

Neural Recurrent Structure Search for Knowledge Graph Embedding

Yongqi Zhang¹, Quanming Yao², Lei Chen¹

¹Hong Kong University of Science and Technology, Hong Kong

²Paradigm Inc, China

correspondence to: yaoquanming@4paradigm.com

Abstract

Knowledge graph (KG) embedding is a fundamental problem in mining relational patterns. It aims to encode the entities and relations in KG into low dimensional vector space that can be used for subsequent algorithms. Lots of KG embedding models have been proposed to learn the interactions between entities and relations, which contain meaningful semantic information. However, structural information, which encodes local topology among entities, is also important to KG. In this work, we propose *S2E* to distill structural information and combine it with semantic information for different KGs as a neural architecture search (NAS) problem. First, we analyze the difficulty of using a unified model to solve the distillation problem. Based on it, we define the path distiller to recurrently combine structural and semantic information along relational paths, which are sampled to preserve both local topologies and semantics. Then, inspired by the recent success of NAS, we design a recurrent network based search space for specific KG tasks, and propose a natural gradient (NG) based search algorithm to update architectures. Experimental results demonstrate that the searched models by our proposed *S2E* outperform human-designed ones, and the NG based search algorithm is efficient compared with other NAS methods. Besides, our work is the first NAS method for RNN that can search architectures with better performance than human-designed models.

1 Introduction

Knowledge Graph (KG) (Auer et al. 2007; Socher et al. 2013; Wang et al. 2017), as a special kind of graph with lots of relational facts, is important to data mining and machine learning. In KGs, each relational fact is represented as a triplet in the form of (*subject entity*, *relation*, *object entity*), and is abbreviated as (s, r, o) . KGs have gradually been employed in many knowledge-driven tasks, e.g., structured search (Dong et al. 2014), question answering (Lukovnikov et al. 2017) and recommendation (Zhang et al. 2016). However, the discrete representation of triplets can not support these applications well (Wang et al. 2017). KG embedding methods, which map the discrete entities and relations into continuous vector spaces, have recently emerged and been developed as a promising direction serving this purpose (Bordes et al. 2013; Guo, Sun, and Hu 2019; Sun et al. 2018;

Wang et al. 2017). The vector representations, i.e. embeddings, can preserve information in original triplets and can be used together with other machine learning techniques (Lukovnikov et al. 2017; Zhang et al. 2016).

As a relational graph, the *semantic information*, which models the interactions between entities and relations, is the main focus in KG. When learning KG embedding, the most fundamental issue is how to preserve information in the relational triplets (s, r, o) 's. TransE (Bordes et al. 2013), a representative embedding model, interprets the triplet (s, r, o) as a translation r from subject s to object o , i.e. the embeddings satisfy $s + r \approx o$. Following works, such as TransD (Ji et al. 2015), ComplEx (Trouillon et al. 2017), ConvE (Dettmers et al. 2017), etc., also learn the embeddings based on the single triplet (s, r, o) . PTransE (Lin et al. 2015) extends TransE with multi-hop relation translation. By using manually selected paths, PTransE can learn compositional relation patterns, but fail to explore the structural information. These models mainly focus on the semantic information.

As a kind of graph structured data, the *structural information*, which encodes local topology among entities, gradually arises attention. GCN-Align (Wang et al. 2018) learns cross lingual alignment by using graph convolutional network to leverage the structural information. However, leaving semantic information alone prevents this model from fully capturing properties in KG. More recently, (Guo, Sun, and Hu 2019) proposes recurrent skipping network (RSN) to process the relational paths with RNNs. By sampling paths along the KG and incorporating both entity and relation embeddings, RSN becomes a promising model to exploit structural information along with semantic information. However, it focuses more on learning the structural information along paths and does not adapts well to tasks where semantic information is more important.

Given a KG task or dataset, what compositional relation patterns it has and what structure it forms are diverse (Wang et al. 2017). How to exploit the structural information and combine it with semantic information for a specific KG task is non-trivial. Inspired by the success of neural architecture search (NAS) (Elsken et al. 2019), we propose *S2E* in this paper, which can automatically combine *Structural* and *Semantic* information for KG *Embeddings*. Similar as (Guo,

Table 1: Recurrent function of existing KG embedding models. “ \odot ” is the elementwise multiply, and “ \otimes ” is the Hermitian product (Trouillon et al. 2017) in complex space. σ is a non-linear activation function.

model	semantic	structural
TransE	$\mathbf{v}_t = \mathbf{s}_t + \mathbf{r}_t$	\times
ComplEx	$\mathbf{v}_t = \mathbf{s}_t \otimes \mathbf{r}_t$	\times
PTransE	add $\mathbf{v}_t = \mathbf{h}_t, \mathbf{h}_t = \mathbf{h}_{t-1} + \mathbf{r}_t$	\times
	multiply $\mathbf{v}_t = \mathbf{h}_t, \mathbf{h}_t = \mathbf{h}_{t-1} \odot \mathbf{r}_t$	\times
	RNN $\mathbf{v}_t = \mathbf{h}_t, \mathbf{h}_t = \sigma(\mathbf{W}_1 \mathbf{r}_t + \mathbf{W}_2 \mathbf{h}_{t-1} + \mathbf{b})$	\times
ChainR	$\mathbf{v}_t = \mathbf{h}_t$	$\mathbf{h}_t = \sigma(\mathbf{W}_1 \mathbf{h}_{t-1} + \mathbf{W}_2 \mathbf{s}_t + \mathbf{W}_3 \mathbf{r}_t + \mathbf{b})$
RSN	$\mathbf{v}_t = \mathbf{W}_5 \mathbf{s}_t + \mathbf{W}_6 \mathbf{h}_t$	$\mathbf{h}_t = \sigma(\mathbf{W}_3 \sigma(\mathbf{W}_1 \mathbf{h}_{t-1} + \mathbf{W}_2 \mathbf{s}_t + \mathbf{b}_1) + \mathbf{W}_4 \mathbf{r}_t + \mathbf{b}_2)$
S2E (NAS)	a recurrent network searched by natural gradient descent	

Sun, and Hu 2019), we generate paths to extract both structural information and semantic information in KGs. Based on the paths, we define the path distiller, i.e. a recurrent network, to distill the structural information in the paths and combine it with the semantic information. By reviewing lots of human-designed KG embedding models, we set up the search space for the recurrent network, which is quite different from the general NAS for RNN problem. In order to search efficiently, we propose a natural gradient based search algorithm for the network architecture. Extensive experiments on entity alignment and link prediction tasks show the effectiveness of the designed search space and efficiency of the search algorithm. Our searched recurrent networks are novel and can well adapt to different KG tasks by more effectively combining the structural and semantic information. Contributions of our work are summarized as follows:

- We analyze the difficulty and importance of distilling structural information and combining it with semantic information for KGs. Based on the analysis, we formulate the above problem as an NAS problem which targets at searching recurrent networks specific to the KG domain.
- We propose a domain-specific search space for the recurrent network. Different from searching RNN cells, the recurrent network in our space is specifically designed for KG tasks and covers many human-designed embedding models. This also throws light upon designing of future embedding models.
- We identify the problems of adopting one-shot architecture search in our search space. The problems are rarely discussed in NAS literature. and motivate us to design a natural gradient based searching algorithm to give pseudo gradient for architecture search, which is much more efficient than existing derivative-free NAS methods.
- We conduct experiments on entity alignment and link prediction tasks. Empirical experiments demonstrate that the searched models by S2E outperform human-designed ones, and the search algorithm is more efficient compared with other NAS baselines.

2 Related Works

Before we present the related work, we first give the notations used in this paper. We denote vectors by lowercase boldface, and matrix by uppercase boldface. Knowledge Graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S})$ is defined by the set of entities \mathcal{E} ,

relations \mathcal{R} and triplets \mathcal{S} . A single triplet $(s, r, o) \in \mathcal{S}$ represents a relation r that links from the subject entity s to the object entity o . The embeddings in this paper are denoted as boldface letters of indexes, e.g. $\mathbf{s}, \mathbf{r}, \mathbf{o}$ are embeddings of s, r, o respectively. Vectors are denoted by lowercase boldface, and matrices by uppercase boldface.

