ABSTRACT
Knowledge Graph (KG) is a flexible structure that is able to describe the complex relationship between data entities. Currently, most KG embedding models are trained based on negative sampling, i.e., the model aims to maximize some similarity of the connected entities in the KG, while minimizing the similarity of the sampled disconnected entities. Negative sampling helps to reduce the time complexity of model learning by only considering a subset of negative instances, which may fail to deliver stable model performance due to the uncertainty in the sampling procedure. To avoid such deficiency, we propose a new framework for KG embedding—Efficient Non-Sampling Knowledge Graph Embedding (NS-KGE). The basic idea is to consider all of the negative instances in the KG for model learning, and thus to avoid negative sampling. The framework can be applied to square-loss based knowledge graph embedding models or models whose loss can be converted to a square loss. A natural side-effect of this non-sampling strategy is the increased computational complexity of model learning. To solve the problem, we leverage mathematical derivations to reduce the complexity of non-sampling loss function, which eventually provides us both better efficiency and better accuracy in KG embedding compared with existing models. Experiments on benchmark datasets show that our NS-KGE framework can achieve a better performance on efficiency and accuracy over traditional negative sampling based models, and that the framework is applicable to a large class of knowledge graph embedding models.

CCS CONCEPTS
- Computing methodologies → Knowledge representation and reasoning; Learning latent representations; Neural networks.

KEYWORDS
Knowledge Graph Embedding; Non-Sampling Machine Learning; Computational Efficiency; Space Efficiency

1 INTRODUCTION
Nowadays, Knowledge Graph (KG) is an important structure to store and process structured information, and has been widely used as an advanced knowledge interlinking solution in many applications. Concretely, KG is a collection of interlinked descriptions of entities. For example, Freebase, which is regarded as a practical and massive tuple database used to structure general human knowledge, is powered by KG algorithms [1]. Yet another Great Ontology (YAGO) also benefits from KG structures when building its light-weight and extensible ontology with high coverage and quality [2]. Moreover, DBpedia builds a large-scale, multilingual knowledge base by extracting structured data from Wikipedia editions in 111 languages [3]. These knowledge bases power a wide range of intelligent systems in practice.

Although KG has been proved as an effective method to represent large-scale heterogeneous data [4], it suffers from high computation and space cost when searching and matching the entities in a discrete symbolic space. In order to leverage the power of KG more efficiently, researchers have proposed Knowledge Graph Embedding (KGE), which represents and manipulates KG entities in a latent space [5]. In particular, KGE techniques embed the components of KG, including entities and relations, into a continuous vector space, so as to simplify the manipulation while preserving the inherent structure of the KG [6]. With the help of KGE, implementing knowledge graph operations to a large scale becomes practical.

Over the past few years, a lot of efforts have been put into developing embedding algorithms, and a growing number of embedding models are proven effective. Most of the current embedding methods, however, depend on a very basic operation in model training called negative sampling [5, 7], which randomly or purposely samples some disconnected entities as negative samples (compared to the connected entities as positive samples), and the embedding model aims to distinguish positive vs. negative samples in the loss function for embedding learning. Typical examples include DistMult [8], Simple [9], ComplEx [10], TransE [5], RESCAL [11], etc. While negative sampling increases the training efficiency, it also
This paper makes the following key contributions:

- We propose a Non-Sampling Knowledge Graph Embedding (NS-KGE) framework for learning effective knowledge graph embeddings.
- We derive an efficient method to mitigate the time and space bottlenecks caused by the non-sampling strategy.
- We demonstrate how the framework can be mathematically applied to existing KGE models by using DistMult [8], SimplE [9], ComplEx [10], and TransE [5] as examples.
- We conduct comprehensive experiments—including both quantitative and qualitative analyses—to show that the framework increases both accuracy and efficiency for knowledge graph embedding.

In the following part of this paper, we first introduce the related work in Section 2. In Section 3, we introduce our NS-KGE framework in detail, and in Section 4 we show how the framework can be applied to different KGE models. We provide and analyze the experimental results in Section 5, and conclude the work together with future directions in Section 6.
We first provide a general formulation for the knowledge graph embedding problem. Given a knowledge graph \( G \), our goal is to learn a scoring function \( f \) which allows us to distinguish whether the head entity \( h \) and tail entity \( t \) are connected by relation \( r \) in the knowledge graph. Suppose \( f_r(h, t) \) is the ground-truth value of the triplet \((h, r, t)\), generated from training sets, and \( \hat{f}_r(h, t) \) is the predicted value by the knowledge graph embedding model, where \( \hat{f}_r(h, t) = 1 \) represents the connected entities, and \( \hat{f}_r(h, t) = 0 \) denotes disconnected entities. Based on these definition, a general KG embedding model aims to minimize the difference between the ground-truth and the predicted values based on a loss function \( L \). For example, we can use square loss to train the model:

\[
L = \sum_{r \in R} \sum_{h \in E} \sum_{t \in E} c_{hrt} \left( f_r(h, t) - \hat{f}_r(h, t) \right)^2
\]  

(1)

where the three summations enumerate all of the possible \((h, r, t)\) triplet combinations in the knowledge graph, and \( c_{hrt} \) represents the importance (i.e., weight score) of the corresponding triplet. In traditional negative sampling-based KGE models such as TransE, \( c_{hrt} = 1 \) is set as the positive instances and the sampled negative instances, while for all other negative instances, \( c_{hrt} = 0 \). In our non-sampling KGE framework, however, all \( c_{hrt} \) values are non-zero. In the simplest case, \( c_{hrt} = 1 \) for all instances, regardless of positive or negative.

Many knowledge graph embedding models can be regarded as a special case of this formulation. For example, TransE uses \( \hat{f}_r(h, t) = ||e_h + r - e_t|| \) as the scoring function. For connected entities, the ground-truth value \( f_r(h, t) \) would be 0, and for disconnected entities, the ground-truth value would be a constant value greater than 0 (e.g., it would be 3 when \( e_h, r \) and \( e_t \) are regularized as unit vectors). Similar to TransE, many other KG embedding models can be represented by Eq.(1) with no or only a little trivial transformation, as we will show later in Section 4.

