2019 IEEE 35th International Conference on Data Engineering (ICDE)

NSCaching: Simple and Efficient Negative Sampling for Knowledge Graph Embedding

Yongqi Zhang[†], Quanming Yao[‡], Yingxia Shao[§] and Lei Chen[†]
[†]The Hong Kong University of Science and Technology, Hong Kong SAR, China
[‡]4Paradigm Inc., Beijing, China
[§]Beijing Key Lab of Intelligent Telecommunications Software and Multimedia, BUPT, Beijing, China
[†]{yzhangee,leichen}@cse.ust.hk, [‡]yaoquanming@4paradigm.com, [§]shaoyx@bupt.edu.cn

***Abstract*—Knowledge graph (KG) embedding is a fundamental problem in data mining research with many real-world applications. It aims to encode the entities and relations in the graph into low dimensional vector space, which can be used for subsequent algorithms. Negative sampling, which samples negative triplets from non-observed ones in the training data, is an important step in KG embedding. Recently, generative adversarial network (GAN), has been introduced in negative sampling. By sampling negative triplets with large scores, these methods avoid the problem of vanishing gradient and thus obtain better performance. However, using GAN makes the original model more complex and harder to train, where reinforcement learning must be used. In this paper, motivated by the observation that negative triplets with large scores are important but rare, we propose to directly keep track of them with cache. However, how to sample from and update the cache are two important questions. We carefully design the solutions, which are not only efficient but also achieve good balance between exploration and exploitation. In this way, our method acts as a "distilled" version of previous GAN-based methods, which does not waste training time on additional parameters to fit the full distribution of negative triplets. The extensive experiments show that our method can gain significant improvement on various KG embedding models, and outperform the state-of-the-arts negative sampling methods based on GAN.**

I. Introduction

Knowledge graph (KG) is a special kind of graph structure, with entities as nodes, and relations as directed edges. Each edge (also called a fact) is represented as a triplet with the form *(head entity, relation, tail entity)*, which is denoted as (h, r, t), indicating that two entities are connected by a specific relation, e.g. *(Shakespeare, isAuthorOf, Hamlet)* [2], [4], [30]. KG is very general and useful, and it has been used as fundamental building blocks for many applications, e.g., structured search [29] and question answering [5]. Such importance has also inspired many famous KG projects, e.g., FreeBase [4], DBpedia [2], and YAGO [30].

However, as these triplets are hard to manipulate, one fundamental problem is how to find a good representation for entities and relations in the KG [25]. Early works towards this goal lie in statistical relational learning using the symbolic triplet data [11], [18], [20]. However, these methods neither lead to good generalization performance, nor can they be applied for large scale knowledge graphs. Recently, graph embedding techniques [34] have been introduced in KG. These methods attempt to encode entities and relations in KG into a low-dimensional vector space while capturing nodes' and edges' connection properties. They are scalable and have also shown a promising performance in basic KG tasks, such as link prediction and triplet classification [7], [34].

Besides, based on the learned entity and relation embeddings, downstream tasks, such as entity classification [27] and entity linking [6], can also be benefited. Given that the relation encoding entity types (denoted as *IsA*) is contained in the KG and has been included into the learning process, entity classification can be treated as the link prediction task $(x, IsA, ?)$. Similarly, if the relation encoding equivalent entities (denoted as *EqualTo*) is included, entity linking can be treated as triplet classification task to classify $(a, EqualTo, b)$. A more direct entity linking method proposed in [27] is to check the similarity score between embeddings of two entities.

In recent years, constructing new scoring functions which can better model the complex interactions between entities and relations has been the main focus for improving KG embedding's performance [16], [32], [36], [38]. However, another very important perspective of KG embedding, i.e., negative sampling, is not sufficiently emphasized. The need of negative sampling comes from the fact that there are only positive triplets in KG [11], [34]. To avoid trivial solutions of the embedding, for each positive triplet, a set that contains its all possible negative samples, needs to be hand-made. Then, for the effectiveness and efficiency of stochastic updates in the KG embedding, once we have picked up a positive triplet, we also need to sample some negative triplets from its corresponding negative sample set. Unfortunately, the quality of these negative triplets does matter.

Due to its simplicity and efficiency, uniform sampling is broadly used in KG embedding [34]. However, it is a fixed scheme and ignores changes on the distribution of negative triplets during the training. Thus, it suffers seriously from the *vanishing gradient* problem. Specifically, as observed in [33], most negative triplets in the sampling set are easily classified ones. Since scoring functions tend to give observed (positive) triplets large values, as training goes, scores (evaluated from scoring functions) for most non-observed (probably negative) triplets become smaller. Thus, when negative triplets are uniformly sampled, it is very likely that we pick up one

This work is partially done when Y. Zhang was an intern in 4Paradigm Inc., and Q. Yao is the correspondence author.

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00061

with zero gradient. As a result, the training process of KG embedding will be impeded by such vanishing gradients rather than by the optimization algorithm. Such problem prevents KG embedding getting desired performance. A better sampling scheme, i.e., Bernoulli sampling, is introduced in [36]. It improves uniform sampling by considering one-to-many, many-to-many, and many-to-one mapping in relation between head and tail. However, it is still a fixed sampling scheme, which suffers from vanishing gradients.

Therefore, high-quality negative triplets should have large scores. To efficiently capture them during training, we have two main challenges for the negative sampling: (i). How to capture and model negative triplets' dynamic distribution? and (ii). How can we sample negative triplets in an efficient way? Recently, there are two pioneered works, i.e., IGAN [33] and KBGAN [8], attempting to address these challenges. Their ideas are both replacing the fixed sampling scheme with a generative adversarial network (GAN) [13]. However, the GAN-based solutions still have many problems. First, GAN increases the number of training parameters because an extra generator is introduced. Second, GAN training can suffer from instability and degeneracy [1], [15], and the REINFORCE gradient [37] used in IGAN and KBGAN is known to have high variance. These drawbacks lead to instable performance for different scoring functions, and hence pretrain becomes a must for both IGAN and KBGAN.

In this paper, to address the challenges of high-quality negative sampling while avoiding the problems from using GAN, we propose a new negative sampling method based on cache, called NSCaching. With empirically studying the score distribution of negative samples, we find that the score distribution is highly skewed, i.e., there are only a few negative triplets with large scores and the rest are useless. This observation motivates to only maintain high-quality negative triplets during the training, and dynamically update the maintained triplets. First, we store the high-quality negative triplets in cache, and then design importance sampling (IS) strategy to update the cache. The IS strategy can not only capture the dynamic characteristic of the distribution, but also benefit the efficiency of NSCaching. Furthermore, we also take good care of "exploration and exploitation", which balances exploring all possible high-quality negative triplets and sampling from a few large score negative triplets in cache. Contributions of our work are summarized as follows:

- We propose a simple and efficient negative sampling scheme, NSCaching. It is a general negative sampling scheme, which can be injected into all popularly used KG embedding models. NSCaching has fewer parameters than both IGAN and KBGAN, and can be trained with gradient descent as the original KG embedding models.
- We propose the uniform strategy to sample from the cache and IS strategy to update the cache in NSCaching with good care of *"exploration and exploitation"*.
- We analyze the connection between NSCaching and the *self-paced learning* [3], [19]. We show NSCaching can firstly learn easily classified samples, and then gradually switch to harder samples.
- We conduct experiments on four popular datasets, i.e., WN18 and FB15K, and their variants WN18RR and FB15K237. Experimental results demonstrate that our method is very efficient and is more effective than the state-of-the-arts, i.e., IGAN and KBGAN, as well.

Notations. We denote the set of entities as $\mathcal{E}$ and set of relations as $\mathcal{R}$. A fact (edge) in KG is represented by a triplet, i.e., (h, r, t), where $h \in \mathcal{E}$ is the head entity, $t \in \mathcal{E}$ is the tail entity, and $r \in \mathcal{R}$ is the relationship. Observed facts in a KG are represented by a set $\mathcal{S} \equiv \{(h, r, t)\}$. Finally, we denote the embedding vectors of h, r and t by its corresponding boldface character, i.e. $\mathbf{h}$, $\mathbf{r}$ and $\mathbf{t}$.

II. Preliminary: Framework of KG Embedding

In this section, we first introduce the general framework for training KG embedding models in Section II-A. Then, we describe its two key components, i.e., negative sampling and scoring function in Section II-B and II-C respectively.