2.1 Knowledge Graph Embedding

Knowledge graph (KG) embedding aims to embed entities and relations into low-dimensional vector space, while preserving relevant properties in original KG. Given a single triplet (s, r, o) , TransE (Bordes et al. 2013), and its following works TransD (Ji et al. 2015), ComplEx (Trouillon et al. 2017), ConvE (Dettmers et al. 2017), etc., interpret the interactions between embeddings \mathbf{s}, \mathbf{r} and \mathbf{o} in different ways. PTransE (Lin et al. 2015) manually selects some composite relations with semantic reliability in KG and models the sequence of relations (r_1, r_2, \dots, r_n) instead of the single triplet. However, these works focus on learning semantic information, i.e. interactions of entities with single or compositional relations.

Definition 1 (Relational Path (Guo, Sun, and Hu 2019)). A path (of length L) is formed by a set of triplets $(s_1, r_1, o_1), (s_2, r_2, o_2), \dots, (s_L, r_L, o_L)$ where $o_i = s_{i+1}$ for all $i = 1 \dots L - 1$.

Structural information, which means the topology among entities, gradually arises attention in learning KG embeddings. GCN-Align (Wang et al. 2018), incorporates graph convolutional network (GCN) (Kipf and Welling 2016) to aggregate neighbors of each entity to capture structural information. However, ignoring semantic information among the triplets leads to their inferior performance. IPTransE (Zhu et al. 2017) iteratively aligns entities in two KGs based on PTransE to implicitly leverage the internal structural information. More recently, recurrent skipping network (RSN) (Guo, Sun, and Hu 2019) is proposed to exploit long-term relational dependencies in KGs. The idea is to model relational paths (Definition 1), which contains both structural and semantic information, instead of the single triplet. By processing paths with a basic RNN and a skip connection, RSN incorporates the structural information well, but is still limited in learning semantic information. The expressions of several human-designed models are given in Tab. 1, and a graphical illustration of TransE, PTransE and RSN model is shown in Fig. 1. They have distinct connection patterns and

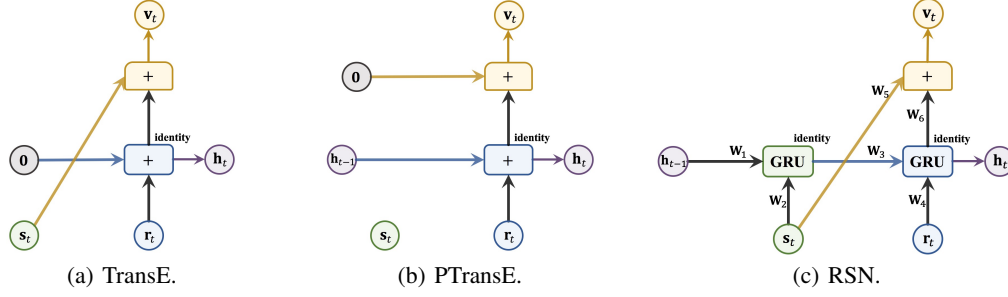


Figure 1: Graphical illustration of TransE, PTransE, RSN. TransE is a single-triplet model thus it is non-recurrent. PTransE recurrently processes the path without entity embeddings. RSN uses full RNN and a skip connection. Each edge is an identity mapping, except an explicit mark of weighted mapping with the weight matrix W_i , where $i = 1, 2, \dots, 6$ for the six edges. Activation functions of OP1 and OP2 are shown in the upper right corner of themselves.

composition operators, leading to different ways in dealing with structural and semantic information.

2.2 Neural Architecture Search (NAS)

NAS is a promising way to design reasonable neural network architectures for different tasks. It has two important perspectives (Zoph and Le 2017; Elsken, Metzen, and Hutter 2019; Hutter, Kotthoff, and Vanschoren 2018): i). *search space*: it defines which architectures can be represented in principle like CNN or RNN. The search space should be carefully designed for specific scenarios and cover human wisdom as special cases. ii). *search algorithm*: universal optimization tools are generally inefficient to find good architectures. In order to make optimization fast, algorithms should be designed to utilize properties of the search space and domain information of the NAS problem.

The search space of CNN has been developed from searching basic convolution units for the whole architecture (Zoph and Le 2017), to repeated cells with fixed macro architecture based on human-designed networks (Liu, Simonyan, and Yang 2019). Many architectures have been searched to outperform CNNs in literature. However, designing the search space for RNN attracts little attention and the searched architectures do not outperform human-designed ones (Liu, Simonyan, and Yang 2019).

Recently, one-shot architecture search methods, e.g., DARTS (Liu, Simonyan, and Yang 2019), ASNG (Akimoto et al. 2019) and SNAS (Xie et al. 2018), have become the most popular NAS methods that can efficiently find good architectures. These methods directly construct a supernet, which contains all possible architectures spanned by selected operations, and jointly optimize network weights and architectures' parameters by stochastic gradient descent. However, their success counts on two conditions. First, network weights can be shared, which means the performance of direct training the supernet is not bad (Xie et al. 2018) and removing a good operation can deteriorate the supernet's performance more than a bad operation (Bender et al. 2018). Second, the validation measurement needs to be differentiable. Otherwise, even with differentiable relaxation to architectures, we cannot get gradient information to update architectures (Akimoto et al. 2019).

3 Search Problem

As discussed in Sec.2.1, *structural information* and *semantic information* are both important in KG, and they can be captured by relational paths (Definition 1). However, given a specific KG dataset or task, the interactions between entities and relations, and the degree distribution of entities are complex (Perozzi, Al-Rfou, and Skiena 2014; Nickel et al. 2016; Wang et al. 2017). Designing a model that can adapt well to different tasks and datasets is non-trivial. Single triplet models, e.g., TransE and ComplEx, process each triplet in the paths individually, thus no structural information is utilized. PTransE aggregates relations rather than entities, and thus mainly focus on semantics. ChainR and RSN recurrently process the entities and relations along paths. However, they emphasize too much on the structural information and fail to learn the semantics well. Therefore, we are motivated to search the model architectures to adaptively distill structural information and combine it with semantic information.

3.1 Search Objective

The key problem now becomes *how to leverage the relational path (Definition 1) to distill the structural information and to combine with semantic information*. Based on the existing embedding models, and inspired by the success of RSN (Guo, Sun, and Hu 2019) as well as the fact that a path is a sequence of many triplets, which is similar as that a sentence is a sequence of many words (Sundermeyer, Schlüter, and Ney 2012), we model the paths as a recurrent function in Definition 2.

Definition 2 (Path Distiller). *A path distiller processes the embeddings of s_1, r_1 to s_L, r_L recurrently. In each recurrent step t , the distiller combines embeddings of s_t, r_t and a distillation of preceding information h_{t-1} to get an output v_t . The distiller is formulated as a recurrent function*

$$[v_t, h_t] = f(s_t, r_t, h_{t-1}), \quad t = 1 \dots L, \quad (1)$$

where h_t 's are hidden state of recurrent steps and $h_0 = s_1$. The output v_t should approach object entity o_t .

Specifically, at each step t , we focus on one triplet (s_t, r_t, o_t) to preserve the semantic information. The subject entity embedding s_t and relation embedding r_t are the inputs

to the model. We use hidden state \mathbf{h}_t to combine structural and semantic information along the path. Besides, we collect an output \mathbf{v}_t in each step and use it to predict object entity \mathbf{o}_t in the current step. Therefore, how to form the recurrent function f for certain KG tasks is the key issue. As it is difficult to design an f that can adapt well to various KG tasks and datasets and motivated by the recent success of NAS (Elsken, Metzen, and Hutter 2019) in searching neural network architectures, we propose to search the recurrent function f as a RNN. The network here is not a general RNN but one specific to KG embedding tasks. Therefore, searching f is modeled as an NAS problem in Definition 3. We use α to denote the architecture of the recurrent network f .

