.r.t. candidate number on four datasets.

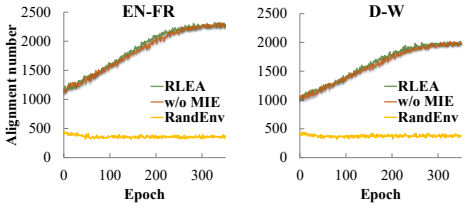


Figure 5: Alignment number w.r.t. episode number, on EN-FR and D-W datasets.

With the basic sequential strategy Seq, four baseline methods also achieved better Hits@1 results on most datasets except D-W. This observation empirically proves the advantages of modeling EA as a sequential decision-making task.

Note that RLEA also has its limitations. We find there exists a dataset bias. The performance improvement on EN-FR dataset is notable, but that on D-W is less significant. We believe that the sequential evaluation process might cause this bias. For instance, Seq also got worse performance than the original method SEA on D-W. We leave how to mitigate this problem in future work.

### 4.4 Influence of Candidate Number

In RLEA, the candidate number for each entity is an important hyper-parameter as it decides the length of the candidate sequence. A large value means covering more correct alignment pairs as well as more plausible pairs. Therefore, it is necessary to study how this hyper-parameter influences the performance of RLEA.

We used the embeddings of the best-performing method RDGCN as input in this experiment. As shown in the left of Figure 4, the Hits@1 results on four datasets gradually increase with candidate number from 1 to 5, but converge after 10. When candidate number was set to 1, for each entity, only the pair with the highest similarity was added to the sequence, resulting in similar or even worse results compared with the original method. For example, on D-W, the hits@1 of RLEA is 0.478, significantly below that of the original RDGCN (0.541). As the candidate number increased, more aligned pairs were added to the sequence, the performance improved steadily. It then gets saturated due to more unaligned pairs were also added to the sequence. On the right of Figure 4, we show the runtime of one testing episode w.r.t. candidate number, which, however, grows exponentially. This observation suggests that setting a large candidate number is computationally expensive. Therefore, we decide to use the top-10 candidates in our implementation, for sake of performance and efficiency.

### 4.5 Effectiveness of Modules

We conducted experiments to verify the effectiveness of mutual information estimator (MIE) and the proposed environment. We developed two variants of RLEA: (1) RLEA without MIE (denoted as w/o MIE), and (2) RLEA with a random environment (RandEnv). The random environment still maintains a candidate pair sequence but does not have the difficulty and skipping settings. All candidate pairs in the sequence are randomly reset at the start of each episode.

We compare the results in Figure 5, from which we find that the agent does not work in the random environment on all datasets. The alignment number even slowly decreases during training. This is because that the state sequence in the random environment changes irregularly. The agent fails to establish an effective policy to maximize the reward for all episodes. Furthermore, the random environment does not have a curriculum learning strategy to help the agent study from easy to hard. Therefore, the agent is not able to capture the general rules in the random environment.

Table 2: Comparing RLEA with conventional methods.

| Methods | EN-FR | | | EN-DE | | | D-W | | | D-Y | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | F1-score | Precision | Recall | F1-score | Precision | Recall | F1-score | Precision | Recall | F1-score |
| LogMap | .818 | .729 | .771 | .925 | .725 | .813 | - | - | - | .960 | .943 | .951 |
| PARIS | **.907** | **.900** | **.903** | **.938** | **.933** | **.935** | **.746** | **.723** | **.734** | .875 | .868 | .872 |
| OpenEA | .755 | .755 | .755 | .830 | .830 | .830 | .572 | .572 | .572 | .931 | .931 | .931 |
| RLEA | .830 | .830 | .830 | .878 | .878 | .878 | .611 | .611 | .611 | **.974** | **.974** | **.974** |

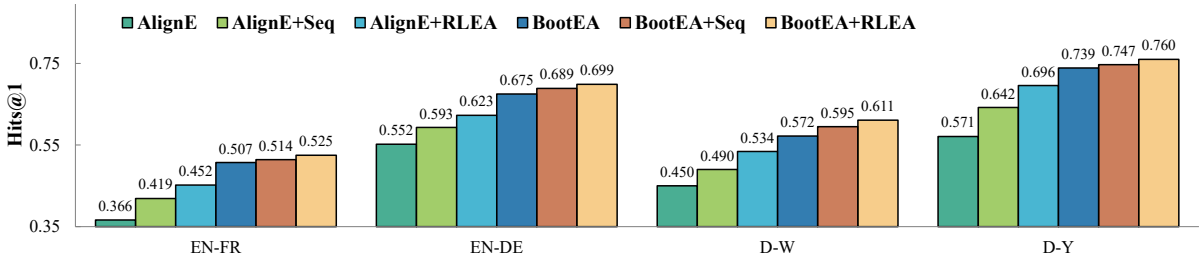


Figure 6: A comparison between RLEA and BootEA.