A. The General Framework

To build a KG embedding model, the most important thing is to design a scoring function f, which captures the interactions between two entities based on a relation [34]. Different scoring functions have their own weaknesses and strengths in capturing the underneath interactions. Besides, the observed facts in KG are supposed to have larger scores than non-observed ones. With the factual information, the embeddings are learned by solving the optimization problem that maximizes the scoring function for observed triplets and minimizes for non-observed triplets at the same time. Based on the properties of scoring functions, KG embedding models are generally divided into two categories. The first one is translational distance model, i.e.,

$$L(\mathcal{E}, \mathcal{R}) = \sum_{(h,r,t)\in\mathcal{S}} \left[\gamma - f(h, r, t) + f(\bar{h}, r, \bar{t})\right]_+, \qquad (1)$$

and the second one is semantic matching model, i.e.,

$$L(\mathcal{E}, \mathcal{R}) = \sum_{(h,r,t)\in\mathcal{S}} \left[\ell\left(+1, f(h, r, t)\right) + \ell\left(-1, f(\bar{h}, r, \bar{t})\right)\right], \quad (2)$$

where $(\bar{h}, r, \bar{t}) \notin \mathcal{S}$ is the hand-made negative triplet for (h, r, t) and $\ell(\alpha, \beta) = \log\left(1 + \exp(-\alpha\beta)\right)$ is the logistic loss.

The above two objectives can be optimized by using stochastic gradient descent in an unified framework (Algorithm 1). In each iteration, a mini-batch $\mathcal{S}_{\text{batch}}$ of size m is firstly sampled from $\mathcal{S}$ at step 3. In step 5, since there are no negative triplets in $\mathcal{S}$, a set $\bar{\mathcal{S}}_{(h,r,t)}$, i.e.,

$$\bar{\mathcal{S}}_{(h,r,t)} = \left\{(\bar{h}, r, t) \notin \mathcal{S} \,|\, \bar{h} \in \mathcal{E}\right\} \cup \left\{(h, r, \bar{t}) \notin \mathcal{S} \,|\, \bar{t} \in \mathcal{E}\right\}, \quad (5)$$

which contains negative triplets for (h, r, t), is made, and one negative triplet $(\bar{h}, r, \bar{t})$ is sampled from $\bar{\mathcal{S}}_{(h,r,t)}$. Finally, embedding parameters are updated in step 6. Thus, in optimization, the most important problem is how to do negative sampling, i.e. generate and sample negative triplet from $\bar{\mathcal{S}}_{(h,r,t)}$.

Algorithm 1 General framework of KG embedding.

Input: training set $\mathcal{S} = \{(h, r, t)\}$, embedding dimension d and scoring function f;

initialize the embeddings for each $e \in \mathcal{E}$ and $r \in \mathcal{R}$.
for $i = 1, \cdots, T$ **do**
  sample a mini-batch $\mathcal{S}_{\text{batch}} \in \mathcal{S}$ of size m;
  for each $(h, r, t) \in \mathcal{S}_{\text{batch}}$ **do**
    sample a negative triplet $(\bar{h}, r, \bar{t}) \in \bar{\mathcal{S}}_{(h,r,t)}$;
    // negative sampling
    update parameters of embeddings w.r.t. the gradients using (i). translational distance models:

$$\nabla \left[\gamma - f\left(h, r, t\right) + f\left(\bar{h}, r, \bar{t}\right)\right]_+, \tag{3}$$

    or (ii). semantic matching models:

$$\nabla \ell\left(+1, f(h, r, t)\right) + \nabla \ell\left(-1, f(\bar{h}, r, \bar{t})\right); \tag{4}$$

  end for
end for

B. Negative Sampling

Existing works on negative sampling can be divided into two categories, i.e., sample from fixed and sample from dynamic distributions.

1) Sample from fixed distributions: In the early works [7], negative triplets are *uniformly* sampled from the set $\bar{\mathcal{S}}_{(h,r,t)}$. Such strategy is simple and efficient. Later, a better sampling scheme, i.e., Bernoulli sampling, is introduced in [36]. It improves uniform sampling by reducing the appearance of false negative triplets existing in one-to-many, many-to-many, and many-to-one relations between head and tail entities. However, as mentioned in the introduction, they still sample from fixed distributions, which can neither model the dynamic changes in distributions of negative triplets nor sample triplets with large scores. Thus, they seriously suffer from vanishing gradient.

2) Sample from dynamic distributions: More recently, two pioneered works [8], [33] made a more dedicated analysis of problems with fixed sampling scheme. They observed that most of the negative triplets are easy ones, of which scores quickly go small during the training. This leads to the vanishing gradient problem if a fixed sampling scheme is used. Motivated by the success of Generative Adversarial Network (GAN) [13] on modeling dynamic distribution, IGAN and KBGAN introduce GAN for negative sampling in KG.

When GAN is applied to negative sampling, a jointly trained generator serves as a sampler that can not only generate high-quality triplets by confusing the discriminator, but also dynamically adapt to the new distributions by keeping training. The discriminator, i.e., the KG embedding model, learns to distinguish between the positive triplets and the negative triplets selected by the generator. Under an alternating training procedure, the generator dynamically approximates the negative sample distribution, and the KG embedding model is improved by high-quality negative samples.

Specifically, given a positive triplet (h, r, t), IGAN models the distribution $\bar{h}, \bar{t} \sim p(e|(h, r, t))$ over all entities to form a negative triplet $(\bar{h}, r, \bar{t})$. The quality of $(\bar{h}, r, \bar{t})$ is measured by the scoring function of the discriminator, i.e. the target KG embedding model. By joint training, IGAN can dynamically sample negative triplets with high quality. KBGAN operates in a different way. Instead of modeling a distribution over the whole entity set, KBGAN learns to sample from a subset of random entities. Namely, it first uniformly samples a set of entities to form a candidate set $\mathcal{N}eg = \{(\bar{h}, r, \bar{t})\}$, and then picks up one triplet from it. Under the framework of GAN, generator in KBGAN can approximate the score distribution of triplets in the set $\mathcal{N}eg$, and sample a triplet with relatively high quality.

Even though GAN provides a solution to model the dynamic negative sample distribution, it is famous for suffering from instability and degeneracy [1], [15]. Besides, REINFORCE gradient [37], which is known to have high variance, has to be used. Thus, pretrain is a must for both IGAN and KBGAN. Finally, it increases the number of model's parameters and brings extra costs on training.

C. Scoring Functions

The design of scoring function has been the main power source for improving embedding performance in recent years. Depending on the property of scoring functions, they are used in either translational distance or semantic matching models.

1) Translational distance model: The simplest and most representative *translational distance model* is TransE [7]. Inspired from the word representation learning area [23], if a triplet (h, r, t) is true, the entity embeddings $\mathbf{h}, \mathbf{t}$ should be connected by the relational vector $\mathbf{r}$, i.e. $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$. Under this assumption for example, two facts (*China, Capital, Beijing*) and (*UK, Capital, London*) will enjoy a relation that $China - UK \approx Beijing - London$ in the embedding space. Thus in TransE, the scoring function is defined as the negative translational distance of $\mathbf{h}$ and $\mathbf{t}$ connected by relation $\mathbf{r}$, i.e., $f(h, r, t) = \|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_1$.

Despite the simplicity of TransE, it faces the problem when dealing with one-to-many, many-to-one and many-to-many relations. Take one-to-many relation for example, TransE enforces $\mathbf{h} + \mathbf{r} \approx \mathbf{t}_i$ for different tail entity t_i, thus resulting in very similar embeddings for these different entities. To solve this problem, variants like TransH [36], TransR [21], TransD [16] are introduced to project embeddings of head/tail entity $\mathbf{h}, \mathbf{t}$ into various spaces. By maximizing the scoring function for all positive triplets, the distance between $\mathbf{h} + \mathbf{r}$ and $\mathbf{t}$ in corresponding space can be reduced.

2) Semantic matching model: Another group of scoring functions operate without the assumption that $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$. Instead, they use similarity to measure the plausibility of triplets (h, r, t). RESCAL [27] is the most original model. The entity embeddings $\mathbf{h}, \mathbf{t}$ are also continuous vectors in $\mathbb{R}^d$. But for each relation, it is represented as a matrix which models the pairwise interaction between every dimension in entity embedding space $\mathbb{R}^d$. Namely, the scoring function of a triplet

(h, r, t) is defined as $f(h, r, t) = \mathbf{h}^\top \mathbf{M}_r \mathbf{t}$, where the relation is represented as a matrix $\mathbf{M}_r \in \mathbb{R}^{d \times d}$. This scoring function captures pairwise interactions between all components of $\mathbf{h}$ and $\mathbf{t}$, which needs $O(d^2)$ parameters per relation.

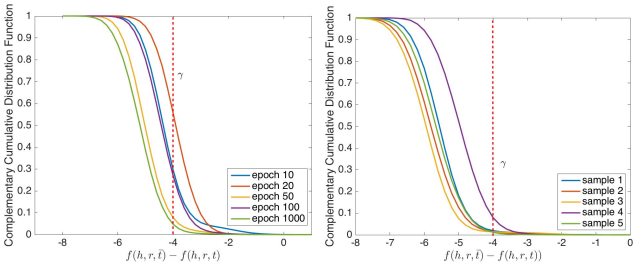
Some simple and effective variants of RESCAL are DistMult [38], HolE [26] and ComplEx [32]. DistMult simplifies RESCAL by restricting the interaction matrix $\mathbf{M}_r$ into a diagonal matrix, which can reduce the number of parameters per relation from $O(d^2)$ to $O(d)$. HolE and ComplEx improves DistMult by modeling asymmetric relations.