Definition 3 (NAS Problem). Let the training set be \mathcal{G}_{tra} and validation set be \mathcal{G}_{val} . $F(\alpha)$ returns the embeddings trained on \mathcal{G}_{tra} with f , of which the architecture is α . $\mathcal{M}(F(\alpha), \mathcal{G}_{val})$ measure the performance of embeddings on \mathcal{G}_{val} . The problem is to find an architecture α for the path distiller such that validation performance is maximized, i.e.,

$$\alpha^* = \arg \max_{\alpha \in \mathcal{A}} \mathcal{M}(F(\alpha), \mathcal{G}_{val}), \quad (2)$$

where \mathcal{A} is the search space of α (i.e., containing all possible architectures of f).

The training data \mathcal{G}_{tra} is a set of paths. The validation and testing data \mathcal{G}_{val} , \mathcal{G}_{tst} are task dependent. In next part, we will introduce how to design proper search space, which should both consider human wisdom and search efficiently.

3.2 Search Space

The human designed KG embedding models in Tab.1 have various forms. To deal with the structural information, the network f should be able to model non-recurrent architectures covering TransE, ComplEx, architectures without entity embedding like ChainR and RSN, etc. To ensure different ways of processing semantic information, we need to determine how to use different combinator like adding, multiplying, and Hermitian product to make different transformations in the embedding space. Therefore, choosing an appropriate space \mathcal{A} for the distillers is not trivial. Based on the various models in Tab.1, we are motivated to design the search space in Fig. 2.

- **Connections:** In the left part, we focus on structural information by controlling connections. First, a single operator OP1 is used to combine entity embedding \mathbf{s}_t with the distilled recurrent information \mathbf{h}_{t-1} . As discussed in Sec.3.1, we should deal with different connection types to distill the structural information. Specifically, considering non-recurrent structures like TransE, hidden state \mathbf{h}_{t-1} may not be involved in the architecture; OP1 can be unconnected to leave entity embedding \mathbf{s}_t alone like PTransE. Therefore, we select one out of four vectors, i.e. $\mathbf{0}$, \mathbf{h}_{t-1} , output of OP1 and \mathbf{s}_t , to combine with relation embedding \mathbf{r}_t in OP2, and to get the output \mathbf{v}_t in OP3.
- **Operations (Table 2):** Once the connections are fixed, the operators, i.e. OP1, OP2 and OP3, should choose

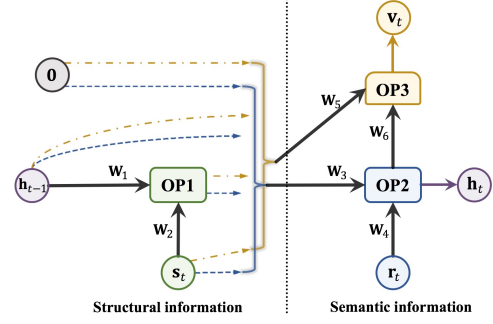


Figure 2: General search architecture of the recurrent network f . OP1-OP3 are three combinators. OP1-OP2 are followed by an activation function. Blue and yellow dashed lines are the one-out-of-four selectors. Left hand side in this figure focuses more on structural information, while the right side is more about semantic.

among different combination functions: adding, multiplying or Hermitian product. To further enhance the distiller’s learning ability, we add the gated operator GRU (Chung et al. 2015) as an additional combinator for OP1 and OP2. Followed by the combination functions, OP1 and OP2 choose activation functions from *identity*, *tanh*, *sigmoid*, *relu* to enable non-linear transformations.

Table 2: Candidates of operations and activation functions for OP1-3.

	operations	activation functions
OP1	adding, multiplying,	identity, tanh,
OP2	Hermitian product, GRU	sigmoid, relu
OP3	adding, multiplying,	identity
	Hermitian product	

Each edge is either a trainable square matrix \mathbf{W} or an identity matrix \mathbf{I} . To summarize, there are three or four candidate operators for OP1-OP3, four activation functions for OP1-OP2, and four different connections for OP2-OP3, resulting in about $4^3 \times 4^2 \times 4^2 \approx 1.6 \times 10^4$ possible models. More details of the search space are given in Appendix B.

3.3 Comparison with NAS for RNN

When forming the recurrent network f , a basic consideration here is to search an RNN cell using standard NAS methods (Zoph and Le 2017). Even though both the distiller here and RNN cells target at learning to control information flow between recurrent steps, there are quite a few differences between them (Tab. 3). First, the search space of RNN focuses more on different operators with fixed connections. However, the search space in Fig. 2 focuses on combining structural and semantic information in KGs. The search space in our work can be explicitly categorized to fit different connections, such as non-recurrent architecture, architectures without entity embedding, fully recurrent architectures, etc. The domain-specific considerations make our space better regularized and generalize many human-designed models. Besides, directly searching RNN cells has not shown ad-

vantage over human-designed models (Zoph and Le 2017; Liu, Simonyan, and Yang 2019) since it lacks domain understanding for specific tasks. However, S2E consistently finds better architectures for various KG tasks.

Table 3: Comparison of state-of-the-arts NAS methods for general RNN with the proposed S2E.

perspective	NAS for RNN	S2E
space	general	KG specific
algorithm	one-shot NAS	natural gradient
performance	worse than human designed models	consistently better

4 Search Algorithm

In previous section, we have introduced the search space \mathcal{A} , which contains tens of thousands of different models. Therefore, how to efficiently search in \mathcal{A} is an important problem. Designing appropriate optimization algorithm for the discrete architecture parameters is a big challenge.

4.1 Failure of One-Shot Architecture Search

One-shot architecture search has been widely used for NAS (Liu, Simonyan, and Yang 2019; Xie et al. 2018; Akimoto et al. 2019) by jointly optimizing the architecture parameters in a supernet, i.e., a network containing all possible architectures spanned by selected operations. Even though the problem of searching the recurrent function can be regarded as a special kind of NAS, i.e., RNN search problem, those are not applicable for the following two reasons:

- P1). validation performance measurement is not differentiable w.r.t architecture parameters α ;
- P2). sharing embedding parameters leads to problematic gradients.

For the first point, training and evaluation are quite different in KG embedding models. In the training procedure, the objective is to maximize score on positive triplets and minimize for the negative ones, while during validation, the ranking metric (Bordes et al. 2013), which is discrete and not differentiable, is generally used to measure the performance. Thus, algorithms that rely on differentiable relaxations to operations, like DARTS (Liu, Simonyan, and Yang 2019), SNAS (Xie et al. 2018), are not applicable in this case. For the second point, embedding parameter sharing is problematic. Different from NAS applications, the training samples are the learned embedding, and architecture performance highly depends on the status of embeddings (see Section 5.5).

4.2 Searching Architectures by Natural Gradient

As one-shot architecture search methods cannot be applied, we may turn to derivative-free optimization methods (Conn, Scheinberg, and Vicente 2009), such as reinforcement learning (Zoph and Le 2017; Baker et al. 2017) and Bayes optimization (Bergstra et al. 2011). However, they are generally

too slow (Conn, Scheinberg, and Vicente 2009). In this sequel, we propose a search algorithm based on natural gradient (NG) (Amari 1998; Pascanu and Bengio 2013), which guides more efficient search by generating pseudo gradient.