3 NON-SAMPLING KGE FRAMEWORK

In this section, we will first introduce the notations that will be used in this paper. Then, we will introduce the Non-Sampling Knowledge Graph Embedding (NS-KGE) framework step by step. In particular, we will first provide a general formulation of the framework, and then devote two subsections to show how to improve the time and space efficiency in this work. Based on this, we can achieve three benefits: better entity ranking accuracy, better computational efficiency, and better space efficiency.

### 3.1 Problem Formalization and Notations

In this section, we provide a square-loss based formalization of the Knowledge Graph Embedding (KGE) problem, which will be used in the following parts of the paper. However, we acknowledge that not all of the existing KGE methods can be represented by this square loss formalization. In this paper, we consider those KGE methods whose loss function is a square loss or can be converted into a square loss format for non-sampling KGE learning. Table 1 introduces the basic notations that will be used in this paper. We first provide a general formulation for the knowledge graph embedding problem. Given a knowledge graph \( G \), our goal is to learn a scoring function \( \hat{f}_r(h, t) \), which is able to distinguish whether the head entity \( h \) and tail entity \( t \) should be connected by relation \( r \) in the knowledge graph. Suppose \( f_r(h, t) \) is the ground-truth value of the triplet \((h, r, t)\), generated from training sets, and \( \hat{f}_r(h, t) \) is the predicted value by the knowledge graph embedding model, where \( \hat{f}_r(h, t) = 1 \) represents the connected entities, and \( \hat{f}_r(h, t) = 0 \) denotes disconnected entities. Based on these definition, a general

| Symbol | Description |
|--------|-------------|
| \( G \) | A knowledge graph |
| \( E \) | The set of entities in a knowledge graph |
| \( R \) | The set of relations in a knowledge graph |
| \( h, t \) | A head (h) or a tail (t) entity in a knowledge graph |
| \( r \) | A relation in a knowledge graph |
| \( e_{h,t} \) | Embedding vector of the entity \( h \) and \( t \) |
| \( r_{i,j} \) | Embedding vector of the relation \( r \) |
| \( \sum_{c} \) | Dimension of the embedding vectors |
| \( c_{hrt} \) | The weight of the triplet \((h, r, t)\) |
| \( f_r(h, t) \) | Ground-truth value of the triplet \((h, r, t)\) |
| \( \hat{f}_r(h, t) \) | Predicted value of the triplet \((h, r, t)\) |

Table 1: Summary of the notations in this work.
In this case the loss function can be re-organized as:

\[
L = \sum_{r \in R} \sum_{h \in E_h^+} \sum_{t \in E_t^+} \text{cht}(f_r(h, t) - \hat{f}_t(h, t))^2
\]

\[
+ \sum_{r \in R} \sum_{h \in E_h^-} \sum_{t \in E_t^-} c^r f_r(h, t)^2
\]

\[
+ \sum_{r \in R} \sum_{r \in R} \sum_{h \in E_h} \sum_{t \in E_t} c^r f_r(h, t)^2 + \sum_{r \in R} \sum_{r \in R} \sum_{h \in E_h} \sum_{t \in E_t} c^r f_r(h, t)^2
\]

\[
+ \sum_{r \in R} \sum_{r \in R} \sum_{h \in E_h} \sum_{t \in E_t} c^r f_r(h, t)^2 + \sum_{r \in R} \sum_{r \in R} \sum_{h \in E_h} \sum_{t \in E_t} c^r f_r(h, t)^2
\]

where $E_{h}^+$ represents the set of entities in the KG that are connected to head entity $h$ by relation $r$, while $E_{h}^-$ is the set of entities that are not connected to $h$ through $r$. We have $E_{h}^+ \cup E_{h}^- = E$. In the step 1 above, we split the loss function by considering $E_{h}^+$ and $E_{h}^-$ separately. In step 2, we replace the $E_{h}^-$ term by subtracting $E_{h}^+$ from the total summation, and in step 3, we re-organize the loss function into the positive term $L^P$, all entity term $L^A$, and a constant value term. In the loss function, we separate the positive entities and the constant value, and we use all of the data to replace the negative samples. In the next step, we will introduce how to optimize the time complexity after such transformation.

As we can see, the $L^P$ term enumerates over all of the connected triplets in the KG, and its time complexity is $O(d \times \#)$ positive triplets, which is an acceptable complexity. However, since most of the KG datasets are highly sparse, $L^A$, even with pretty concise form, contributes the most significant time complexity to the loss function. Actually, the time complexity of $L^A$ is $O(d|R||E|^2)$, which is very expensive. We hope the time complexity of $L^A$ can be further reduced. As a result, we will take a closer look at the $f_r(h, t)^2$ term.

Fortunately, NS-KGE can be applied to most of the factorization-based KG embedding models. For these models, we can conduct certain transformations over the scoring function $\hat{f}_r(h, t)$ to reduce the time complexity. For a factorization-based KG embedding model, the scoring function $\hat{f}_r(h, t)$ can be represented as the following general formulation:

\[
\hat{f}_r(h, t) = \mathbf{e}_h^T (r \odot \mathbf{e}_i) = \sum_{i} \mathbf{e}_{h,i}r_{i} \mathbf{e}_{j,i}
\]

where $\mathbf{e}_h$, $r$, and $\mathbf{e}_i$ are the head entity embedding, relation embedding, and tail entity embedding, respectively, and the symbol $\odot$ denotes element-wise product. Besides, $\mathbf{e}_{h,i}$, $r_i$, and $\mathbf{e}_{j,i}$ denote the $i$-th element of the corresponding embedding vector. Since our NS-KGE framework is based on square loss, we calculate the square of $\hat{f}_r(h, t)$. By manipulating the inner-product operation, the square of $f_r(h, t)$ can be rearranged as:

\[
\hat{f}_r(h, t)^2 = \left( \sum_{i} \mathbf{e}_{h,i}r_{i} \mathbf{e}_{j,i} \right) \left( \sum_{j} \mathbf{e}_{h,j}r_{j} \mathbf{e}_{j,j} \right)
\]

\[
= \sum_{i} \sum_{j} \mathbf{e}_{h,i}r_{i} \mathbf{e}_{j,i} \left( \sum_{j} \mathbf{e}_{h,j}r_{j} \mathbf{e}_{j,j} \right) \left( \sum_{j} \mathbf{e}_{h,j}r_{j} \mathbf{e}_{j,j} \right)
\]