III. Proposed Model

In this section, we first describe our key observations in Section III-A, which are ignored by existing works but are the main motivations of our work. The proposed method is described in Section III-B, where we show how challenges in negative sampling are addressed by cache. Finally, we show an interesting connection between NSCaching and self-pace learning [19] in Section III-C, which further explains the effectiveness.

A. Closer Look at Distribution of Negative Triplets

Recall that, in Equation (5), the negative triplet $(\bar{h}, r, \bar{t}) \notin \mathcal{S}$ is formed by replacing either the head or tail entity of a positive triplet $(h, r, t) \in \mathcal{S}$ with any other entities in $\mathcal{E}$. Before introducing the proposed method, we analyze the distribution of scores for $(\bar{h}, r, \bar{t}) \in \bar{\mathcal{S}}_{(h,r,t)}$.



(a) Different epochs. (b) Different triplets.

Fig. 1. Distribution of negative triplets on WN18 trained by Bernoulli-TransD (see Section IV-B1). For a given triplet (h, r, t), we fix the head entity h and relation r, and compute the distance $D_{(h,r,\bar{t})} = f(h, r, \bar{t}) - f(h, r, t)$. We measure the *complementary cumulative distribution function* (CCDF) $F_D(x) = P(D \geq x)$ to show the proportion of negative triplets that satisfy $D \geq x$. The red dashed line shows where the margin γ lies. (a) is the distribution of negative triplets in 6 timestamp of a certain triplet (h, r, t). (b) is the negative sample distribution of 5 different triplets (h, r, t) after the pretraining stage.

Figure 1(a) shows the changes in the distribution of negative samples for one positive triplet; and Figure 1(b) shows distributions of negative samples from different positive triplets. Note that once the distance is larger than the margin γ, i.e., the red vertical line, the gradient of corresponding negative triplets will vanish to zero. Indeed, we can see the distribution changes during the training process; and negative triplets with large scores are rare. These observations are consistent with those in [8], [33], which further explain the vanishing gradient problem of uniform sampling, as most sampled negative triplets will have small scores.

Although, the necessity of finding negative triplets with large scores from a dynamic distribution is mentioned by above works, they do not deeply study these distributions.

Key Observations. *The more important observations are:*

- *The score distribution of negative triplets is highly skewed.*
- *Regardless of the training (Figure 1(a)) and the choice of positive triplets (Figure 1(b)), only a few negative triplets have large scores.*

Thus, while GAN has strong ability to monitor the full generation process of negative triplets, it wastes a lot of parameters and training time on learning how negative triplets with small scores are distributed. This is obviously not necessary. Besides, reinforcement learning has been used once GAN is applied, which increases the difficulties on training. As a result, *is it possible to directly keep track of those negative triplets with large scores*?

B. NSCaching: the Proposed Method

In this section, we describe the proposed method, which addresses the aforementioned question. The basic idea is very simple and intuitive. Recall that the challenges in negative sampling are (i) how to model the dynamic distribution of negative triplets and (ii) how to sample negative triplets in an efficient way. By considering the key observations, we are motivated to use a small amount of extra memory, which caches negative samples with large scores for each triplet in $\mathcal{S}$, and sample the negative triplet directly from the cache. Algorithm 2 shows the KG embedding framework based on our cache-based negative sampling scheme. Note that the proposed sampling scheme does not depend on the choice of scoring functions, all ones previously mentioned in Section II-C can be used here.

Algorithm 2 NSCaching: Cache-based KG embedding.

Input: training set $\mathcal{S} = \{(h, r, t)\}$, embedding dimension d, scoring function f, and head cache $\mathcal{H}$, tail cache $\mathcal{T}$.

```
initialize embeddings for each e ∈ E and r ∈ R, head-cache H and tail-cache T;
for i = 1, ··· , T do
    sample a mini-batch S_batch ∈ S of size m;
    for each (h, r, t) ∈ S_batch do
        index the cache H by (r, t) and T by (h, r) to get the candidate sets of heads H_(r,t) and tails T_(h,r).
        core step: sample h̄ ∈ H_(r,t) and t̄ ∈ T_(h,r).
        uniformly pick up (h̄, r, t̄) ∈ {(h̄, r, t), (h, r, t̄)}.
        core step: update the cache H_(r,t) and T_(h,r) using Algorithm 3;
        update embeddings using Equation (3) or (4);
    end for
end for
```

Basically, as a negative triplet can be constructed by either replacing the head or tail entity, we maintain a head-cache $\mathcal{H}$ (indexed by (r, t)) and a tail-cache $\mathcal{T}$ (indexed by (h, r)), which store $\bar{h} \in \mathcal{E}$ and $\bar{t} \in \mathcal{E}$ respectively. Each pair

TABLE I
COMPARISON OF THE PROPOSED APPROACH WITH STATE-OF-THE-ARTS, WHICH ADDRESS THE NEGATIVE SAMPLE. MODEL PARAMETERS ARE BASED ON TRANSE, m IS THE SIZE OF MINI-BATCH, n IS THE EPOCH OF LAZY-UPDATE.

	strategy		minibatch computation		model
	negative sample	training	time	space	parameters
baseline	uniform random	gradient descent (from scratch)	$O(md)$	$O(md)$	$(\|\mathcal{E}\|+\|\mathcal{R}\|)d$
IGAN [33]	GAN	reinforce learning (with pretrain)	$O(m\|\mathcal{E}\|d)$	$O(m\|\mathcal{E}\|d)$	$3(\|\mathcal{E}\|+\|\mathcal{R}\|)d$
KBGAN [8]	GAN	reinforce learning (with pretrain)	$O(mN_1d)$	$O(mN_1d)$	$2(\|\mathcal{E}\|+\|\mathcal{R}\|)d$
NSCaching	using cache	gradient descent (from scratch)	$O(\frac{m}{n+1}(N_1+N_2)d)$	$O(m(N_1+N_2)d)$	$(\|\mathcal{E}\|+\|\mathcal{R}\|)d$

(h,r) or (r,t) corresponds to a unique index. First, when a positive triplet is received, the corresponding cache containing candidates for negative triplets, i.e., $\mathcal{H}_{(r,t)}$ and $\mathcal{T}_{(h,r)}$, are indexed in step 5. A negative triplet is generated from $\mathcal{H}_{(r,t)}$ and $\mathcal{T}_{(h,r)}$ at step 6-7, and then the cache is updated in step 8. Finally, the embeddings are updated based on the choice of scoring functions.

An overview of the proposed method with state-of-the-arts are in Table I. The main difference with general KG embedding framework in Algorithm 1 is step 5-8 in Algorithm 2, where the sampling scheme is based on the cache instead. Besides, compared with previous complex GAN-based works [8], [33], our method in Algorithm 2 acts like a discriminative and distilled model of GAN, which only cares about negative triplets with large scores during the training. Thus, the proposed method, i.e., NSCaching, not only has fewer parameters, but also can be easily trained from randomly initialized models (from the scratch). Moreover, experimental results in Section IV show that NSCaching achieves the best performance.

However, in order to achieve best performance, we need to carefully design how to sample from the cache (step 6) and update the cache (step 8). In the sequel, we will describe the "*exploration and exploitation*" inside these steps and how they are balanced in detail. Then, we give a time and space analysis of Algorithm 2, which further explain its efficiency and memory saving. Note that, we only discuss operations and designs for the head-cache $\mathcal{H}$ here, as designs are the same for the tail-cache $\mathcal{T}$.

1) Uniform sampling strategy from the cache (step 6): Recall that only head $\bar{h}$ in negative triplets with large scores are in cache $\mathcal{H}_{(r,t)}$, thus picking up any $\bar{h} \in \mathcal{H}_{(r,t)}$ probably avoids the vanishing gradient problem. As larger scores also lead to bigger gradients, a very natural scheme is to always sample the negative triplet with the largest score.

However, as the distribution can change during the iterations of the algorithm, the negative triplets in the cache may not be accurate enough for the sampling in the latest iteration. Besides, there are false negative triplets in the negative sample sets, of which scores can also be very high [34]. As a consequence, we also need to consider other triplets except the one with largest score in the cache.

This raises the question that how to keep the balance between *exploration* (i.e., explore all the possible high-quality negative samples) and *exploitation* (i.e., sample the largest score negative triplet in cache).

Algorithm 3 Updating head-cache (step 8).