On the other hand, we find that MIE slightly improves the performance. It does not have a significant advantage over the final reward or alignment number. This may be because that the output embeddings of GNNs have already included sufficient information to judge entity pairs. Nevertheless, the estimation provided by MIE helps the agent find the best policy rapidly, which is crucial when applying to larger datasets. A more detailed version of Figure 5 is shown in Appendix B, from which we can obtain the consistent observations.

### 4.6 A Comparison of RLEA and BootEA

BootEA (Sun et al., 2018) is a bootstrapping method that iteratively labels possible entity alignment as training data. Like RLEA, BootEA assumes a candidate entity should not be aligned twice. Therefore, it is interesting to compare and discuss these two methods.

We illustrate the experimental results on four datasets in Figure 6. AlignE is a variant of BootEA without bootstrapping process. We first compare the left four columns. Obviously, BootEA (4th column) has a better effect in improving the performance of AlignE, as it directly participates in training AlignE by iteratively adding plausible alignment pairs. In contrast, Seq (2nd column) and RLEA (3rd column) only use the trained embeddings as input and do not modify the embeddings or training procedure. They are thus more extensible and applicable to arbitrary EEA methods.

In fact, it is no contradiction to integrate these two types of methods. The performance improvement (5th and 6th columns) is still significant and consistent on all four datasets.

### 4.7 Competing with Conventional Methods

There has always been an argument about the practical use of EEA. Most EEA methods are end-to-end and easy to be deployed. The performance also improves when new models are developed. However, a significant performance gap still exists between EEA methods and those conventional methods like Paris (Suchanek et al., 2012) and LogMap (Jiménez-Ruiz and Grau, 2011). We show in Table 2 that RLEA with the embeddings of best EEA methods as input can narrow this gap and even outperform the conventional methods on some datasets. As shown in Table 2, PARIS is the best method that outperformed others on all datasets except D-Y. However, The second method changed from LogMap to RLEA. We can find that RLEA not only outperformed LogMap on EN-FR, but also achieved the best performance on D-Y.

We should notice that the alignment pairs exploited by EEA methods and conventional methods are not all overlapped (Sun et al., 2020b). It is possible to integrate them to achieve better performance (Sun et al., 2017, 2020b). In this sense, RLEA is also the best choice to be combined with conventional methods.

## 5 Conclusion and Future Work

In this paper, we proposed an RL-based entity alignment framework, which can advance most existing EEA methods without modifying their parameter settings or infrastructures. Our experiments demonstrate consistent and significant improvement on all baseline methods. We plan to study how to jointly train EEA methods and RLEA for further improvement in future work.

## References


Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. 2007. Dbpedia: A nucleus for a web of open data. In *ISWC*.

Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeswar, Sherjil Ozair, Yoshua Bengio, R. Devon Hjelm, and Aaron C. Courville. 2018. Mutual information neural estimation. In *ICML*.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *NeurIPS*.

Yixin Cao, Zhiyuan Liu, Chengjiang Li, Zhiyuan Liu, Juanzi Li, and Tat-Seng Chua. 2019. Multi-channel graph neural network for entity alignment. In *ACL*.

Muhao Chen, Yingtao Tian, Kai-Wei Chang, Steven Skiena, and Carlo Zaniolo. 2018. Co-training embeddings of knowledge graphs and entity descriptions for cross-lingual entity alignment. In *IJCAI*.

Muhao Chen, Yingtao Tian, Mohan Yang, and Carlo Zaniolo. 2017. Multilingual knowledge graph embeddings for cross-lingual knowledge alignment. In *IJCAI*.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. 2018. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *ICLR*.

Jeffrey L Elman. 1993. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99.

MS Fabian, Kasneci Gjergji, WEIKUM Gerhard, et al. 2007. Yago: A core of semantic knowledge unifying wordnet and wikipedia. In *WWW*.

Lingbing Guo, Zequn Sun, and Wei Hu. 2019. Learning to exploit long-term relational dependencies in knowledge graphs. In *ICML*.

Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. 2011. Logmap: Logic-based and scalable ontology matching. In *ISWC*.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *ICLR*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533.

Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML*.

Shichao Pei, Lu Yu, Robert Hoehndorf, and Xiangliang Zhang. 2019a. Semi-supervised entity alignment via knowledge graph embedding with awareness of degree difference. In *WWW*.

Shichao Pei, Lu Yu, and Xiangliang Zhang. 2019b. Improving cross-lingual entity alignment via optimal transport. In *IJCAI*.

Meng Qu, Jian Tang, and Yoshua Bengio. 2019. Weakly-supervised knowledge graph alignment with adversarial learning. *CoRR*, abs/1907.03179.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489.

Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. 2012. PARIS: Probabilistic alignment of relations, instances, and schema. *PVLDB*.

Zequn Sun, Wei Hu, and Chengkai Li. 2017. Cross-lingual entity alignment via joint attribute-preserving embedding. In *ISWC*.

Zequn Sun, Wei Hu, Qingheng Zhang, and Yuzhong Qu. 2018. Bootstrapping entity alignment with knowledge graph embedding. In *IJCAI*.

Zequn Sun, Chengming Wang, Wei Hu, Muhao Chen, Jian Dai, Wei Zhang, and Yuzhong Qu. 2020a. Knowledge graph alignment network with gated multi-hop neighborhood aggregation. In *AAAI*.

Zequn Sun, Qingheng Zhang, Wei Hu, Chengming Wang, Muhao Chen, Farahnaz Akrami, and Chengkai Li. 2020b. A benchmarking study of embedding-based entity alignment for knowledge graphs. *PVLDB*.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57.

Guojia Wan, Shirui Pan, Chen Gong, Chuan Zhou, and Gholamreza Haffari. 2020. Reasoning like human: Hierarchical reinforcement learning for knowledge graph reasoning. In *JICAI*, pages 1926–1932.

Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. 2017. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29.

Zhichun Wang, Qingsong Lv, Xiaohan Lan, and Yu Zhang. 2018. Cross-lingual knowledge graph alignment via graph convolutional networks. In *EMNLP*.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8:229–256.

Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1.

Yuting Wu, Xiao Liu, Yansong Feng, Zheng Wang, Rui Yan, and Dongyan Zhao. 2019. Relation-aware entity alignment for heterogeneous knowledge graphs. In *IJCAI*.

Wenhan Xiong, Thien Hoang, and William Yang Wang. 2017. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *EMNLP*.

Weixin Zeng, Xiang Zhao, Jiuyang Tang, Xuemin Lin, and Paul Groth. 2021. Reinforcement learning-based collective entity alignment with adaptive features. *ACM Trans. Inf. Syst.*, 39(3):26:1–26:31.

Hao Zhu, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2017. Iterative entity alignment via joint knowledge embeddings. In *IJCAI*.

# A Implementation

## A.1 Algorithm

We show the training procedure of RLEA by Algorithm 1. The input is two KGs, trained embeddings of an arbitrary EEA method, and parameter settings. If the EEA method has projection matrices (Chen et al., 2017; Pei et al., 2019a), the embeddings of $\mathcal{G}_2$ should be projected to the space of $\mathcal{G}_1$ by the corresponding matrix before the training starts. We first initialize all parameters of the policy network. The episode sequence and candidate pair sequence will be reset at the start of each episode. After that, the agent interacts with the environment, which generates a state-action sequence. We then use REINFORCE algorithm to update the policy network with the generated sequence.

**Algorithm 1:** RLEA

**Input:** Two KGs $\mathcal{G}_1, \mathcal{G}_2$, entity embeddings $\mathbf{E}_1, \mathbf{E}_2$, and number of episodes $N$.
**Output:** The policy $\pi(a|s, \theta)$.

```
Initialize the policy parameter θ;
for t=1:N do
    Reset episode sequence to empty and initialize candidate pair sequence;
    repeat
        Pop a candidate pair c from candidate pair sequence;
        Calculate the skip rate p_s with Equation (10);
        μ ← RandInt(0, 1, p_s);
        if μ = 1 then continue ;
        else s ← c ;
        a ~ π(a|s, θ);
        if a = 1 then Update candidate pair sequence according to c ;
        Add s, a to episode sequence;
    until All entities are matched or candidate pair sequence is empty.;
    for each step in the episode sequence do
        Update the policy parameter θ according to Equation (7).
    end
end
Output π(a|s, θ).
```

## A.2 Parameter Settings

For each EEA method, we directly used their trained entity embeddings as input and did not modify these vectors during training. The embedding-size was identical to that used in OpenEA (Sun et al., 2020b). The number of training episodes was set to 500, and the learning step-size was set to 0.0001. The candidate pair number for each entity was set to 10.