(5)

In this case, $\mathbf{e}_h$, $r$, and $\mathbf{e}_i$ are separated from each other, and thus $\sum_{i} \sum_{j} \mathbf{e}_{h,i}r_{i} \mathbf{e}_{j,i}$, $\sum_{i} \sum_{j} \mathbf{e}_{h,j}r_{j} \mathbf{e}_{j,j}$, and $\sum_{i} \sum_{j} \mathbf{e}_{h,i}r_{i} \mathbf{e}_{j,i}$ are independent from each other. Therefore, we can disentangle the parameters and calculate $\hat{f}_r(h, t)^2$ in a more efficient way. We thus apply Eq.(5) to the $L^A$ term of the loss function Eq.(3) and we have:

\[
L = L^P + c^{-1} \sum_{r \in R} \sum_{h \in E_h^+} \sum_{t \in E_t^+} \left( \sum_{i} \mathbf{e}_{h,i}r_{i} \mathbf{e}_{j,i} \right) \left( \sum_{j} \mathbf{e}_{h,j}r_{j} \mathbf{e}_{j,j} \right)
\]

\[
= \sum_{r \in R} \sum_{h \in E_h} \sum_{t \in E_t} c^{-1} \left( \sum_{i} \mathbf{e}_{h,i}r_{i} \mathbf{e}_{j,i} \right) \left( \sum_{j} \mathbf{e}_{h,j}r_{j} \mathbf{e}_{j,j} \right)
\]

(6)

In the second step of Eq.(6), the reason that we can reorganize the summary of $i, j$ and the summary over $h$, $r$, and $t$ is because the $L_H$, $L_R$, and $L_T$ terms are independent from each other. We also leave out the constant term since it does not influence the optimization result.

As noted before, $L^A$ contributes the most significant complexity to the loss function. Based on the above operation, the complexity of $L^A$ is reduced from $O(d|R||E|^2)$ in Eq.(3) to $O(d^2(|E| + |R| + |E|))$ in Eq.(6). In Section 4, we take the Bilinear-Diagonal embedding model (DistMult), the Simple enhancement of Canonical Polyadic model (Simple), the Complex Embedding model (ComplEx) and the Translational Embedding model (TransE) as examples to show how the NS-KGE framework can be applied to different models.

### 3.4 Improving Space Efficiency

Apart from time complexity, we also provide a method to reduce the space complexity which is still based on the factorization-based scoring function $\hat{f}_r(h, t)$ in Eq.(4). The models that will be studied in Section 4 satisfy this form or could be extended with some simple transformations.

First, we use two $|E| \times d$ matrices, $H_e$ and $T_e$, to store the embedding vectors of all head entities and tail entities, respectively. Similarly, we store the embedding vectors of all relations in the matrix $R_e$ with the size of $|R| \times d$. According to Eq.(6), the calculation of $L_H$, $L_R$ and $L_T$ are independent from each other with the calculation of each term only relies on the corresponding $H_e$, $R_e$ or $T_e$ matrix. For example, given the index $i$ and $j$, the value of the $L_H$ term is equal to the inner product of the $i$-th column vector and the $j$-th column vector of matrix $H_e$. As a result, we can construct three intermediate matrices denoted as $M_H$, $M_R$, and $M_T$ with the size of $d \times d$ to record the intermediate results for each term. The details are displayed in Eq.(7).

\[
M_H = H_e^T H_e, \quad M_R = R_e^T R_e, \quad M_T = T_e^T T_e
\]

(7)
Note that $M_I[i][j]$ is equal to the inner product of the $i$-th and the $j$-th column vector of $H_e$, similar for $M_R[i][j]$ and $M_F[i][j]$. Based on this, the calculation of the $L^A$ term in Eq.(6) can be simplified as:

$$L^A = c^* \sum(M_I \circ M_R \circ M_F)$$ \hspace{1cm} (8)

where $\circ$ means element-wise product of matrices, and sum means adding up all elements of a matrix. In this way, we can calculate $L^A$ in the space complexity of $O(d \times (|R| + |E| + d))$, so that we do not need to use any batch optimization for standard knowledge graph benchmarks such as the FB15K237 and WN18RR datasets (to be introduced in Section 5), since we can directly use the whole training data to calculate the loss function within reasonable time and space complexity.

As will be shown in the following section, for different models we may need to construct the $M_I$, $M_R$ and $M_F$ matrices in different ways, but this does not increase the space complexity. Besides, for extremely large datasets that cannot be loaded into memory as a whole, our framework with smaller space complexity can use fewer batches to train the model, which results in less training epochs.

4 APPLY NS-KGE ON DIFFERENT MODELS

In this section, we will show how our NS-KGE framework can be applied over different KGE models. As mentioned before, we will take the Bilinear-Diagonal embedding model (DistMult) [8], the Simple enhancement of Canonical Polyadic model (SimplE) [9], the Complex Embedding model (ComplEx) [10] and the Translational Embedding model (TransE) [5] as examples.