Let $p_\theta(\alpha)$ be the distribution of architectures $\alpha \in \mathcal{A}$. To enable the usage of NG, we first use stochastic relaxation (Xie et al. 2018; Akimoto et al. 2019) to transform (2) into a maximization problem over $p_\theta(\alpha)$, i.e.,

$$\begin{aligned} \max_{\theta} J(\theta) &\equiv \max_{\theta} \int_{\alpha \in \mathcal{A}} \mathcal{M}(F(\alpha), \mathcal{G}_{val}) p_\theta(\alpha) d\alpha \\ &= \max_{\theta} \mathbb{E}_{p_\theta} [\mathcal{M}(F(\alpha), \mathcal{G}_{val})], \end{aligned} \quad (3)$$

where the performance is averaged over the distribution p_θ of α . Then, the NG updates θ by $\theta^{t+1} = \theta^t + \rho \mathbf{H}^{-1}(\theta^t) \nabla_{\theta} J(\theta^t)$ where ρ is the step-size, $\mathbf{H}(\theta^t)$ is Fisher matrix at θ^t , and

$$\begin{aligned} \nabla_{\theta} J(\theta^t) &= \mathbb{E}_{p_\theta} [\mathcal{M}(F(\alpha), \mathcal{G}_{val}) \nabla_{\theta} \ln(p_\theta(\alpha))], \\ &= \sum_{i=1}^{\lambda} \mathcal{M}(F(\alpha_i), \mathcal{G}_{val}) \nabla_{\theta} \ln(p_\theta(\alpha_i)), \end{aligned} \quad (4)$$

where each architecture α_i is sampled i.i.d. from p_θ , and λ is the number of samples to approximate the $\mathbb{E}_{p_\theta}[\cdot]$. The key observation in NG to avoid problems P1 is that we do not need to take gradient w.r.t. \mathcal{M} in (4), which is non-differentiable. Instead we only need to get validation performance, which is directly measured by \mathcal{M} . In this way, we can see (4) offers pseudo gradients of α in a parameterized space by θ . Then, for problem P2, the embedded vectors are not shared across different α 's, instead it is directly obtained by model training, i.e., each $F(\alpha_i)$ is different in (4).

Besides, NG itself has several nice properties. As discussed in (Amari 1998; Pascanu and Bengio 2013), NG descent is parameterization-invariant, has good generalization ability, and moreover can be regarded as a second-order method in the space of parameters θ . Details of the NG descent used in our work are given in Appendix A. These facts together with NG's ability of overcoming problem P1 and P2 make NG an ideal choice here. In Sec.5.4, we also show the proposed NG based search method is much efficient than other derivative-free optimization methods.

Remark 4.1. NG has also recently been adopted for searching CNN (Akimoto et al. 2019). However, it is still an one-shot architecture search method, which needs parameter sharing. Thus, it still fails here due to problem P2, which are empirically demonstrated in Sec. 5.5. Besides, it does not consider searching RNN.

5 Experiments

5.1 Experiment setup

Following (Grover and Leskovec 2016; Guo, Sun, and Hu 2019), we sample the relational paths from biased random walks (details in Appendix C.2). The paths make up the training set and will not be resampled for each datasets when evaluating different models.

We use two basic tasks in KG, i.e. entity alignment and link prediction. Same as the literature (Bordes et al. 2013;

Table 4: Performance comparison on entity alignment tasks. $H@k$ is short for $\text{Hit}@k$. The results of TransD (Ji et al. 2015), BootEA (Sun et al. 2018), IPTransE (Zhu et al. 2017) and RSN (Guo et al. 2019) are copied from (Guo et al. 2019).

models		DBP-WD			DBP-YG			EN-FR			EN-DE		
		H@1	H@10	MRR	H@1	H@10	MRR	H@1	H@10	MRR	H@1	H@10	MRR
semantic	TransE	28.4	51.4	0.36	27.0	57.4	0.37	16.2	39.0	0.24	40.3	60.9	0.47
	TransD*	27.7	57.2	0.37	17.3	41.6	0.26	21.1	47.9	0.30	24.4	50.0	0.33
	PTransE	16.7	40.2	0.25	7.4	14.7	0.10	7.3	19.7	0.12	27.0	51.8	0.35
structural	BootEA*	32.3	63.1	0.42	31.3	62.5	0.42	31.3	62.9	0.42	44.2	70.1	0.53
	IPTransE*	23.1	51.7	0.33	22.7	50.0	0.32	25.5	55.7	0.36	31.3	59.2	0.41
	ChainR	32.2	60.0	0.42	35.3	64.0	0.45	31.4	60.1	0.41	41.3	68.9	0.51
	RSN*	38.8	65.7	0.49	40.0	67.5	0.50	34.7	63.1	0.44	48.7	72.0	0.57
NAS-RNN		38.3	65.1	0.47	37.4	67.1	0.47	33.4	62.8	0.43	46.0	72.4	0.54
S2E (proposed)		41.1	71.0	0.51	40.6	73.4	0.52	36.1	67.5	0.46	50.2	75.7	0.59

Zhu et al. 2017; Guo, Sun, and Hu 2019), we use the ranking based metrics: mean reciprocal ranking (MRR) and $\text{Hit}@k$ ($k = 1, 10$). MRR is computed by average of the reciprocal ranks $1/|\mathcal{S}| \sum_{i=1}^{|\mathcal{S}|} \frac{1}{\text{rank}_i}$, where \mathcal{S} is the validation or testing set and $\text{rank}_i, i = 1 \dots \mathcal{S}$ is a set of ranking results. $\text{Hit}@k$ is the percentage of appearance in top- k , namely $1/|\mathcal{S}| \sum_{i=1}^{|\mathcal{S}|} \mathbb{I}(\text{rank}_i \leq k)$, where $\mathbb{I}(\cdot)$ is the indicator function. To avoid underestimating the different models, we report the performance in a “filtered” setting, i.e., all the correct pairs or triplets that exist in training, validation or testing set are filtered out (Bordes et al. 2013). All the models are compared with the same embedding dimension.

Experiments are written in Python with PyTorch framework (Paszke et al. 2017) and run on a TITAN Xp GPU. Training details of each task are given in Appendix C.3

5.2 Comparison with Human-designed models

Entity alignment In this task, two KGs $\mathcal{G}_1 = (\mathcal{E}_1, \mathcal{R}_1, \mathcal{S}_1)$ and $\mathcal{G}_2 = (\mathcal{E}_2, \mathcal{R}_2, \mathcal{S}_2)$ are given. The two KGs have a set of entities that are aligned $\mathcal{E}_{\text{align}} = \{(e_1, e_2) | e_1 \in \mathcal{E}_1, e_2 \in \mathcal{E}_2\}$, and $\mathcal{E}_{\text{align}}$ is split into training/validation/testing sets. During training, the paths can walk through the aligned entity pairs in training set \mathcal{E}_{tra} to interact between \mathcal{G}_1 and \mathcal{G}_2 . And we evaluate the cosine similarity of each (e_1, e_2) in validating set \mathcal{E}_{val} or testing set $\mathcal{E}_{\text{test}}$. We use the four cross-lingual and cross-database subset from DBpedia and Wikidata generated by (Guo, Sun, and Hu 2019): DBP-WD, DBP-YG, EN-FR, EN-DE. Details are given in Appendix C.1.

In this task, structural information is more important since aligned entities usually have similar local topologies. The comparison of the testing performance of the models searched by S2E and human-designed ones is given in Tab. 4. We roughly categorize the human designed models into “semantic” and “structural” based on their main focus. We can observe that, models that leverage structural information generally perform better than those that only focus on semantics. BootEA and IPTransE win over TransE and PTransE respectively by iteratively aligning discovered entity pairs to implicitly leverage the structural information. Performance of PTransE is very bad since it emphasizes too much on learning compositional relations, and ignores entities’ topology along paths. ChainR, RSN and NAS-RNN

outperform BootEA and IPTransE by explicitly processing both entities and relations along path. NAS-RNN wins over ChainR by adapting the RNN architectures but is inferior to RSN due to lack of domain-specific constraint. The searched models by S2E are given in Appendix C.4.

Link prediction To further demonstrate the effectiveness of S2E, we do *link prediction* task which is a general testbed on single triplet models like TransE, ComplEx, SimpleE, etc. In this task, an incomplete KG $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S})$ is given. Our target is to predict the missing entities in unknown link. The evaluation is conducted in the following ways. For each triplet (h, r, t) in the validation set \mathcal{S}_{val} or testing set $\mathcal{S}_{\text{test}}$, we compute the score of (h', r, t) for all $h' \in \mathcal{E}$ and get the rank of h , the same for t based on scores of (h, r, t') over all $t' \in \mathcal{E}$. We use two famous benchmark datasets, WN18-RR (Dettmers et al. 2017) and FB15k-237 (Toutanova and Chen 2015) (statistics in Appendix C.1), which are more realistic than their superset WN18 and FB15k (Bordes et al. 2013).