Input: head cache $\mathcal{H}_{(r,t)}$ of size N_1, triplet $(h,r,t) \in \mathcal{S}$.
1: initialize $\tilde{\mathcal{H}}_{(r,t)} \leftarrow \emptyset$;
2: uniformly sample a subset $\mathcal{R}_m \subset \mathcal{E}$ with N_2 entities;
3: $\hat{\mathcal{H}}_{(r,t)} \leftarrow \mathcal{H}_{(r,t)} \cup \mathcal{R}_m$;
4: compute the score $f(\bar{h},r,t)$ for all $\bar{h} \in \hat{\mathcal{H}}_{(r,t)}$;
5: **for** $i = 1, \cdots, N_1$ **do**
6: sample $\bar{h} \in \hat{\mathcal{H}}_{(r,t)}$ with probability in Equation (6);
7: remove $\bar{h}$ from $\hat{\mathcal{H}}_{(r,t)}$;
8: $\tilde{\mathcal{H}}_{(r,t)} \leftarrow \tilde{\mathcal{H}}_{(r,t)} \cup \bar{h}$;
9: **end for**
10: update by $\mathcal{H}_{(r,t)} \leftarrow \tilde{\mathcal{H}}_{(r,t)}$.

These motivate us to use uniformly random sampling scheme in step 6. It is simple, efficient, and does not introduce any bias into the selection process. Indeed, a stronger scheme can be sampling based on triplets' scores, where larger score indicates higher probability to be sampled. However, it has extra memory costs as scores needs to be stored as well. Moreover, it introduces bias causing by dynamic changing distribution and false negative triplets, which leads to inferior performance as shown in Section IV-C1.

2) Importance sampling strategy to update the cache (step 8): As mentioned in Section II-A, the cache needs to be dynamically changed during the iterations of the algorithm. Otherwise, while negative triplets are kept in $\mathcal{H}_{(r,t)}$, sampling from cache is still a scheme with fixed distribution, which eventually suffers from vanishing gradient problem. Thus, we need to refresh the cache in each iteration. Moreover, the cache needs to be updated in an efficient way.

The proposed importance sampling (IS) strategy is presented in Algorithm 3. First, we uniformly sample a subset $\mathcal{R}_m \subset \mathcal{E}$ of size N_2 (step 2), then union it with $\mathcal{H}_{(r,t)}$ and obtain $\hat{\mathcal{H}}_{(r,t)}$. The scores for all triplets in $\hat{\mathcal{H}}_{(r,t)}$ are evaluated in step 4. After that, we construct a subset $\tilde{\mathcal{H}}_{(r,t)}$ from $\hat{\mathcal{H}}_{(r,t)}$ by sampling entries in $\hat{\mathcal{H}}_{(r,t)}$ without replacement N_1 times following probability

$$p(\bar{h}|(t,r)) = \frac{\exp(f(\bar{h},r,t))}{\sum_{h_i \in \hat{\mathcal{H}}_{(r,t)}} \exp(f(\bar{h}_i,r,t))}. \tag{6}$$

Finally, $\tilde{\mathcal{H}}_{(r,t)}$ is returned as the updated head-cache.

Note that exploration and exploitation also need to be carefully balanced in Algorithm 3. As the cache needs to be updated, we have to sample from $\mathcal{E}$, and uniform sampling

is chosen due to its efficiency. Thus, a bigger N_1 implies more exploitation, while a larger N_2 leads to more exploration. In step 6, indeed, uniform sampling or keeping triplets with top N_1 scores can be alternative choices. However, both of them are inappropriate. First, uniformly sampling is obviously not proper, as triplets in $\mathcal{H}_{(r,t)}$ have much larger scores than those in $\mathcal{R}_m$. Then, deterministically sampling top N_1 is not appropriate as well, which again dues to the existence of false negative triplets (Section III-B1). All above concerns will also be empirically studied in experiments Section IV.

3) Space and time complexities: Here, we analyze the space and time complexities of NSCaching (Algorithm 2). Comparing with basic Algorithm 1, the main additional cost by introducing cache comes from Algorithm 3 in step 8. In Algorithm 3, the time complexity of computing the score of $N_1 + N_2$ candidate triplets $f(\bar{h}, r, t)$ is $O((N_1 + N_2)d)$. The cost of step 6 contains two parts, i.e., normalization of the score and uniform sampling, they take $O(N_1 + N_2)$ and $O(N_1)$ respectively, which are very small. Thus, the total cost of introducing cache is $O((N_1 + N_2)d)$ for one triplet. We can lazily update the cache n epochs later rather than immediately updating, which can further reduce update complexity to $O((N_1 + N_2)d/(n + 1))$.

As for space complexity, evaluating the scores for $N_1 + N_2$ candidate triplets takes $O((N_1 + N_2)d)$ space. Since we only store indices in the cache, it takes $O(|\mathcal{S}|N_1)$ space to store these indices for negative triplets. However, since there are many one-to-many, many-to-one and many-to-many relations, the cost will be smaller than $O(|\mathcal{S}|N_1)$ and the cache does not need to be stored in memory. In our experiments, values of N_1 and N_2 used on WN18 and FB15K are both 50, and are 30 for WN18RR and FB15K237, which are much smaller than the number of entities.

In comparison, to generate one negative triplet, the generator in IGAN [33] costs $O(|\mathcal{E}|d)$ time since it needs to compute the distribution over all entities. KBGAN [8] needs $O(N_1 d)$ cost for measuring a candidate set of N_1 triplets. The additional space cost for IGAN and KBGAN is also $O(|\mathcal{E}|d)$ and $O(N_1 d)$ respectively. Finally, the comparisons are summarized in Table I with TransE as the scoring function.

4) Discussion on the Convergence: Both the baseline KG embedding models [34] and NSCaching use stochastic gradient descent (SGD) for model training. While there is no theoretical guarantee, SGD has been applied on many non-convex and complex models [17], where the convergence is empirically observed, including the baseline KG embedding model [6], [7], [9], [22], [26], [32], [36]. The only difference of NSCaching to that baseline model is the way to sample negative triplets.

Besides, since NSCaching samples negative triplets with larger scores, its gradients have larger magnitude than that of baseline approaches. This also prevents NSCaching from being early stopped by the sampling process and helps the embedding model to converge with higher testing performance than that of baseline approaches. The above are all empirically shown and studied in Section IV.

C. Connection to Self-Pace Learning

The main idea of self-paced (or curriculum) learning [3], [19] is to pick up easy samples first, and then gradually switch to hard ones. In this way, the classifier can first identify the rough position where the decision boundary should locate, and then the boundary can be further refined by the nearby hard examples. It is very effective for complex and noncovex models.

Recently, it is also introduced into network embedding and a big improvement on embedding's quality has been reported [10]. Besides, GAN is also used to monitor the distribution of edges in the network, and negative edges with scores above one threshold are sampled from the generator in GAN. Self-paced learning is achieved by increasing the threshold during the training of embedding [10]. Thus, we can see neither KBGAN nor IGAN has benefited from self-paced learning.

In contrast, our caching scheme can explicitly benefit from it. The reason is that the embedding model only has weak discriminative ability in the beginning of the training. Thus, while there are still a lot of negative triplets with large scores, it is more likely that they are easy ones as most of negative samples are easy. However, as training goes on, those easy samples will gradually have small scores and are removed from the cache. These mean NSCaching will learn from easy samples first, but then gradually focus on hard ones, which is exactly the principle of self-paced learning. The above explanations are also verified by experiments, where we can see the negative triplets in the cache change from easy to hard ones (Section IV-F) and NSCaching training from scratch can already achieve better performance than IGAN and KBGAN with pretrain (Section IV-B).

IV. Experiments

In this section, we carry empirical study of our method. All algorithms are written in Python with PyTorch framework [28] and run on a TITAN Xp GPU. [1]

A. Experiment Setup

1) Datasets: Four datasets are used here, i.e., WN18, FB15K and their variants WN18RR, FB15K237. WN18 and FB15K are firstly introduced in [7]. They are widely tested among the most famous Knowledge Graph embedding learning works [7], [8], [16], [32], [33]. WN18RR and FB15K237 are variants that remove near-duplicate or inverse-duplicate relations from WN18 and FB15K, and are introduced by [35] and [31] respectively. The two variants are harder and more realistic. Their statistics are shown in Table II.

TABLE II
Detailed information of the datasets used in experiments

Dataset	#entity	#relation	#train	#valid	#test
WN18	40,943	18	141,442	5,000	5,000
WN18RR	40,943	11	86,835	3,034	3,134
FB15K	14,951	1,345	484,142	50,000	59,071
FB15K237	14,541	237	272,115	17,535	20,466

[1]Code available in https://github.com/yzhangee/NSCaching.

Specifically, WN18 and WN18RR are subsets of Wordnet [24], which is a large lexical database of English. The entities correspond to word senses, and relations mean the lexical relation between them. FB15K and FB15K237 are subsets of Freebase dataset [4] which contains general facts of the world. Freebase keeps growing until January 2014 and it now contains approximately 44 million topics and 2.4 billion triplets.

2) Tasks: Following previous KG embedding works [7], [16], [32], [36], and the GAN-based works [8], [33], we test the performance on *link prediction* task. This is also the testbed to measure KG embedding models. Link prediction aims to predict the missing entity h or t for a positive triplet (h, r, t). In this task, we measure the rank of head entity h and tail entity t among all the entity sets. Thus, link prediction emphasizes the rank of the correct entity rather than their concrete scores.