4.1 Bilinear-Diagonal Embedding (DistMult)

DistMult is a representative factorization-based multi-relation representation learning model [8]. It learns each entity as a vector embedding, and learns each relation as a diagonal matrix. For a triplet $(h, r, t)$, DistMult trains the model based on the following scoring function:

$$\hat{f}_r(h, t) = e_h^T \cdot diag(r) \cdot e_t = \sum_{i} e_{h,i}r_i e_{t,i}$$ \hspace{1cm} (9)

We can see that the scoring function of DistMult is the same as our framework (Eq.(4)). As a result, we can directly apply the loss function of the NS-KGE framework (Eq.(6)) for model learning, and use Eq.(8) for better space complexity. The final time complexity is $O(d^2(|E| + |R| + |E|))$. In the experiments, we will show that our Non-sampling DistMult is both more efficient and more effective that the original sampling-based DistMult model.

4.2 Simple Enhancement of CP (SimplE)

Canonical Polyadic (CP) decomposition is one of the earliest work on tensor factorization approaches [42]. Since CP learns two independent embedding vectors for each entity, it performs poorly in KG link prediction tasks. The Simple [9] embedding method provides a simple improvement of CP by learning the two embeddings of each entity dependently, which gains much better performance on link prediction.

For SimplE, its scoring function is a little bit more complex than DistMult. The scoring function is a combination of two parts:

$$\hat{f}_r(h, t) = \frac{1}{2} e_h^T (r \circ e_t) + e_t^T (r^{-1} \circ e_h) \hspace{1cm} (10)$$

To apply the NS-KGE framework on SimplE, we need to consider $\hat{f}_r(h, t)^2$, which consists of three terms:

$$\hat{f}_r(h, t)^2 = \frac{1}{4} \sum_{i,j} (e_{h, i} e_{h, j})(r_i r_j)(e_{t, i} e_{t, j})$$

$$+ 2 \sum_{i} \sum_{j} (e_{h, i} e_{h, j})(r_i r_j^{-1})(e_{t, i} e_{t, j})$$

$$+ \sum_{i} \sum_{j} (e_{h, i} e_{h, j})(r_i^{-1} r_j^{-1})(e_{t, i} e_{t, j})$$ \hspace{1cm} (11)

We can see that for each term in Eq.(11), its structure is the same as that in Eq.(5). As a result, the rest of the work is similar to what we did in Eq.(6). The only difference is that we result in three $L^A$ terms in Eq.(6), but the time and space complexity remain unchanged.

4.3 Complex Embeddings (ComplEx)

The ComplEx embedding model learns KG embeddings in a complex number space [10]. It adopts the Hermitian dot product to construct the scoring function. But we can still do similar rearrangements for non-sampling knowledge graph embedding. The scoring function of ComplEx is:

$$\hat{f}_r(h, t) = h_{re}^T(r_{re} \circ t_{re}) + h_{im}^T(r_{im} \circ t_{im})$$

$$+ h_{re}^T(r_{im} \circ t_{im}) - h_{im}^T(r_{im} \circ t_{re})$$ \hspace{1cm} (12)

where $h_{re}, r_{re}, t_{re}$ are the real part of the head, relation, and tail embedding vectors, while $h_{im}, r_{im}, t_{im}$ are the imaginary part of the head, relation, and tail embedding vectors. There will be six terms for $\hat{f}_r(h, t)^2$. In the following, we would only show the expansion result of the first term $L^A = h_{re}^T(r_{re} \circ t_{re}) \cdot h_{im}^T(r_{im} \circ t_{im})$, since the other terms look similar.

$$L^A = c^* \sum_{r \in R} \sum_{h \in E} \sum_{t \in E} \left( \sum_{i,j} h_{re,i} r_{re,j} t_{im,j} \right) \left( \sum_{i,j} h_{im,i} r_{im,j} t_{re,j} \right)$$

$$= c^* \sum_{i,j} \sum_{h \in E} \sum_{t \in E} \left( h_{re,i} h_{im,j} \right) \left( r_{re,i} t_{im,j} \right) \left( r_{im,i} t_{re,j} \right)$$ \hspace{1cm} (13)

In this way, we can calculate the loss of non-sampling ComplEx within $O(d^2(|R| + |E|))$ time complexity and $O(d(|R| + |E| + d))$ space complexity.

4.4 Translational Embedding (TransE)

Unlike the previous factorization-based knowledge graph embedding models, TransE [5] is a translation-based embedding model. As a result, the NS-KGE framework cannot be directly applied to TransE. However, we will show that by applying very minor modifications to the scoring function without changing the fundamental idea of translational embedding, the NS-KGE framework can still be applied to TransE.


| Dataset      | #entities | #relations | #train | #test  |
|--------------|-----------|------------|--------|--------|
| FB15K237    | 14,541    | 237        | 272,115| 20,466 |
| WN18RR      | 49,943    | 11         | 86,835 | 3,134  |

Table 2: Basic statistics of the datasets

The original scoring function of TransE is \( f_E(h, t) = \|h + r - t\|^2 \), which calculates the distance between \( h + r \) and \( t \). For positive examples, we hope the distance would be as small as possible, which could be 0 in the most ideal case. For negative examples, we hope the distance would be as far away as possible. In many implementations of TransE, to avoid over-fitting, we usually apply unit-vector constraints to the embedding vectors, i.e., \( h, r \) and \( t \) are regularized as length-one vectors. In this case, the maximum possible distance between \( h + r \) and \( t \) would be 3, as a result, the optimal value for negative examples would be 3. However, the mathematical derivation of our framework in Eq.(3) relies on the assumption that the ground-truth value for positive instances is 1 and for negative instances it is 0. As a result, we slightly modify the scoring function of TransE to a new function \( f_E'(h, t) = 1 - \frac{1}{3} f_E(h, t) \). In this way, the ground-truth value satisfy our assumption while it does not influence the optimization of TransE. The only difference is that instead of minimizing \( f_E(h, t) \) for positive examples in TransE, we aim to maximize \( f_E'(h, t) \), which is basically the same in terms of optimization.