# B Detailed Results of Ablation Study

The detailed results of RLEA and its two variants are shown in Figure 7. Overall, the full RLEA still has the best performance and training speed, especially on D-Y dataset. The method without MIE also have competitive performance on four datasets, which demonstrates the effectiveness of the RL-based sequential EEA.

From the bottom sub-figures, we find that the agent tries to find a policy to achieve high rewards in the random environment. However, 0 is almost the best reward it can get. The agent fails to establish a good policy in this dynamic environment.

# C Results on OpenEA 100K datasets

As shown in Table 3, the Hits@1 results on OpenEA 100K datasets are consistent with those on 15K datasets. RLEA still outperformed the baselines on four datasets. We did not consider RDGCN and RSN in this experiment, as they can only be trained on CPUs (confirmed from the authors of OpenEA).

Table 3: Hits@1 results on OpenEA 100K datasets.

| Methods | EN-FR | | | EN-DE | | | D-W | | | D-Y | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Orig | Seq | RLEA | Orig | Seq | RLEA | Orig | Seq | RLEA | Orig | Seq | RLEA |
| JAPE (Sun et al., 2017) | .165 | .172 | **.197** | .152 | .162 | **.169** | .211 | .229 | **.257** | .287 | .308 | **.323** |
| SEA (Pei et al., 2019a) | .225 | .229 | **.261** | .341 | .345 | **.376** | .291 | .293 | **.338** | .490 | .525 | **.545** |

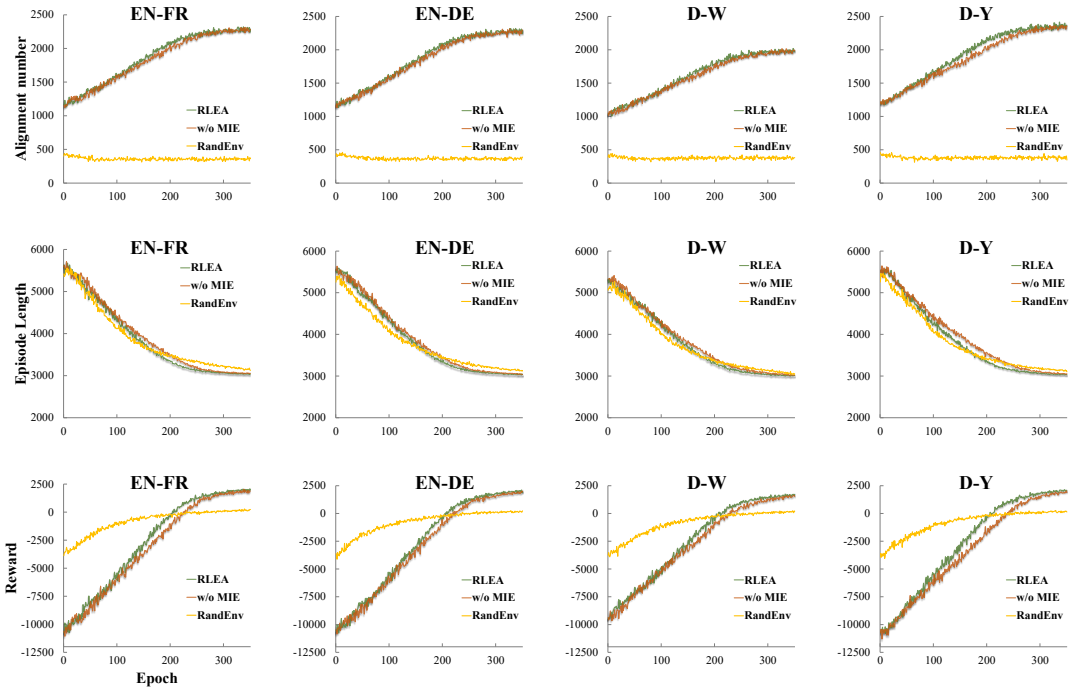


Figure 7: Alignment number, episode length, and reward w.r.t. episode number, on four datasets.