3) Performance measurements: As in previous works [7], [8], [32], [33] , we evaluate different models based on the following metrics:

- Mean reciprocal ranking (MRR): It is computed by average of the reciprocal ranks $1/|\mathcal{S}| \sum_{i=1}^{|\mathcal{S}|} \frac{1}{\text{rank}_i}$ where $\text{rank}_i, i \in \{1, \ldots, |\mathcal{S}|\}$ is a set of ranking results;
- Hit@10: The percentage of appearance in top-10: $1/|\mathcal{S}| \sum_{i=1}^{|\mathcal{S}|} \mathbb{I}(\text{rank}_i < 10)$, where $\mathbb{I}(\cdot)$ is the indicator function;
- Mean rank (MR): It is computed by $\frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \text{rank}_i$. Smaller value of MR tends to infer better results.

MRR and Hit@10 measure the top rankings of positive entity in different level. Hit@10 cares about general top rankings, and the top 1 samples contribute most to MRR. The larger value of MRR and Hit@10 indicates better performance. To avoid underestimating the performance of different models, we report the performance in a "Filtered" setting, i.e., all the corrupted triplets that exist in train, valid and test set are filtered out [8], [33]. Note that, MR is not a good metric, as it is easily influenced by false positive samples. We report it here to keep consistent with existing literatures [8], [33].

4) Choices of the scoring function: A large amount of scoring functions have been proposed in literature, please see a recent survey [34] for a review. Here, following [8], [33], TransE [7], TransH [36], TransD [16], DistMult [38] and ComplEx [32] will be used as scoring functions (Table III) for comparison.

TABLE III
DEFINITIONS OF DIFFERENT SCORING FUNCTIONS. ALL MODEL EMBEDDINGS ARE REAL VALUES, EXCEPT COMPLEX ARE COMPLEX VALUES. $Re(\cdot)$ TAKES THE REAL PART OUT OF COMPLEX NUMBERS, $conj(\cdot)$ IS THE CONJUGATE OF $\mathbf{t} \in \mathbb{C}^d$. $\langle \cdot, \cdot, \cdot \rangle$ IS THE INNER PRODUCT.

model	scoring function	definition
translational distance	TransE [7]	$\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_1$
	TransH [36]	$\left\|\mathbf{h} - \mathbf{w}_r^\top \mathbf{h} \mathbf{w}_r + \mathbf{r} - (\mathbf{t} - \mathbf{w}_r^\top \mathbf{t} \mathbf{w}_r)\right\|_1$
	TransD [16]	$\left\|\mathbf{h} + \mathbf{w}_r \mathbf{w}_h^\top \mathbf{h} + \mathbf{r} - (\mathbf{t} + \mathbf{w}_r \mathbf{w}_t^\top \mathbf{t})\right\|_1$
semantic matching	DistMult [38]	$\langle \mathbf{h}, \mathbf{r}, \mathbf{t} \rangle$
	ComplEx [32]	$\text{Re}\left(\langle \mathbf{h}, \mathbf{r}, \text{conj}(\mathbf{t}) \rangle\right)$

B. Comparison with State-of-the-arts

In this section, we focus on the comparison with state-of-the-arts methods. Hyper-parameters of *NSCaching* are studied in Section IV-C.

1) Compared methods: Following methods for negative sampling are compared:

- *Bernoulli* [36]: As a basic extension of the uniform sampling scheme used in TransE, Bernoulli sampling aims at reducing false negative labels by replacing the head or tail with different probability for one-to-many, many-to-one and many-to-many relations. Specifically, it samples $(\bar{h}, r, t)$ or $(h, r, \bar{t})$ under a predefined Bernoulli distribution. Since it is shown to be better than uniform sampling, we choose it as the basic random sampling scheme;
- *KBGAN* [8]: This model firstly samples a set $\mathcal{N}eg$ uniformly from the whole entity set $\mathcal{E}$. Then head or tail entity is replaced with the entities in $\mathcal{N}eg$ to form a set of candidate $(\bar{h}, r, t)$ and $(h, r, \bar{t})$. The generator in KBGAN tries to pick up one triplet among them. As proposed in [8], we choose the simplest model TransE as the generator. For fair comparison, the size of set $\mathcal{N}eg$ is same as our cache size N_1. We use the published code [2] and change the configure same as ours for fair comparison;
- *NSCaching* (Algorithm 2): As in Section III, the negative samples are formed by replacing the head entity h or tail entity t with one uniformly sampled from head cache $\mathcal{H}$ or tail cache $\mathcal{T}$. The cache is updated as in Algorithm 3. Note that we can also lazily update the cache several iterations later, which can further save time. However, we just report the result of immediate update, which is shown to be both effective and efficient. We use $N_1 = N_2 = 50$ and lazy-update with $n = 0$ unless otherwise specified.

As the source code of *IGAN* [33] is not available, we do not compare with it here. Instead, we directly use the reported performance in the sequel. Finally, we also use Bernoulli sampling to choose between $(\bar{h}, r, t)$ and $(h, r, \bar{t})$ for *KBGAN* and *NSCaching*. Besides, as suggested in [8], [33], two training strategies are used for *KBGAN* and *NSCaching*, i.e.,

- From *scratch*: The embedding of relations and entities are initialized by the Xavier uniform initializer [12], and the models (denoted as *KBGAN + scratch* and *NSCaching + scratch*) are directly applied to train the given KG;
- With *pretrain*: Same as [8], [33], we firstly pretrain each scoring function under the baseline model, i.e. *Bernoulli* sampling, several epochs on both data sets. We denote it as *pretrained*. Then the obtained parameters are used to warm-start the given KG rather than from scratch. We keep training based on the warm-started KG embedding and evaluate the performance under different models, i.e., *Bernoulli*, *KBGAN + pretrain* and *NSCaching + pretrain*. Besides, the generator in *KBGAN* is warm-started with corresponding TransE model.

2) Hyper-parameter settings: We use grid search to select the following hyper-parameters: hidden dimension $d \in \{50, 100, 200\}$, batch size $m \in \{1024, 2048, 4096\}$, learning rate $\eta \in \{0.0001, 0.001, 0.01, 0.1\}$. For translational distance models, we tune the margin value $\gamma \in \{1, 2, 3, 4\}$. And

[2]https://github.com/cai-lw/KBGAN

TABLE IV

COMPARISON OF VARIOUS ALGORITHMS ON THE FOUR DATASET. PERFORMANCE OF THE *pretrained* MODEL IS INCLUDED AS REFERENCE. AS CODE OF IGAN IS NOT AVAILABLE, ITS PERFORMANCE IS DIRECTLY COPIED FROM [33]. NOTE THAT MRR, AND THOSE ON WN18RR, FB15K237 DATASETS ARE NOT REPORTED AS THEY ARE NOT SHOWN [33]. BOLD NUMBER MEANS THE BEST PERFORMANCE, AND UNDERLINE MEANS THE SECOND BEST.