By inserting \( f_E'(h, t) \) into Eq.(3), we have:

\[
L = L^P + \sum_{r \in R} \sum_{h \in E} \sum_{t \in E} c \left(1 - \frac{1}{3} f_E'(h, t)\right)^2
\]

\[
= L^P + \sum_{r \in R} \sum_{h \in E} \sum_{t \in E} c \left(\frac{2}{3} (h^T r + r^T (t - h))^2\right)
\]

\[
= L^P + \sum_{r \in R} \sum_{h \in E} \sum_{t \in E} \frac{4c}{9} \left((h^T r)^2 + (r^T t)^2 + (r^T h)^2 - 2r^T tr^T h\right)
\]

\[
+ \sum_{r \in R} \sum_{h \in E} \sum_{t \in E} \left(\sum_{i \in E} \sum_{j \in E} |E| \sum_{r \in R} \sum_{t \in E} h_i h_j r_i t_j - 2 \sum_{i \in E} \sum_{j \in E} |E| \sum_{h \in E} h_i h_j \sum_{r \in R} r_i t_j\right)
\]

\[
= L^P + \frac{4c}{9} \sum_{i \in E} \sum_{j \in E} |E| \sum_{r \in R} \sum_{t \in E} r_i t_j + \sum_{i \in E} \sum_{j \in E} |E| \sum_{h \in E} h_i h_j \sum_{r \in R} r_i t_j
\]

\[
+ \sum_{r \in R} \sum_{h \in E} \sum_{t \in E} \left(\sum_{i \in E} \sum_{j \in E} |E| \sum_{r \in R} \sum_{t \in E} r_i t_j + \sum_{i \in E} \sum_{j \in E} |E| \sum_{h \in E} h_i h_j \sum_{r \in R} r_i t_j\right)
\]

\[
= \frac{1}{L_H} \sum_{r \in R} \sum_{h \in E} \sum_{t \in E} l_H + \frac{1}{L_T} \sum_{r \in R} \sum_{h \in E} \sum_{t \in E} l_T
\]

We can see that similar to Eq.(6), the final loss can also be decomposed to the \( L_H \), \( L_R \) and \( L_T \) terms, which are independent from each other for better time complexity, and can be calculated with better space complexity (Section 3.4). The \( S_H \) and \( S_T \) terms are just summation of the entity embedding matrix, whose calculation is even easier than the \( L_H \) and \( L_T \) terms. The final time complexity of non-sampling TransE is \( O(d^2(|R| + |E|)) \).

5 EXPERIMENTS

In this section, we conduct experiments to evaluate both the efficiency and effectiveness of the NS-KGE framework.1

5.1 Experimental Setup

5.1.1 Dataset Description. We conduct the experiments on two benchmark datasets for knowledge graph embedding research, namely, FB15K237 and WN18RR. The detailed statistics of the datasets are shown in Table 2, and we will briefly introduce these two datasets in the following.

FB15K237: One of the most frequently used dataset for KGE. The original version of the FB15K dataset is generated from a subset of the Freebase knowledge graph [46]. However, in the original FB15K dataset, a large number of the test triplets can be obtained by simply reversing the triplets in the training set, as shown in [47, 48]. For example, the test set may contain a triplet (home, car, work), while the training set contains a reverse triplet (work, car, home). The existence of such cases make the original dataset suffer from the test leakage problem. As a result, the FB15K237 [47] dataset is introduced by removing these reverse triplets, which mitigates the test leakage problem in a large extend. In this paper we use FB15K237 for evaluation.

WN18RR: WN18 is also a standard dataset for KGE. The original WN18 dataset is a subset of WordNet [49], which is an English lexical database. Similar to FB15K, WN18 is also corrected to WN18RR [50] by removing the reverse triplets to avoid test leakage. In this work, we use WN18RR for evaluation.

We use the default train-test split of the original datasets. Both the training set and the testing set are a set of \((h, r, t)\) triplets. The number of training and testing triplets of the two datasets are shown in Table 2.

5.1.2 Baselines. We study the performance of the NS-KGE framework by comparing the performance of a KGE model with or without using the framework. Similar to what we have introduced before, we consider the following KGE models.

- DistMult [8]: The bilinear-diagonal embedding model, which uses diagonal matrix to represent the relation between head and tail entities.
- SimpleE [9]: The model is a simple enhancement of the Canonical Polyadic (CP) decomposition model [42] by learning two dependent embeddings for each entity.
- ComplEx [10]: The model learns KG embedding in a complex number space, which uses complex number vectors to represent the entities and relations.
- TransE [5]: The translational embedding model, which minimizes the distance between head and tail entities after translation by the relation vector.

All of the models are implemented by PyTorch, an open source library. And we use the baselines implemented by OpenKE2 [51], an open source tool-kit for knowledge graph embedding.

5.1.3 Evaluation Metrics. For each triplet \((h, r, t)\) in the testing set, we first use \( h \) and \( r \) to rank all tail entities, and evaluate the position of the correct tail entity \( t \). And then we use \( t \) and \( r \) to rank all head entities, and evaluate the position of the correct head entity \( h \). As a result, if there are \(|S|\) triplets in the testing set, we will conduct \(2|S|\) evaluations.

1Source code available at https://github.com/ruetersielab/NS-KGE

2https://github.com/thunlp/OpenKE
We use Hit Ratio (HR), Mean Rank (MR) and Mean Reciprocal Rank (MRR) to evaluate the models. HR is used to measure whether the correct entity is in the Top-K list. MR is the mean of the correct entity’s rank, defined as $MR = \frac{1}{|D|} \sum_{(h,r,t) \in D} \text{rank}_t$. MRR is defined as $MRR = \frac{1}{|D|} \sum_{(h,r,t) \in D} \frac{1}{\text{rank}_t + \text{rank}_c}$. These three metrics are widely used in KG embedding evaluation [5, 9, 41]. For HR and MRR, larger value means better performance, and for MR, smaller value means better performance.