Table 5: Performance comparison on link prediction tasks.

models	WN18-RR			FB15k-237		
	H@1	H@10	MRR	H@1	H@10	MRR
TransE	12.5	44.5	0.18	17.3	37.9	0.24
ComplEx	41.4	49.0	0.44	21.7	47.5	0.30
PTransE	27.2	46.4	0.34	20.3	45.1	0.29
RSN	38.0	44.8	0.40	19.2	41.8	0.27
S2E	42.1	52.4	0.45	21.9	48.1	0.30

Due to the high computation cost in this task, we compare different models with a small dimension size 64. As shown in Fig. 5, PTransE outperforms TransE by modeling compositional relations, but worse than ComplEx due to the adding operation is inferior to Hermitian product when modeling the interaction between entities and relations (Trouillon et al. 2017). RSN is worse than ComplEx since it pays more attention on structures rather than semantics. And our S2E can search models that perform better than the human-designed models by adaptively searching the architectures.

5.3 Case Study

In this part, we use the synthetic data Countries (Bouchard, Singh, and Trouillon 2015), which contains 271 countries

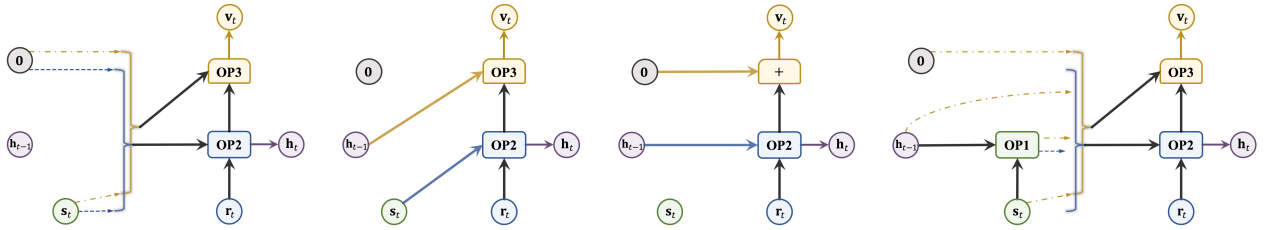


Figure 3: Four patterns with different connection components within the recurrent search space. From left to right: P1 to P4.

and regions, and 2 relations *neighbor* and *locatedin*. This dataset contains three tasks: S1 and S2 require inferring 2 hop relations, i.e. $neighbor \wedge locatedin \rightarrow locatedin$; S3 is harder and requires modeling 3 hop relations $neighbor \wedge locatedin \wedge locatedin \rightarrow locatedin$.

In order to show the difficulty of designing a proper recurrent function for given tasks, we extract four patterns within the search space in Fig. 2 and individually search f in each pattern to fit S1-S3. Specifically, as shown in Fig. 3,

- P1 represents the single hop embedding models;
- P2 processes 2 successive steps;
- P3 models long relational path without entity embedding;
- P4 includes both entities and relations along path.

For each task in S1-S3, we randomly generate 100 models for each pattern and record the model with best *area under curve of precision recall* (AUC-PR) (Boyd, Eng, and Page 2013) on validation set. The procedure is repeated 5 times for each pattern to evaluate the testing performance given in Tab. 6. We can see that no single pattern perform well on all tasks. For easy tasks S1 and S2, semantic information is more important. Incorporating entity embeddings along paths like P4 will hinder the learning ability of directly modeling 2 hop relationships. The two-step pattern P2 consistently performs good in S1 and S2 since it fits best in the two tasks. For the harder task S3, P3 and P4 win over the others since it can model longer steps. P4 slightly outperform P3 by incorporating structural information along path.

Table 6: Performance in Countries datasets.

	S1	S2	S3
P1	0.998±0.001	0.997±0.002	0.918±0.031
P2	1.000±0.000	0.999±0.001	0.923±0.023
P3	0.992±0.001	1.000±0.000	0.944±0.016
P4	0.977±0.028	0.984±0.010	0.949±0.015
S2E	1.000±0.000	1.000±0.000	0.968±0.007

We evaluate the best model searched in each task. As in the last line of Tab. 6, the model searched by S2E achieves good performance on the hard task S3. Besides, S2E prefers different candidates (see Appendix C.4) for S1-S3 over the same search space. This verifies our analysis that distilling structural information and combining it with semantic information is difficult, and there does not exist a unified model that can adapt different KG tasks and datasets well.

5.4 Comparison with NAS Methods

We compare the proposed algorithm and the other search algorithms here. Basically, we use random search (*Rand*) as the simplest baseline. Reinforcement learning (*RL*) has similar objective as (3), but only uses first order gradient. Besides, we use tree parzen estimator (*TPE*) (Bergstra et al. 2011) as a representative Bayes optimization method, which is widely used in searching architectures. Finally, our natural gradient based search algorithm is given as *S2E*.

We show the top1 and top5 mean MRR values w.r.t. the sampled model in Fig. 4 to show the effectiveness of natural gradient. Entity alignment task is evaluated on DBP-WD dataset, and link prediction uses WN18-RR. All the sampled models are trained into convergence and the MRR value of each model is the converged value on validation set. As shown, all heuristic algorithms, i.e. S2E, RL and TPE, outperform random search. Among them, S2E converges faster and performs better than the other searching algorithms as indicated by the blue lines. The reason is that, using REINFORCE gradient (Williams 1992) in RL will lead to sub-optimal solution when θ is not well parameterized. TPE easily falls into local optimum without warm start training (Bergstra et al. 2011). In comparison, the natural gradient $H^{-1}(\theta^t)$ we used is parameterization-invariant and can be regarded as a second-order method, thus performs the best.

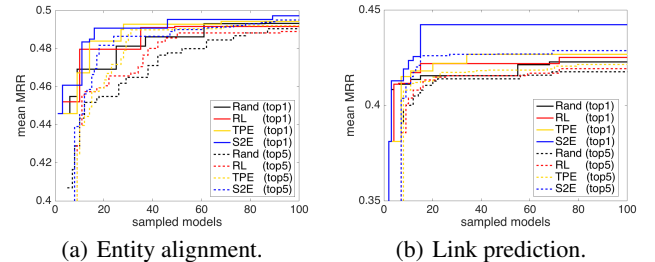


Figure 4: Efficiency comparison of S2E with various NAS methods. Each model is evaluated with the same number of epochs. In entity alignment task, DBP-WD is used. In link prediction, WN18-RR is used.

5.5 Problem on Embedding Sharing

In this part, we empirically show the inherent problems by training a supernet with shared embedding and model parameters. When searching CNN architectures under the one-shot framework, parameters can be shared in the supernet such that the model can focus on the operations that are most

useful for generating good predictions (Bender et al. 2018). However, the input training set is fixed for all the operations in CNN, while the KG embeddings keep changing. As in Fig. 5(a), the distributions of entity embeddings trained by two different KG embedding models are rather distinct. Simultaneously updating the embeddings with different operations will lead to distorted performance as shown in Fig. 5(b). Besides, it is not reliable to compare different operators when the embedding distribution is different. Combining the discussion in Sec. 4.1 and the empirical analysis in this part, we conclude that one-shot architecture search is not adoptable in our searching problem.

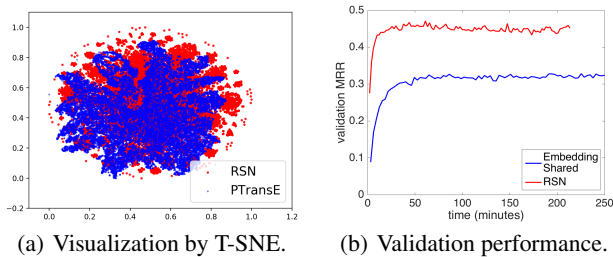


Figure 5: Left: T-SNE (van der Maaten and Hinton 2008) visualization of embeddings from RSN and PTransE. Right: validation performance of embedding shared model and training single model RSN.

6 Conclusion

In this paper, we propose an NAS method S2E to search RNN for learning from relational paths, which contains structural and semantic information, in KGs. By designing a specific search space based on human-designed KG embedding models, S2E can adaptively distill structural information and combine with semantic information for different KG tasks. Since the one-shot architecture framework is not applicable in this problem, we propose an NG based search algorithm that is efficient compared with the other baselines.