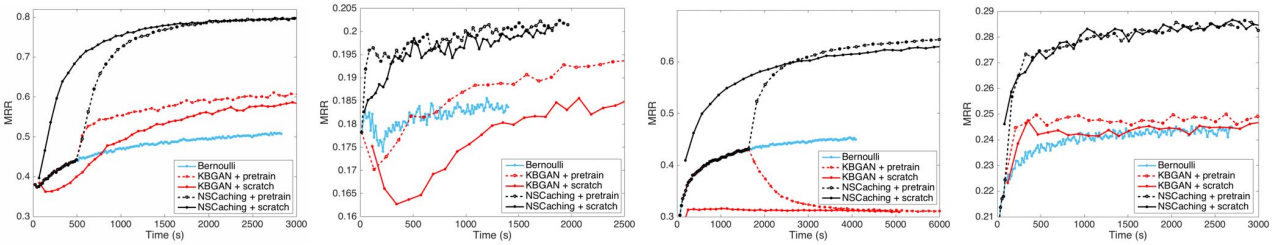
scoring	Dataset		WN18			WN18RR			FB15K			FB15K237		
functions	Metrics		MRR	MR	Hit@10	MRR	MR	Hit@10	MRR	MR	Hit@10	MRR	MR	Hit@10
TransE	pretrained		0.4213	**217**	91.50	0.1753	4038	44.48	0.4679	**60**	74.70	0.2262	237	38.64
	Bernoulli		0.5001	249	94.13	0.1784	**3924**	45.09	0.4951	65	77.37	0.2556	197	41.89
	KBGAN	pretrain	0.6880	293	**94.92**	0.1864	4420	45.39	0.4858	82	77.02	0.2938	628	46.69
		scratch	0.6606	301	94.80	0.1808	5356	43.24	0.3771	335	72.67	0.2926	722	46.59
	NSCaching	pretrain	**0.7867**	271	**66.62**	**0.2048**	4404	47.38	**0.6475**	62	**81.54**	**0.3004**	188	47.36
		scratch	0.7818	249	94.63	0.2002	4472	**47.83**	0.6391	62	80.95	0.2993	**186**	**47.64**
	IGAN*	pretrain	——	240	91.3	——	——	——	——	81	74.0	——	——	——
		scratch	——	244	92.7	——	——	——	——	90	73.1	——	——	——
TransH	pretrained		0.4527	**233**	92.71	0.1755	5646	43.30	0.4316	58	73.98	0.2222	223	38.80
	Bernoulli		0.5206	288	94.52	0.1862	**4113**	45.09	0.4518	60	76.55	0.2329	202	40.10
	KBGAN	pretrain	0.6168	335	94.84	0.1923	4708	45.31	0.4262	86	75.91	0.2807	401	46.39
		scratch	0.6018	288	94.60	0.1869	4881	44.81	0.3364	311	72.53	0.2779	455	46.19
	NSCaching	pretrain	**0.8063**	286	**95.32**	0.2038	4425	**48.04**	**0.6520**	**54**	**81.96**	0.2812	187	46.48
		scratch	0.8038	266	95.29	**0.2041**	4491	**48.04**	0.6391	**54**	81.05	**0.2832**	**185**	**46.59**
	IGAN*	pretrain	——	258	94.0	——	——	——	——	81	77.0	——	——	——
		scratch	——	276	86.9	——	——	——	——	90	73.3	——	——	——
TransD	pretrained		0.4426	243	92.69	0.1782	4955	42.18	0.4320	59	73.98	0.2244	215	39.53
	Bernoulli		0.5093	256	94.61	0.1901	3555	46.41	0.4529	63	76.55	0.2451	188	42.89
	KBGAN	pretrain	0.6130	307	94.92	0.1917	3785	46.49	0.4069	75	74.27	0.2487	798	44.33
		scratch	0.5950	332	94.68	0.1875	4083	46.41	0.3151	184	69.77	0.2465	825	44.40
	NSCaching	pretrain	**0.8022**	295	94.99	**0.2013**	**2952**	48.36	**0.6567**	**54**	**82.02**	**0.2883**	**184**	**48.33**
		scratch	0.7994	286	**95.16**	**0.2013**	3104	**48.39**	0.6415	58	81.32	0.2863	189	47.85
	IGAN*	pretrain	——	248	93.3	——	——	——	——	79	77.6	——	——	——
		scratch	——	**221**	93.0	——	——	——	——	89	74.0	——	——	——
DistMult	pretrained		0.6340	1174	92.28	0.3765	**7405**	44.85	0.4985	94	78.28	0.2247	408	36.03
	Bernoulli		0.7918	**862**	**93.38**	0.3964	7420	45.25	0.5376	102	78.69	0.2491	280	42.03
	KBGAN	pretrain	0.6955	1143	93.11	0.3849	7586	44.32	0.5568	201	75.57	0.2670	370	45.34
		scratch	0.7275	794	93.08	0.2039	11351	29.52	0.4227	321	64.35	0.2272	276	39.91
	NSCaching	pretrain	0.8297	1038	93.83	**0.4148**	7477	**45.80**	0.7447	**81**	**84.16**	**0.2882**	**265**	**45.79**
		scratch	**0.8306**	827	93.74	0.4128	7708	45.45	**0.7448**	**81**	83.91	0.2834	273	45.56
ComplEx	pretrained		0.8046	1106	93.75	0.3934	8259	41.63	0.5191	**85**	78.02	0.2201	418	35.55
	Bernoulli		0.9115	**808**	**94.39**	0.4431	**4693**	**51.77**	0.6253	138	80.72	0.2596	238	43.54
	KBGAN	pretrain	0.8976	1060	93.73	0.4287	6929	47.03	0.6254	162	80.95	0.2818	268	45.37
		scratch	0.7233	966	85.81	0.3180	7528	35.51	0.5002	294	76.10	0.1910	881	32.07
	NSCaching	pretrain	0.9326	1079	94.03	**0.4487**	4861	51.76	0.7994	94	**86.32**	0.3017	**220**	47.75
		scratch	**0.9355**	1072	93.98	0.4463	5365	50.89	**0.7995**	94	86.28	**0.3021**	221	**48.05**

for semantic matching models, we tune the penalty value $\lambda \in \{0.001, 0.01, 0.1\}$ [32]. We use Adam [17], which is a popular variant of SGD algorithm for the training, and adopt its default settings, except for the learning rate. The best hyper-parameter is tuned under Bernoulli sampling scheme and evaluated by the MRR metric on validation set. We keep them fixed for the baseline methods *Bernoulli*, *KBGAN* and our proposed *NSCaching*. Following [8], we save and record the *pretrained* model after several initial training epochs. Then, *Bernoulli* method keeps training until 3000 epochs; and the results of *KBGAN* and *NSCaching* algorithm are evaluated within 1000 epochs, either from scratch or with pretrain. All the recorded results are tested based on the best parameters chosen by the MRR value on validation set. [3]

3) Results on translational distance models: The performance on link prediction is compared in Table IV. First, we can see that, for the translational distance models (TransE, TransH, TransD), *KBGAN*, *NSCaching* and *IGAN* (both *with pretrain* and *from scratch*) gain significant improvement upon the baseline scheme *Bernoulli*, especially for the gaining on MRR, which is mainly influenced by top rankings. This verifies the needs of using high-quality negative triplets during negative sampling and these methods can effectively pick up these negative triplets.

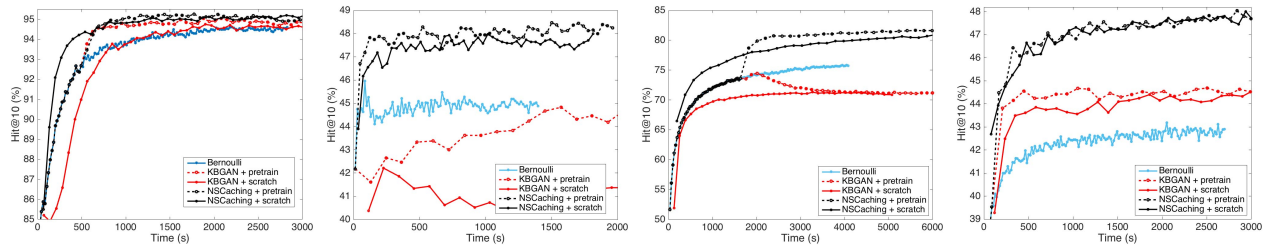
Then, *IGAN* and *KBGAN* with pretrain can perform better,

[3]More experiments on triplets and relation classification are in [40].



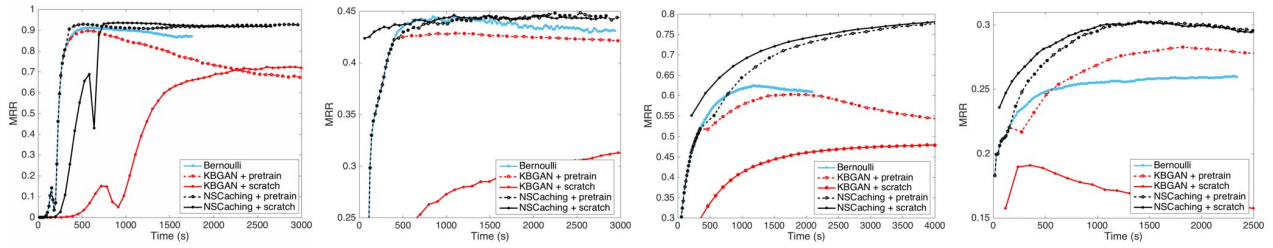
(a) WN18. (b) WN18RR. (c) FB15K. (d) FB15K237.

Fig. 2. Testing MRR performance v.s. clock time (in seconds) based on TransD.



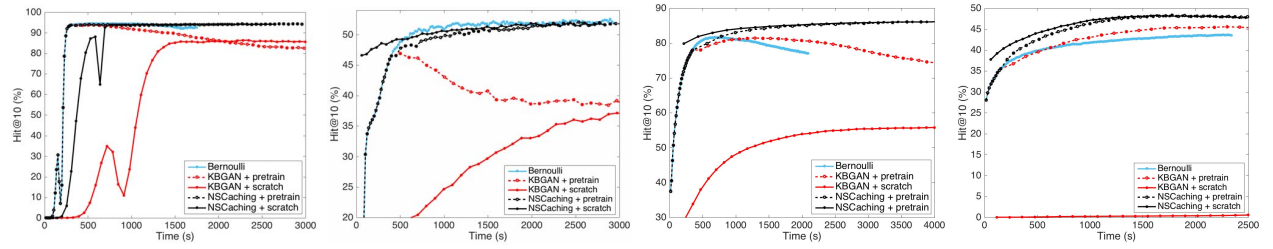
(a) WN18. (b) WN18RR. (c) FB15K. (d) FB15K237.

Fig. 3. Testing Hit@10 performance v.s. clock time (in seconds) based on TransD.



(a) WN18. (b) WN18RR. (c) FB15K. (d) FB15K237.

Fig. 4. Testing MRR performance v.s. clock time (in seconds) based on ComplEx.



(a) WN18. (b) WN18RR. (c) FB15K. (d) FB15K237.