5.1.4 Parameter Settings. We set the default embedding dimension as 200, the number of training epochs as 2000, initial learning rate as 0.0001, and use Adam optimizer [52] for all models. To avoid over-fitting, we apply $L_2$ normalization over the parameters for all models, and we conduct grid search to find the best coefficient of regularization for each model in $[10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}]$. We also conduct grid search to find the best learning rate decay for each model in $[0.1, 0.3, 0.5, 0.7]$. For negative sampling-based models, we set the number of negative samples as 25; the batch size is 4000. For non-sampling models, we do not split training data into batches, because our model has lower space complexity; the coefficient of positive instances $c^+$ is set to 1, and the coefficient of negative instances $c^-$ is grid searched in $[10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}]$. The default setting of $c^-$ is 0.001 in all experiments except when we are tuning $c^-$ to see its influence. For each model on each dataset, we run the model 5 times and report the average result of the 5 times. We use paired $t$-test to verify the significance of the results.

5.2 Performance Comparison

We apply NS-KGE to DistMult, SimplE, ComplEx and TransE. The experimental results on prediction accuracy are shown in Table 3, and more intuitive comparison are shown in Figure 1. We have the following observations from the results.

First and most importantly, compared to the four baselines, in most cases, our NS-KGE framework achieves the best performance on both of the two datasets. Although some baselines are slightly better than NS-KGE in some cases, for example, on the WN18RR dataset, SimplE’s HR@10 has a slightly better performance, but we can see the results are comparable. For HR@1 and HR@3, NS-KGE has 9.78% and 49.01% improvement on average, respectively.

We also conducted some qualitative analysis of the entity ranking results, as shown in Table 4. First, for the same entity and relation, we see that the correct prediction gains higher rank in our non-sampling models. Second, compared to the sampling-based model, the top-10 ranked entities by our non-sampling model tend to be more intuitive and more relevant to the given entity and relation.

The reason why applying the NS-KGE framework can improve the performance of sampling-based methods (DistMult, SimplE, Complex and TransE) is that, the sampling-based methods only use part of the negative instance information in the dataset, and these models may ignore some important negative instances. However, our NS-KGE framework makes it possible to use all of the information in the dataset and brings better computational time and space consumption at the same time (to be discussed in the following experiments). Therefore, NS-KGE can avoid the problem of sampling-based methods and thus improve the performance.

One interesting observation is that on the WN18RR dataset, our NS-KGE framework is consistently better on the MRR measure, but is worse on the MR measure. The difference between MRR and MR is that MRR is more sensitive to bad cases. Due to the large number of entities in the WN18RR dataset, if the correct entity is ranked to lower positions in some lists, it will have a huge influence on the MR measure, but not too much on the MRR measure due to the reciprocal operation. The result implies that our framework can rank the correct entity to very low positions in some cases. However, since our performance on MRR is better, it means that in most cases our framework ranks the correct entity to top positions. Moreover, our NS-KGE framework is more intuitive and more relevant to the given entity and relation.

Besides, the observation that NS-KGE improves the performance of both factorization-based (DistMult, SimplE, ComplEx) models and translation-based (TransE) models indicates the effectiveness of the NS-KGE framework, and also shows the potential of applying the framework on other KG embedding models.

5.3 Analyses of Hyper Parameters

In this section, we will analyze the impact of different dimension size $d$ and the negative instance weight $c^-$. The impact of dimension size. Figure 2 shows the performance of our NS-KGE framework under different choices of embedding dimension size $d$. We can see that in most cases, the performance becomes better when the embedding dimension size increases. It
| Dataset | FB15K237 |
|---------|-----------|
| Relation | media common/netflix genre/titles |

| Entity | The Notebook | Funny Girl |
|--------|--------------|------------|

| Model       | NS-DistMult | DistMult | NS-Simple | Simple |
|-------------|-------------|----------|-----------|--------|
| Predicted Top-10 Entities | | | |
| Entities | | | | |
| comedy | historical period drama | comedy | historical period drama | United States of America |
| drama | thriller | drama | thriller | drama |
| fantasy | romance film | musical film | musical film | psychological thriller |
| biography | musical film | fantasy | biography | thriller |
| thriller | suspense | suspense | political drama | United Kingdom |
| psychological thriller | | | psychological thriller | |
| suspense | | | thriller | |

**Table 4:** Qualitative results on entity ranking. NS-X means the non-sampling version of model X under our NS-KGE framework. Bold entities are the ground truth. Common entities between model X and NS-X are in gray to highlight the difference entities.

Figure 1: Performance on HR@10 and MRR with and without the NS-KGE framework. The models on x-axis labeled with * mean that the performance of NS-KGE framework improved more than 20% from the corresponding baselines.

Figure 2: Performance on HR@10 (left two figures) and MRR (right two figures) under different dimension size $d$.

Figure 3: Performance on HR@10 (left two figures) and MRR (right two figures) under different negative instance weight $c^−$.

As we have shown in Section 3.3, the training time is proportional to the square of dimension size. Therefore, we need to trade-off between the training time and the performance. As we can see in Figure 2, in most cases, the performance tends to be stable at round
200 dimensions. As a result, we choose 200 as the default dimension size for all of the models.

**Impact of negative instance weight.** In this experiment, we fix the positive instance weight in the NS-KGE framework as $c^+ = 1$, and we tune the negative instance weight $c^-$ to analyze its influence. Figure 3 shows the results when we change the negative instance weight $c^-$ on the two datasets. We see that for all models on both datasets, when the value of $c^-$ increases, the performance tends to increase first and then decreases when $c^-$ is too large. This shows that a proper selection of $c^-$ value is important to the model performance. If $c^-$ is too small (e.g., close to 0), the model would not be able to leverage the information included in the negative instances of the KG. However, the information in negative instances is also noisy, e.g., if two entities are not connected, this may not directly indicate they are irrelevant, instead, this may be caused by the noise in data collection process. As a result, negative instance information is not as reliable as positive instances, and if $c^-$ is too large, it may decrease the performance. Because negative samples are usually much more than positive samples, to avoid class imbalance, the weight of negative instances $c^-$ should be smaller than $c^+$. In most cases, the optimal selection of $c^-$ is 0.001.