References

Akimoto, Y.; Shirakawa, S.; Yoshinari, N.; Uchida, K.; Saito, S.; and Nishida, K. 2019. Adaptive stochastic natural gradient method for one-shot neural architecture search. In *ICML*, 171–180.

Amari, S. 1998. Natural gradient works efficiently in learning. *Neural Computation* 10(2):251–276.

Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J.; Cyganiak, R.; and Ives, Z. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*. Springer. 722–735.

Baker, B.; Gupta, O.; Naik, N.; and Raskar, R. 2017. Designing neural network architectures using reinforcement learning. In *ICLR*.

Bender, G.; Kindermans, P.-J.; Zoph, B.; Vasudevan, V.; and Le, Q. 2018. Understanding and simplifying one-shot architecture search. In *ICML*, 549–558.

Bergstra, J. S.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *NeurIPS*, 2546–2554.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating embeddings for modeling multi-relational data. In *NeurIPS*, 2787–2795.

Bouchard, G.; Singh, S.; and Trouillon, T. 2015. On approximate reasoning capabilities of low-rank vector spaces. In *AAAI Spring Symposium Series*.

Boyd, K.; Eng, K. H.; and Page, C. 2013. Area under the precision-recall curve: point estimates and confidence intervals. In *Joint ECML-KDD*, 451–466. Springer.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2015. Gated feedback recurrent neural networks. In *ICML*, 2067–2075.

Conn, A. R.; Scheinberg, K.; and Vicente, L. N. 2009. *Introduction to derivative-free optimization*, volume 8. Siam.

Dettmers, T.; Minervini, P.; Stenatorp, P.; and Riedel, S. 2017. Convolutional 2D knowledge graph embeddings. In *AAAI*.

Dong, X.; Gabrilovich, E.; Heitz, G.; Horn, W.; Lao, N.; Murphy, K.; Strohmman, T.; Sun, S.; and Zhang, W. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *SIGKDD*, 601–610.

Elsken, T.; Metzen, J. H.; and Hutter, F. 2019. Neural architecture search: A survey. *JMLR* 20(55):1–21.

Grover, A., and Leskovec, J. 2016. Node2vec: Scalable feature learning for networks. In *SigKDD*, 855–864. ACM.

Guo, L.; Sun, Z.; and Hu, W. 2019. Learning to exploit long-term relational dependencies in knowledge graphs. In *ICML*.

Guthrie, D.; Allison, B.; Liu, W.; Guthrie, L.; and Wilks, Y. 2006. A closer look at skip-gram modelling. In *LREC*, 1222–1225.

Hutter, F.; Kotthoff, L.; and Vanschoren, J., eds. 2018. *Automated Machine Learning: Methods, Systems, Challenges*. Springer. In press, available at <http://automl.org/book>.

Ji, G.; He, S.; Xu, L.; Liu, K.; and Zhao, J. 2015. Knowledge graph embedding via dynamic mapping matrix. In *ACL*, volume 1, 687–696.

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

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv:1609.02907*.

Lacroix, T.; Usunier, N.; and Obozinski, G. 2018. Canonical tensor decomposition for knowledge base completion. In *ICML*.

Lin, Y.; Liu, Z.; Luan, H.; Sun, M.; Rao, S.; and Liu, S. 2015. Modeling relation paths for representation learning of knowledge bases. Technical report, arXiv:1506.00379.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable architecture search. In *ICLR*.

Lukovnikov, D.; Fischer, A.; Lehmann, J.; and Auer, S. 2017. Neural network-based question answering over knowledge graphs on word and character level. In *WWW*, 1211–1220.

Nickel, M.; Murphy, K.; Tresp, V.; and Gabrilovich, E. 2016. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE* 104(1):11–33.

Pascanu, R., and Bengio, Y. 2013. Revisiting natural gradient for deep networks. Technical report, arXiv preprint arXiv:1301.3584.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch. In *ICLR*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*, 701–710. ACM.

Socher, R.; Chen, D.; Manning, C.; and Ng, A. 2013. Reasoning with neural tensor networks for knowledge base completion. In *NeurIPS*, 926–934.

Sun, Z.; Hu, W.; Zhang, Q.; and Qu, Y. 2018. Bootstrapping entity alignment with knowledge graph embedding. In *IJCAI*, 4396–4402.

Sundermeyer, M.; Schlüter, R.; and Ney, H. 2012. Lstm neural networks for language modeling. In *CISCA*.

Toutanova, K., and Chen, D. 2015. Observed versus latent features for knowledge base and text inference. In *Workshop on CVSMC*, 57–66.

Trouillon, T.; Dance, C.; Gaussier, E.; Welbl, J.; Riedel, S.; and Bouchard, G. 2017. Knowledge graph completion via complex tensor factorization. *JMLR* 18(1):4735–4772.

van der Maaten, L., and Hinton, G. E. 2008. Visualizing data using t-SNE. *JMLR*.

Wang, Q.; Mao, Z.; Wang, B.; and Guo, L. 2017. Knowledge graph embedding: A survey of approaches and applications. *TKDE* 29(12):2724–2743.

Wang, Z.; Lv, Q.; Lan, X.; and Zhang, Y. 2018. Cross-lingual knowledge graph alignment via graph convolutional networks. In *EMNLP*, 349–357.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *MLJ* 8(3-4):229–256.

Xie, S.; Zheng, H.; Liu, C.; and Lin, L. 2018. SNAS: stochastic neural architecture search. In *ICLR*.

Zhang, F.; Yuan, N. J.; Lian, D.; Xie, X.; and Ma, W.-Y. 2016. Collaborative knowledge base embedding for recommender systems. In *SIGKDD*, 353–362.

Zhu, H.; Xie, R.; Liu, Z.; and Sun, M. 2017. Iterative entity alignment via joint knowledge embeddings. In *IJCAI*, 4258–4264.

Zoph, B., and Le, Q. 2017. Neural architecture search with reinforcement learning. In *ICLR*.

A NG details

To get the optimization direction of architectures α , we use natural gradient to generate pseudo gradient. The key point here is to use stochastic relaxation to give differentiable feedback. Specifically, we introduce a parametric family $\mathcal{P} = \{P_\theta : \theta \in \Theta\}$ of probability distributions defined on the architecture space \mathcal{A} . Assume that for all θ , the distribution P_θ admits the density function p_θ w.r.t. the reference measure $d\alpha$ on \mathcal{A} . The density p_θ approaches the Dirac-Delta distribution δ_α and is differentiable w.r.t. $\theta \in \Theta$. Then we define the stochastic relaxation of $\mathcal{M}(F(\alpha), \mathcal{G}_{val})$ given \mathcal{P} as follows:

$$\begin{aligned} J(\theta) &:= \int_{\alpha \in \mathcal{A}} \mathcal{M}(F(\alpha), \mathcal{G}_{val}) p_\theta(\alpha) d\alpha \\ &= \mathbb{E}_{p_\theta} [\mathcal{M}(F(\alpha), \mathcal{G}_{val})], \end{aligned}$$

leading to the maximization problem in (3).

Since the limit of p_θ converges to a Dirac-Delta distribution δ_{α^*} , maximization of J leads to the maximization of f depending on the condition $\sup_\theta J(\theta) = \sup_{\alpha \in \mathcal{C}} \mathcal{M}(\alpha, \mathcal{G}_{val}) = \mathcal{M}(\alpha^*, \mathcal{G}_{val})$. The gradient is computed as

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{p_\theta(\alpha)} [\mathcal{M}(F(\alpha), \mathcal{G}_{val})] \\ &= \int \nabla_\theta [\mathcal{M}(F(\alpha), \mathcal{G}_{val}) p_\theta(\alpha)] d\alpha \\ &= \int [\mathcal{M}(F(\alpha), \mathcal{G}_{val}) \nabla_\theta p_\theta(\alpha)] d\alpha \\ &= \int [\mathcal{M}(F(\alpha), \mathcal{G}_{val}) \nabla_\theta \log p_\theta(\alpha) \cdot p_\theta(\alpha)] d\alpha \\ &= \mathbb{E}_{p_\theta(\alpha)} [\mathcal{M}(F(\alpha), \mathcal{G}_{val}) \nabla_\theta \ln(p_\theta(\alpha))], \end{aligned}$$

where a well-known log-trick is used in this deduction.