Fig. 5. Testing Hit@10 performance v.s. clock time (in seconds) based on ComplEx.

indicated by MRR and Hit@10, than from scratch. This shows pretrain is helpful for GAN based models. In comparison, the proposed *NSCaching* trained from either state (pretrain or scratch) can outperform *IGAN* and *KBGAN*. Finally, we find that MR is not an appropriate metric, as many of the *pretrained* models, which is not converged yet, show even smaller MR than the *Bernoulli*.

Convergence of testing performance for various algorithms are shown in Figure 2 and 3. We use TransD as it offers the best performance among the three translational distance models. As can be seen, all algorithms will converge to a stable testing MRR and Hit@10, which verifies the empirical convergence of Adam optimizer. Then, while pretrain is a must for *KBGAN* to achieve good performance, *NSCaching* can obtain good performance either from scratch or using pretrain. Finally, in all cases, *NSCaching* converges much faster and is more stable than both *Bernoulli* and *KBGAN*.

4) Results on semantic matching models: The performance is shown in the bottom rows of Table IV. Same as the performance on translational distance models, *NSCaching* outperforms baseline scheme *Bernoulli* significantly, as indicated by the bold and underline numbers. However, *KBGAN* does not show consistent performance. It performs even worse than the *Bernoulli* sampling scheme on WN18, WN18RR and FB15K, *KBGAN from scratch* even performs much worse than *with pretrian*. This observation further verifies the fact that GAN

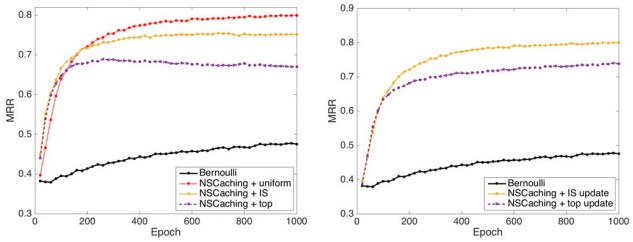
based methods usually suffer from instability and degeneracy. This method needs careful balance between the generator and the target KG embedding model. However, *NSCaching* works consistently and performs the best among various settings.

Convergence of testing performance for various algorithms are shown in Figure 4 and 5. We use ComplEx as the representative since it is much better than DistMult. As can be seen, both *Bernoulli* and the proposed *NSCaching* will converge to a stable state. In the contrast, *KBGAN* will turn down and overfit after several epochs. However, *NSCaching*, either with pretrain or from scratch, leads the performance and is well adopted on the semantic matching models without further tuning.

C. Cache Update and Sampling Scheme

In Section IV-B, we have shown that NSCaching achieves the best performance on four benchmark datasets. Here, we analyze design concerns on "exploration and exploitation" at step 6 and 8 in Algorithm 2. TransD and WN18 are used here.

1) Uniform sampling from the cache (step 6): Given a cache, which stores high-quality negative samples, how to sample from it is the first question we care about. Recall that we discussed three strategies in Section III-B1, i.e., (i) uniform sampling from the cache (dented as "uniform sampling"); (ii) importance sampling according to the score of each sample in cache (denoted as "IS sampling"); and (iii) top sampling, by choosing the sample with largest score (denoted as "top sampling"). Testing performance of MRR on WN18 trained by TransD are compared in Figure 6.(a). As can be seen, top sampling has the worst performance, and uniform sampling is the best.

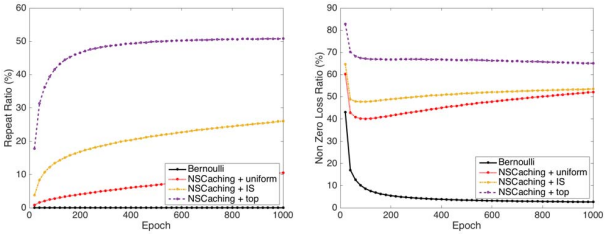


(a) Diff. sampling strategies. (b) Diff. cache updating strategies.

Fig. 6. Comparison on testing MRR v.s. epoch of different sampling strategies and cache updating strategies. Evaluated by TransD model on WN18.

To show how exploration and exploitation are balanced here, we further compute two criterion to show the difference between these strategies. (i) Repeat ratio (denoted as "RR"), which measures the percentage of repeated negative triplets $(\bar{h}, r, \bar{t})$ within 20 epochs; and (ii) non-zero loss ratio (denoted as "NZL"), which is the percentage of non-zero losses in same range. The value of RR is related to *exploration*, if the number of repeated negative triplets is high, the negative samples only explore a small part of the sample spaces, thus result in worse exploration. NZL ratio measures *exploitation*, a larger NZL means higher quality of picked negative samples.

The RR is shown in Figure 7(a). The Bernoulli sampling method has almost zero repeat triplets since the number of explored negatives is extremely large, it has the best exploration. Among the schemes based on NSCaching, uniform sampling has better exploration than IS, then followed by top sampling. NZL ratio is shown in Figure 7(b). As training going on, the baseline Bernoulli model suffers the zero loss problem severely, thus leading to vanishing gradient. All of the three schemes have more than half non-zero losses, thus achieves exploitation. To sum up, uniform sampling is the most balanced strategy among the three schemes, thus *NSCaching + uniform* achieving the best performance.



(a) RR v.s. epoch. (b) NZL v.s. epoch.

Fig. 7. Balancing on exploration (left) and exploitation (right) of different sampling strategies. Evaluated by TransD model on WN18.

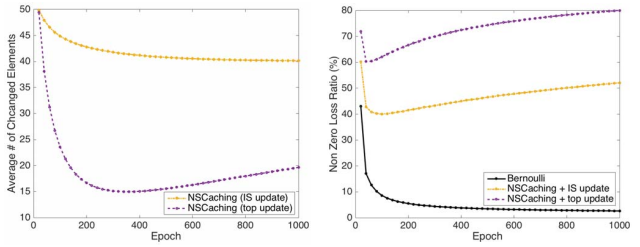
2) Importance sampling strategy to update the cache (step 8): As discussed in Section III-B2, we have two choices on updating the cache: (i) importance sampling based method, which samples N_1 entities from N_1+N_2 candidates according to the probability in (6) without replacement, (IS update). (ii) top sampling method, which directly select N_1 entities with top scores in the candidates, (top update). Again, let us first look at performance comparison in Figure 6.(b). We can see that IS update outperforms top update by a large margin.

Then, to explain the exploration and exploitation here, we add two extra measurements for comparison. They are (i). the number of changed elements in cache (denoted as "CE") and (ii) the ratio of non-zero losses, i.e., NZL. More changed elements leads to larger exploration, and more nonzero losses means more exploitation.

The value of CE measures the different elements in the cache in a period of epochs. As shown in Figure 8.(a), the number of changed elements in top update scheme is much smaller than that of the importance sampling update. As a result, the cache is updated quite slow and the model mainly focuses on these highly scored negative triplets, which may contain many false positive triplets. As a comparison, the importance sampling based update scheme can keep the cache fresh and keep track of dynamic changes of the negative sampling distribution. It not only provides enough qualified negative triplets for the KG embedding model to avoid zero loss, but also explore the large negative sample space well. In summary, we choose the importance sampling strategy to update the cache.

D. Sensitivity Analysis: Cache Size

Comparing with the baseline KG embedding models (i.e., Bernoulli [21], [36]), the only extra hyper-parameters here are N_1 and N_2. Basically, N_1 is the size of cache. Then, N_2 is

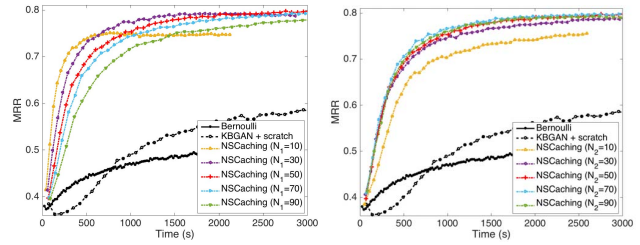


(a) CE v.s. epoch. (b) NZL v.s. epoch.

Fig. 8. Balancing on exploration (left) and exploitation (right) of different strategies for updating the cache. Evaluated by TransD model on WN18.

the size of randomly sampled negative triplets from $\bar{\mathcal{S}}_{(h,r,t)}$, which will be later used to update the cache. Here, we show their impact on NSCaching's performance.

Figure 9.(a) shows how performance changes by varying the cache size N_1 among $\{10, 30, 50, 70, 90\}$, with fixed $N_2 = 50$. When the cache size is small, average score of entities stored in cache should be larger than those in larger cache. Thus, false negative samples will be more likely to be sampled, which will influence the boundary to a bad location. As for the others values of N_1, NSCaching performs quite stable. The convergence speed is similar, as well as the values in converged state. Thus, when finding appropriate cache size, the value of N_1 can be searched from smaller value until the performance is stable.



(a) Diff. N_1. (b) Diff. N_2.