### 5.4 Efficiency Analyses

In this section, we will discuss the training efficiency of our NS-KGE framework. We will compare the training time of the four sampling-based models DistMult, SimpLe, Complex, TransE and their non-sampling versions. For fairness of comparison, all experiments run on a single NVIDIA GeForce 2080Ti GPU. The operating system is Ubuntu 16.04 LTS. For all models, the embedding dimension is set as 200 and number of training epochs is 2000. Results on model training time is shown in Table 5.

| Model       | FB15K237 Speed-up | WN18RR Speed-up |
|-------------|--------------------|-----------------|
| DistMult    | 3546s 1.00         | 1922s 1.00      |
| NS-DistMult | 53s    66.91       | 57s   33.72     |
| SimpLe      | 4447s 1.00         | 2450s 1.00      |
| NS-SimpLe   | 73s    60.92       | 77s   31.82     |
| TransE      | 2353s 1.00         | 673s  1.00      |
| NS-TransE   | 107s   21.99       | 86s   7.83      |
| CompEx      | 6736s 1.00         | 3346s 1.00      |
| NS-CompEx   | 157s   42.90       | 158s  21.18     |

**Table 5: Experimental results on model training time.** The models are ordered from top to bottom in ascending order of the training time on each dataset. Speed-up shows how many times NS-X is faster than the corresponding model X.

We can see that the training efficiency of our NS-KGE framework is significantly better than the baseline models. For example, if we apply NS-KGE to the DistMult model on the FB15K237 dataset, it only takes 53s to finish training the model, while the original sampling-based DistMult model takes 3546s. The acceleration is about 70 times. For other models and datasets, we also get 20 ~ 60 times acceleration. This is not surprising because for the sampling-based KG models, a lot of computational time needs to be spent on sampling the negative examples, while our framework eliminates the sampling procedure. In our implementation, we used in-memory sampling instead of on-disk sampling for the baselines, however, our NS-KGE framework is still much faster than the baselines.

Another interesting observation from Table 5 is that on each dataset, the computational time of our NS-KGE models are NS-DistMult < NS-SimpLe < NS-TransE < NS-CompEx. This is consistent with the mathematical analysis in Section 3 and Section 4. For the NS-DistMult model, its final loss function has one $L^A$ term (see Eq.(6)). For the NS-SimpLe model, its final loss function has three $L^A$ terms (Eq.(11)). For the NS-TransE model, its final loss function has four $L^A$ terms (Eq.(14)). While for the NS-CompEx model, its final loss function has six $L^A$ terms (see Section 4.3, we only show one of terms in Eq.(13)). As we have shown in Section 3.3, the $L^A$ term(s) take the most significant computational time in the loss function. As a result, the final model training time is proportional to the number of $L^A$ terms in the loss function.

### 6 CONCLUSIONS AND FUTURE WORK

In this paper, we propose NS-KGE, a non-sampling framework for knowledge graph embedding, which leverages all of the positive and negative instances in the KG for model training. Besides, we provided mathematical methods to reduce the time and space complexity of the framework, and have shown how the framework can be applied to various KGE models. Experiments on two benchmark datasets demonstrate that the framework is able to enhance both the model performance and the training efficiency.

In the future, we will consider applying our framework on more complex KGE models such as neural network-based models, as well as more complex graph structures such as attributed graphs. We also plan to apply our framework to other graph computation models beyond KGE, such as graph neural networks.

**ACKNOWLEDGEMENT**

We appreciate the reviews and suggestions of the reviewers. This work was supported in part by NSF IIS-1910154 and IIS-2007907. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors.

**REFERENCES**

[1] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 1247–1250, 2008.

[2] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In Proceedings of the 16th international conference on World Wide Web, pages 697–706, 2007.

[3] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Fabio N Mendes, Sebastian Hellmann, Mohamed Mosty, Patrick Van Kiel, Sören Auer, et al. Dbpedia—a large-scale, multilingual knowledge base extracted from wikipedia. Semantic web, 6(2):167–195, 2015.

[4] Shaosong Ji, Shurui Pan, Erik Cambria, Pekka Marttinen, and Philip S Yu. A survey on knowledge graphs: Representation, acquisition and applications. arXiv preprint arXiv:2002.00388, 2020.

[5] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Ok-sana Yakhnenko. Translating embeddings for modeling multi-relational data. In Neural Information Processing Systems (NIPS), pages 1–9, 2013.

[6] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey on knowledge graphs: Representation, acquisition and applications. IEEE Transactions on Knowledge and Data Engineering, 2017.

[7] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. arXiv preprint arXiv:1205.2618, 2012.
[8] Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In Proceedings of the International Conference on Learning Representations (ICLR), May 2015.

[9] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In Advances in neural information processing systems, pages 4284–4295, 2018.

[10] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Gaël Guillaume Bouchard. Complex embeddings for simple link prediction. In International Conference on Machine Learning, pages 2071–2080. PMLR, 2016.

[11] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In IJML, volume 11, pages 809–816, 2011.

[12] Menghan Wang, Mingming Gong, Xiaolin Zheng, and Kun Zhang. Modeling dynamic missingness of implicit feedback for recommendation. In Advances in neural information processing systems, pages 6669–6678, 2018.

[13] Chong Chen, Min Zhang, Chenyang Wang, Weizhi Ma, Minming Li, Yiqun Liu, and Shaoping Ma. An efficient adaptive transfer neural network for social-aware recommendation. In Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 225–234, 2019.