Then the NG algorithm updates θ by

$$\theta^{t+1} = \theta^t + \rho \mathbf{H}^{-1}(\theta^t) \nabla_\theta J(\theta^t), \quad (5)$$

where ρ is the step-size, and $\mathbf{H}^{-1}(\theta^t)$ is Fisher information matrix at θ^t . Usually, computing $\mathbf{H}^{-1}(\theta^t)$ is time consuming. To reduce computation cost, the exponential family $h(\alpha \cdot \exp(\eta(\theta)^\top T(\alpha) - A(\theta)))$, where $h(\alpha)$, $T(\alpha)$ and $A(\theta)$ are known functions depending on the target distribution, is used to parameterize the distribution \mathcal{P} . For simplicity, we set $h(\alpha) = 1$ and choose the expectation parameters $\theta = \mathbb{E}_{p_\theta}[T(\alpha)]$. Then the gradient reduces to $\nabla_\theta \ln(p_\theta(\alpha)) = T(\alpha) - \theta$, and inverse Fisher information matrix is $\mathbf{H}^{-1}(\theta) = \mathbb{E}[(T(\alpha) - \theta)(T(\alpha) - \theta)^\top]$. In this work, since the architecture parameters are all categorical, for a component with K categories, we use a K dimensional vector θ to model the distribution p_θ . The probability of sampling the i -category is θ_i and $\theta_K = 1 - \sum_{j=1}^{K-1} \theta_j$. Note that, the natural gradient is only used to update the first $K-1$ dimension of θ , and $T(\alpha)$ is a one-hot vector (without dimension K) of the category α .

B Search Space Details

B.1 Hermitian product

The Hermitian product is defined on complex space same as in (Trouillon et al. 2017). Denote two complex vectors

$\mathbf{x} = \mathbf{x}_{re} + i\mathbf{x}_{im}$, where \mathbf{x}_{re} and \mathbf{x}_{im} are real and image part respectively, and $\mathbf{y} = \mathbf{y}_{re} + i\mathbf{y}_{im}$, the Hermitian product is

$$\begin{aligned} \mathbf{x} \otimes \mathbf{y} &= (\mathbf{x}_{re} + i\mathbf{x}_{im}) \otimes (\mathbf{y}_{re} + i\mathbf{y}_{im}) \\ &= (\mathbf{x}_{re} \cdot \mathbf{y}_{re} - \mathbf{x}_{im} \cdot \mathbf{y}_{im}) + i(\mathbf{x}_{re} \cdot \mathbf{y}_{im} + \mathbf{x}_{im} \cdot \mathbf{y}_{re}). \end{aligned}$$

Based on above equation, we can give the Hermitian product in the real space. Given two real-valued vectors \mathbf{x}, \mathbf{y} which have the same dimension size d and $d\%2 = 0$, we can evenly split both vectors into two parts, i.e. $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2]$ and $\mathbf{y} = [\mathbf{y}_1, \mathbf{y}_2]$, where $\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}_1, \mathbf{y}_2$ have the same dimension $d/2$. Then

$$\begin{aligned} \mathbf{v} &= \mathbf{x} \otimes \mathbf{y} \\ &= [\mathbf{x}_1 \cdot \mathbf{y}_1 - \mathbf{x}_2 \cdot \mathbf{y}_2, \quad \mathbf{x}_1 \cdot \mathbf{y}_2 + \mathbf{x}_2 \cdot \mathbf{y}_1]. \end{aligned} \quad (6)$$

Thus, we get a d -dimensional composition vector \mathbf{v} .

Proposition 1. *Given a triplet (s, r, o) and the embeddings $\mathbf{s}, \mathbf{r}, \mathbf{o}$, let $\mathbf{v} = \mathbf{s} \otimes \mathbf{r}$, then $\mathbf{v}^\top \mathbf{o}$ is the same as Complex (Trouillon et al. 2017).*

Proof. Based on (6),

$$\begin{aligned} \mathbf{v}^\top \mathbf{o} &= \langle \mathbf{v}, \mathbf{o} \rangle \\ &= \langle [\mathbf{s}_1 \cdot \mathbf{r}_1 - \mathbf{s}_2 \cdot \mathbf{r}_2, \quad \mathbf{s}_1 \cdot \mathbf{r}_2 + \mathbf{s}_2 \cdot \mathbf{r}_1], [\mathbf{o}_1, \mathbf{o}_2] \rangle \\ &= \sum (\mathbf{s}_1 \cdot \mathbf{r}_1 \cdot \mathbf{o}_1 - \mathbf{s}_2 \cdot \mathbf{r}_2 \cdot \mathbf{o}_1 + \mathbf{s}_1 \cdot \mathbf{r}_2 \cdot \mathbf{o}_2 + \mathbf{s}_2 \cdot \mathbf{r}_1 \cdot \mathbf{o}_2), \end{aligned} \quad (7)$$

where the summation works along the embedding dimension. By replacing “1” with the real part of complex value and “2” with image part, (7) resembles the equation (7) in (Trouillon et al. 2017). \square

B.2 Space size reduction

As discussed, there are 1.6×10^4 possible architectures. However, not all of them need to be trained and evaluated. We can filter out these combinations that are not promising.

Proposition 2. *When zero vector $\mathbf{0}$ is connected to OP2 and combinator in OP2 is multiply or Hermitian product, then the output of OP2 will be $\mathbf{0}$.*

It is easy to see that multiplying \mathbf{r}_t with $\mathbf{0}$ will lead to zero output. Based on the introduction of Hermitian product in B.1, it will also output $\mathbf{0}$ if $\mathbf{0}$ and \mathbf{r}_t are combined with Hermitian product. In this case, \mathbf{r}_t will be removed and there will be no information propagated recurrently. This case is the same for OP3 since multiplying with $\mathbf{0}$ will get $\mathbf{v}_t = \mathbf{0}$.

Besides, the GRU (Chung et al. 2015) unit already contains weighted edge as well as activation functions. As a function specially designed for recurrent structures, OP2 is only allowed to choose GRU when OP1 or \mathbf{h}_{t-1} is connected. When \mathbf{s}_t or $\mathbf{0}$ is connected to OP2, a recurrent architecture will not exist.

Based on above analysis, we manually avoid evaluating the architectures that are not promising as in Prop. 2 or not making sense as the non-recurrent connection of GRU.

C Experiment details

C.1 Datasets

The datasets we used for entity alignment task are subset of DBpedia and Wikidata. They are cross-lingual or cross-KG in DBpedia and Wikidata. Detailed statistics and informations are given in Tab. 7. They are the same as the *Normal* version, which is very sparse, in (Guo, Sun, and Hu 2019).

The data for link prediction is WN18-RR and FB15k-237, which are subset of WN18 and FB15k respectively. Compared with their superset, the two selected sets are more realistic.

Table 7: Statistics of the datasets we used for entity alignment. Each single KG contains 15,000 entities. There are 4,500 aligned entity pairs in training sets, and 11,500 pairs for evaluation. The first 10% pairs among the 11,500 are used for validation and the left for testing. We use “#” short for “number of”.

	data source	# relations	# triplets
DBP-WD	DBpedia-English	253	38,421
	Wikidata-English	144	40,159
DBP-YG	DBpedia-English	219	33,571
	YAGO3-English	30	34,660
EN-FR	DBpedia-English	211	36,508
	DBpedia-French	177	33,532
EN-DE	DBpedia-English	225	38,281
	DBpedia-German	118	37,069

Table 8: Statistics of the datasets we used for link prediction. “rels” is short for “relations”.