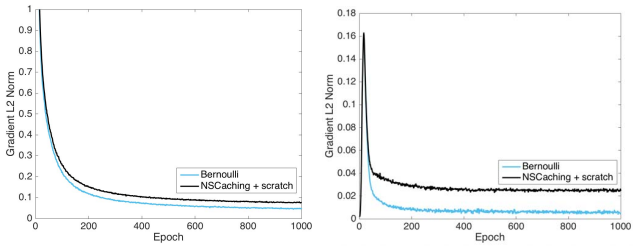
Fig. 9. Comparison of different N_1 when random subset size N_2 is fixed to 50, and different N_2 when cache size N_1 is fixed to 50. Evaluated by TransD model on WN18.

Different performance of the random candidate subset size N_2 is shown in Figure 9.(b). Obviously, the entities in cache will be updated more frequently when N_2 gets larger, which lead to better exploration. But the trade-off is that larger value of N_2 costs more. As shown by the colored lines in Figure 9.(b), NSCaching performs consistently when N_2 is larger than 10. However, if the random subset is small, the content in cache will be harder to be updated, thus lead to poor performance as the yellow dashed line ($N_2 = 10$).

By combining together the influence of cache size N_1 in Figure 9 and random subset size N_2 in Figure 9.(b), we find that (i) NSCaching is not sensitive to the two sizes; (ii) both sizes can not be too small; (iii) $N_1 = N_2$ is a good balance.

E. Illustration of Vanishing Gradients

To further clarity the vanishing gradient problem, we plot average ℓ_2-norm of gradients v.s. number of epochs in Figure 10. Note that Adam [17], which is a stochastic gradient descent algorithm, is used as the optimizer. First, we can see that while norms of gradients for both *NSCaching* and *Bernoulli* become smaller, they will not decrease to zero since the sampling process of the mini-batch will introduce noisy into gradients. However, the norm from *NSCaching* is larger than that from *Bernoulli*, which dues to the usage of caching-based negative sampling scheme. Thus, we can see *NSCaching* can successfully avoid the problem of vanishing gradient.



(a) TransD. (b) ComplEx.

Fig. 10. Mini-batch's average ℓ_2-norm of gradients in one epoch v.s. number of epochs for Bernoulli and NSCaching on WN18RR.

F. Explanation of the connection to Self-Paced Learning

Finally, we visualize the changes of entities in the cache, which verifies the effects of self-paced learning introduced in Section III-C. Following [33], we also use FB13 here since its triplets are more interpretable than the four evaluated datasets. We pick up $(manorama, profession, actor)$ as the positive triplet, and the changes in its tail-cache are show in Table V. As can be seen, entities are firstly meaninglessness, e.g., *ostrava* and *ben_lilly*, then they gradually changes to human jobs, e.g., *artist* and *sex_worker*.

TABLE V
EXAMPLES OF NEGATIVE ENTITIES IN CACHE ON FB13. EACH LINE DISPLAYS 5 RANDOM SAMPLED NEGATIVE ENTITIES FROM TAIL CACHE OF A POSITIVE FACT *(manorama, profession, actor)* IN DIFFERENT EPOCHS.

epoch	entities in cache
0	*allen_clarke, jose_gola, ostrava, ben_lilly, hans_zinsser*
20	*accountant, frank_pais, laura_marx, como, domitia_lepida*
100	*artist, , aviator, hans_zinsse, john_h_cough*
200	*physician, artist, raich_carter, coach, mark_shivas*
500	*artist, physician, cavan, sex_worker, attorney_at_law*

V. RELATED WORK

1) Generative Adversarial Network: Generative Adversarial Network (GAN) is originally introduced as a powerful model for plausible image generation. The GAN contains two modules: a *generator* that serves as a complex distribution sampler, and a *discriminator* that measures the quality of generated samples. Under elaborately control on the training procedure of generator and discriminator, GAN achieved significant success computer vision field [1], [15]. It has been shown to sample high-quality negative samples for knowledge graph embedding [8], [33].

2) Negative Sampling: Negative sampling important for improving learning model when there are only positive samples. Its typical applications include word embedding [23], graph embedding [14], and KG embedding [34] here. More recently,

there have been interests in applying the GAN to negative sampling, e.g., IGAN [33] and KBGAN [8] for KG embedding and self-paced GAN [10] for network embedding.

VI. Conclusion

We proposed NSCaching as a novel negative sampling method for knowledge graph embedding learning. The negative samples are from a cache that can dynamically hold high-quality negative samples. We analyze the designing of NSCaching through the balance of exploration and exploitation. Experimentally, we test NSCaching on four datasets and five scoring functions. Results show that the method can generalize well under various settings and achieves state-of-the-arts performance on benchmark datasets.

As future works, we consider using distributed computation or hashing to improve the scalability of NSCaching, and automated machine learning [39] to make NSCaching easier to be used. Besides, it is also interesting to study the theoretical convergence of NSCaching.

Acknowledgment

The work is partially supported by the Hong Kong RGC GRF Project 16202218, the National Science Foundation of China (NSFC) under Grant No. 61729201, Science and Technology Planning Project of Guangdong Province, China, No. 2015B010110006, Hong Kong ITC ITF grants ITS/391/15FX and ITS/212/16FP, Didi-HKUST joint research lab project, Microsoft Research Asia Collaborative Research Grant and Wechat Research Grant. Yingxia Shao's work is supported by NSFC No. 61702015.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. Technical report, ICLR, 2017.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A nucleus for a web of open data. In *The Semantic Web*, pages 722–735. Springer, 2007.

[3] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *ICML*, pages 41–48. ACM, 2009.

[4] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250. ACM, 2008.

[5] A. Bordes, S. Chopra, and J. Weston. Question answering with subgraph embeddings. In *EMNLP*, pages 615–620, 2014.

[6] A. Bordes, X. Glorot, J. Weston, and Y. Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259, 2014.

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

[8] L. Cai and W.Y. Wang. KBGAN: Adversarial learning for knowledge graph embeddings. In *ACL*, volume 1, pages 1470–1480, 2018.

[9] M. Fan, Q. Zhou, E. Chang, and T. F. Zheng. Transition-based knowledge graph embedding with relational mapping properties. In *PACLIC*, 2014.

[10] H. Gao and H. Huang. Self-paced network embedding. In *SIGKDD*, pages 1406–1415, 2018.

[11] L. Getoor and B. Taskar. *Introduction to statistical relational learning*, volume 1. The MIT Press, 2007.

[12] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010.

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NeurIPS*, pages 2672–2680, 2014.

[14] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864. ACM, 2016.

[15] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. In *NeurIPS*, pages 5767–5777, 2017.

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

[17] D. Kingma and J. Ba. Adam: A method for stochastic optimization. Technical report, arXiv preprint, 2014.

[18] S. Kok and P. Domingos. Statistical predicate invention. In *ICML*, pages 433–440, 2007.

[19] M. P. Kumar, B. Packer, and D. Koller. Self-paced learning for latent variable models. In *NeurIPS*, pages 1189–1197, 2010.

[20] N. Lao, T. Mitchell, and W. W. Cohen. Random walk inference and learning in a large scale knowledge base. In *EMNLP*, pages 529–539. Association for Computational Linguistics, 2011.

[21] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, volume 15, pages 2181–2187, 2015.

[22] H. Liu, Y. Wu, and Y. Yang. Analogical inference for multi-relational embeddings. In *ICML*, pages 2168–2178, 2017.

[23] T. Mikolov, W. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *ACL*, pages 746–751, 2013.

[24] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

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

[26] M. Nickel, L. Rosasco, and T. A. Poggio. Holographic embeddings of knowledge graphs. In *AAAI*, volume 2, pages 3–2, 2016.

[27] M. Nickel, V. Tresp, and H. Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, volume 11, pages 809–816, 2011.

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. Technical report, arXiv preprint, 2017.

[29] A. Singhal. Introducing the knowledge graph: things, not strings. *Official Google blog*, 5, 2012.

[30] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706, 2007.

[31] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66, 2015.

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

[33] P. Wang, S. Li, and R. Pan. Incorporating GAN for negative sampling in knowledge representation learning. *AAAI*, 2018.

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

[35] Y. Wang, D. Ruffinelli, S. Broscheit, and R.ainer Gemulla. On evaluating embedding models for knowledge base completion. *arXiv preprint arXiv:1810.07180*, 2018.

[36] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, volume 14, pages 1112–1119, 2014.

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

[38] B. Yang, W. Yih, X. He, J. Gao, and L. Deng. Embedding entities and relations for learning and inference in knowledge bases. Technical report, arXiv preprint, 2017.

[39] Q. Yao, M. Wang, Y. Chen, W. Dai, Hu Y., Y. Li, W. Tu, Q. Yang, and Y. Yu. Taking human out of learning applications: A survey on automated machine learning. Technical report, arXiv preprint, 2018.

[40] Y. Zhang, Q. Yao, Y. Shao, and L. Chen. NSCaching: Simple and efficient negative sampling for knowledge graph embedding. Technical report, arvix preprint, 2018.