[14] Yongqi Zhang, Quanming Yao, Xinying Shao, and Lei Chen. Ncscaching: Simple and efficient negative sampling for knowledge graph embedding. In 2019 IEEE 35th International Conference on Data Engineering (ICDE), pages 614–625. IEEE, 2019.

[15] Liwei Cai and William Yang Wang. KBgan: Adversarial learning for knowledge graph based recommendation. In Proceedings of the 21th ACM International Conference on Web Search and Data Mining, pages 505–513, 2018.

[16] Steffen Rendle. Factorization machines. In Advances in neural information processing systems, pages 1065–1073, 2010.

[17] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. IEEE Transactions on Knowledge and Data Engineering, 29(12):2724–2743, 2017.

[18] Chenyan Xiong, Russell Power, and Jamie Callan. Explicit semantic ranking for text-centric information retrieval. In The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, pages 1387–1390, 2018.

[19] Tingting Xie, Yaxin Zhu, Wenchao Zhang, and Jianyu Wang. Explainable recommendation over knowledge graphs. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 69–78, 2020.

[20] Pengfei Wang, Hanxiong Chen, Yadong Zhu, Huaweien Shen, and Yongfeng Zhang. Unified collaborative filtering over graph embeddings. In Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 155–164, 2019.

[21] Xin Wang, Jingqian Zhang, Bingcheng Li, and Ping Li. Knowledge graph embedding based question answering. In Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, pages 105–113, 2019.

[22] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping. arXiv preprint arXiv:1808.05658, 2018.

[23] Wenhao Xiong, Zhiwen Huang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, 2017.

[24] Shuai Geng, Peng Gao, Moisoya Chatterjee, Chiori Hori, Jonathan Le Roux, Yongfeng Zhang, Hongsheng Li, and Anoop Cherian. Dynamic graph representation learning for video dialog via multi-modal shuffled transformers. In AAAI, 2021.

[25] Zhiyuan Liu, Zhe Gan, and Junzhou Huang. Neural relational embeddings. In Proceedings of the 31st Conference on Neural Information Processing Systems, pages 3919–3927, 2017.

[26] Xiao Hu, Shenzhi Tian, and Yuanchen Zhou. Fairness-aware recommendation. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 19–26, 2020.

[27] Chong Chen, Min Zhang, Yongfeng Zhang, Yiqun Liu, and Shaoping Ma. Efficient heterogeneous collaborative filtering without negative sampling for recommendation. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 19–26, 2020.

[28] Chong Chen, Min Zhang, Yongfeng Zhang, Yiqun Liu, and Shaoping Ma. Efficient non-sampling factorization machines for optimal context-aware recommendation. In Proceedings of the Web Conference 2020, pages 2400–2410, 2020.

[29] Chong Chen, Min Zhang, Chong Zhang, Weizhi Ma, Yiqun Liu, and Shaoping Ma. Efficient non-sampling factorization machines for optimal context-aware recommendation. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 19–26, 2020.

[30] Chong Chen, Min Zhang, Yongfeng Zhang, Yiqun Liu, and Shaoping Ma. Efficient neural matrix factorization without sampling for recommendation. ACM Transactions on Information Systems (TOIS), 38(2):1–28, 2020.

[31] Chong Chen, Min Zhang, Weizhi Ma, Yiqun Liu, and Shaoping Ma. Jointly non-sampling learning for knowledge graph enhanced recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 189–198, 2020.

[32] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. IEEE Transactions on Knowledge and Data Engineering, 29(12):2724–2743, 2017.

[33] Chenyan Xiong, Russell Power, and Jamie Callan. Explicit semantic ranking for academic search via knowledge graph embedding. In Proceedings of the 26th international conference on world wide web, pages 1271–1279, 2017.

[34] Laura Dietz, Alexander Kotov, and Edgar Meij. Utilizing knowledge graphs for text-centric information retrieval. In The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, pages 1387–1390, 2018.

[35] Qingsuo Ai, Xiaorong Chen, and Yiming Yang. Knowledge graph based semantic fusion. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pages 1006–1014, 2020.

[36] Zhiyu Jiang, Xuejie Luo, Xiaoyao Wang, and Yongfeng Zhang. Neural relational embeddings for knowledge graph based recommendation. In Proceedings of the 33rd ACM Conference on Information and Knowledge Management, page 614–625. IEEE, 2019.

[37] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping. arXiv preprint arXiv:1808.05658, 2018.

[38] Kun Zhou, Wayne Xin Zhao, Shaqing Bian, Yuanhang Zhou, Ji-Rong Wen, and Jingsong Yu. Improving conversational recommender systems via knowledge graph based semantic fusion. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 1006–1014, 2020.

[39] Frank I. Hitchcock. The expression of a tensor or a polyadic as a sum of products. In Foundations and Trends® in Machine Learning, volume 9, issue 2, pages 105–113, 2016.

[40] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and卵巢. Translating embeddings for modeling multi-relational data. In Proceedings of the 12th European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, pages 1062–1071, 2012.

[41] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d’Amato, Gerard de Melo, Felix Fischer, et al. Knowledge graphs. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 29, No. 1, 2015.

[42] Frank I. Hitchcock. The expression of a tensor or a polyadic as a sum of products. In Foundations and Trends® in Machine Learning, volume 9, issue 2, pages 105–113, 2016.

[43] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality, pages 57–66, 2015.

[44] Farahznak Akrami, Mohammad Samiul Saeed, Qingheng Zhang, Wei Hu, and Chengkai Li. Realistic re-evaluation of knowledge graph completion methods: An experimental study. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pages 1995–2010, 2020.

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

[46] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. AAAI, 2018.

[47] Xu Han, Shulin Cao, Lv Xin, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. Openbook: An open toolkit for knowledge embedding. In Proceedings of EAMTL, 2018.

[48] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. ICLR, 2015.