Dataset	#entity	#rels	#train	#valid	#test
WN18-RR	40,943	11	86,835	3,034	3,134
FB15k-237	14,541	237	272,115	17,535	20,466

C.2 Path sampling

Following (Grover and Leskovec 2016; Guo, Sun, and Hu 2019), we sample path from biased random walks. In conventional random walk, all the neighbors have the same probability to be sampled as the next step. In biased random walk, we sample the neighbor that can go deeper or go to another KG with larger probability. For single KG, we use one parameter $\alpha \in (0.5, 1)$ to give the depth bias. Specifically, the next step has probability of α to go to the neighbors that are two step away from the previous one, and probability of $1 - \alpha$ to go to other neighbors. Similarly, we have another parameter $\beta \in (0.5, 1)$ for two KGs to give cross-KG bias. The next step is encouraged to jump to another KG through the entity alignment pairs in \mathcal{E}_{tra} . Since the aligned pairs are usually rare in the training set, encouraging cross-KG walk can learn better about the aligned entities in the two KGs.

Since the KGs are usually large, we sample two path for each triplet in \mathcal{S}_{tra} . The length of paths is 7 for entity alignment task on two KGs and 3 for link prediction task on single KG. The paths are fixed for each dataset to keep a fair

comparison among different models. The parameters used for sampling path are summarized in Tab. 9.

Table 9: Parameter used for sampling path.

parameters	α	β	length
entity alignment	0.9	0.9	7
link prediction	0.7	–	3

Once the paths are sampled, we start training based on paths. The loss for each path is given in (8)

$$\mathcal{M}_{tra} = \sum_{t=1}^L \left\{ -\mathbf{v}_t \cdot \mathbf{o}_t + \log \left(\sum_{o_i \in \mathcal{E}} \exp(\mathbf{v}_t \cdot \mathbf{o}_i) \right) \right\} \quad (8)$$

In each recurrent step t , we focus on one triplet (s_t, r_t, o_t) . The subject embedding \mathbf{s}_t and relation embedding \mathbf{r}_t process along with the distilled information \mathbf{h}_{t-1} to get the output \mathbf{v}_t . The output is encouraged to approach the object embedding \mathbf{o}_t . Thus, in (8), the objective is a multi-class log-loss while the object \mathbf{o}_t is the true label. Rather than use the skip-gram model (Guthrie et al. 2006) based on negative samples, the multi-class loss introduced by (Lacroix, Usunier, and Obozinski 2018) is more stable. Training is based on mini-batch gradient descent. Adam (Kingma and Ba 2014) is used as the optimization method for updating model parameters.

C.3 Hyperparameters and training details

We use RSN (Guo, Sun, and Hu 2019) as a standard baseline to tune parameters. We use the *HyperOpt* package, which is based on tree parsen estimator, to search the learning rate η , L2 penalty λ , decay rate u , batch size m , as well as a dropout rate p , which applies on input embeddings. The tuning ranges are given in Tab. 10.

Table 10: Searching range of hyper-parameters

hyper-param	ranges
η	$[10^{-5}, 10^{-3}]$
λ	$[10^{-5}, 10^{-2}]$
u	$[0.98, 1]$
m	$\{128, 256, 512, 1024, 2048\}$
p	$[0, 0.6]$

The embedding dimension for entity alignment task is 256, and for link prediction is 64. After the hyper-parameter tuning, the proposed S2E starts searching with this hyper-parameter setting. For all the tasks and datasets, we search and evaluate 100 models. Among them, we select the best model indicated by the MRR performance on validation set and fine-tune hyper-parameters for this model.

As for the searching time, about half GPU day for tuning hyper-parameters, two GPU days for searching the model in entity alignment datasets. Link prediction takes longer time since the evaluation process is more expensive. It takes one GPU day for hyper-parameters, two GPU days for WN18-RR and three GPU days for FB15k-237. Note that, the cost for searching a better model is in the same order of magnitude as tuning hyper-parameters. This means our search space is reasonable and the search algorithm is efficient.

C.4 Searched models

Entity alignment The searched models by S2E, which consistently perform best on each dataset, are graphically illustrated in Fig. 6. As shown, all of the searched recurrent networks processing recurrent information in h_{t-1} , entity embedding s_t and relation embedding r_t together. But they have different connections, different composition operator and different activation functions. The functions f learned on DBP-WD and EN-FR, joint two successive triplets to learn a small neighborhood, while the other two process longer steps.

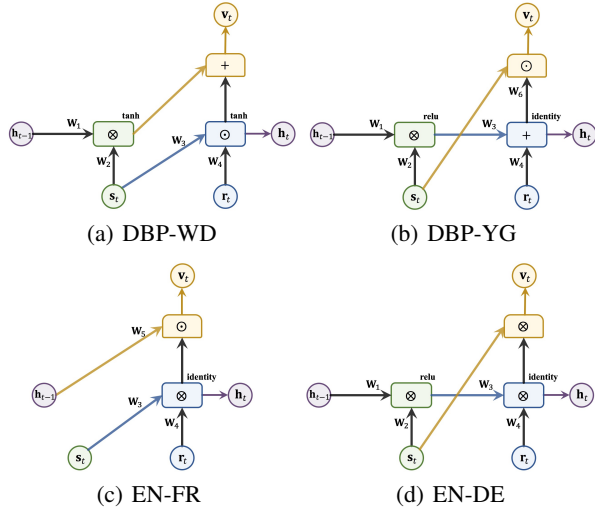


Figure 6: Graphical representation of the searched recurrent network f on each datasets in entity alignment task.

Link prediction The best models searched in link prediction tasks are given in Fig 7. And we make a statistics of the distance, i.e. the shortest path distance when regarding the KG as a simple undirected graph, between two entities in the validation and testing set in Tab. 11. As shown, most of the triplets have distance less than 3. Besides, as indicated by the performance on the two datasets, we infer that triplets far away from 3-hop are very challenging to model. At least in this task, triplets that are less or equal than 3 hops are the main focus of different models. This also explains why RSN, which processes long paths, does not perform well in the link prediction task. The searched models in Fig. 7 do not consider long term structures. The architecture on WN18-RR models two successive triplets, and the architecture on FB15k-237 only models single triplet. They focus more on the semantics and use gradient descent to interact among triplets.

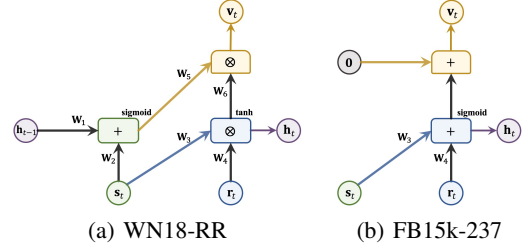


Figure 7: Graphical representation of the searched f on each datasets in link prediction task.

Table 11: Percentage of the n -hop triplets in validation and testing datasets.

Datasets		Hops			
		≤ 1	2	3	≥ 4
WN18-RR	validation	35.5%	8.8%	22.2%	33.5%
	testing	35.0%	9.3%	21.4%	34.3%
FB15k-237	validation	0%	73.2%	26.1%	0.7%
	testing	0%	73.4%	26.8%	0.8%

Countries We also give the graphical illustration of architectures searched in Countries dataset in Fig. 8. The search procedure is conducted in the whole search space rather than the four patterns P1-P4. We can see that in Fig. 8, (a) belongs to P1, (b) belongs to P2 and (d) belongs to P4. There results further verify that S2E can adaptively search architectures for specific KG tasks and datasets.

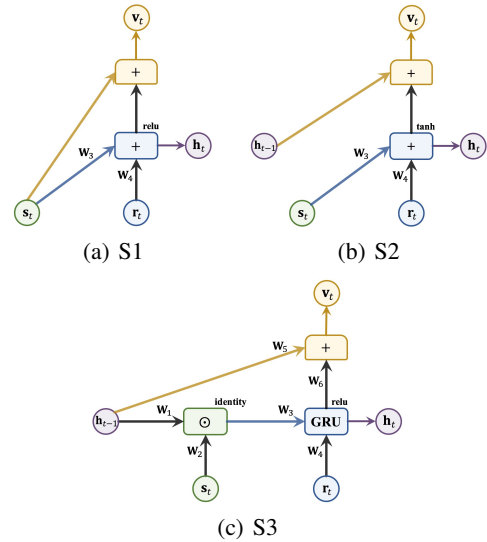


Figure 8: Graphical representation of the searched f on countries dataset.