WHY DOES NEGATIVE SAMPLING NOT WORK WELL? ANALYSIS OF CONVEXITY IN NEGATIVE SAMPLING

Anonymous authors
Paper under double-blind review

ABSTRACT

A negative sampling (NS) loss function is widely used in various tasks because we can choose an appropriate noise distribution considering properties for a targeting task. In particular, since the NS loss function does not have a normalization term, it is useful for classification problems with a large number of labels to be considered, such as knowledge graph embedding in terms of computational efficiency. On the other hand, properties of the NS loss function that are considered important for learning, such as the relationship between the noise distribution and the number of negative samples, have not been investigated theoretically. By analyzing the gradient of the NS loss function, we show that the NS loss function is non-convex and has a partial convex domain. We investigated the conditions of noise distribution and the number of samples required for efficient learning under this property. As a result, we found that the NS loss function behaves as a convex loss function when our induced conditions are satisfied and combined with a scoring method that handles only non-negative values, which enables efficient learning. Experimental results in FB15k-237, WN18RR, and YAGO3-10 showed that NS loss satisfying the conditions we proposed can improve the performance of KG completion by utilizing TransE and RotatE, which are non-negative scoring methods.

1 INTRODUCTION

A negative sampling (NS) loss function (Mikolov et al., 2013) has been used in various tasks due to its computational efficiency. Typical cases are learning of word vectors (Mikolov et al., 2013; Bojanowski et al., 2017) and pretrained language models (Clark et al., 2020b;a) in the natural language processing field, but in recent years, it has been used especially in knowledge graph embedding (Trouillon et al., 2016). A Knowledge Graph (KG) is a graph that describes the relationships between entities by using the entities as nodes and the edges linking the nodes as relations between entities. Since KGs explicitly represent knowledge, they are widely used to solve tasks requiring prior knowledge, especially in natural language processing. Knowledge graph embedding (KGE) (Bordes et al., 2011) is a method to complement this knowledge graph by predicting the edges that are missing between entities. Learning scoring for a knowledge graph describing entities and their relations is equivalent to solving a multiclass classification problem over all possible edges. This means that to learn the combination of entities and relations in the knowledge graph, we need a loss function that can handle a large number of labels. Unlike the softmax cross-entropy (SCE) loss function, the NS loss function does not have a normalization term. The difference makes NS loss an efficient learning method for embedding knowledge graphs considering a large number of combinations of entities and relations in them.

On the other hand, the current evaluation in KGE is a simplified problem where, given an entity and a relation, the model just predicts the corresponding entity. In this setting, the loss function can train the model sufficiently if it can only handle entities as classes. For this reason, not only the NS loss but also the SCE loss function is widely used in the current KGE as well. Thus, there is a situation where we can select from multiple loss functions in the current KGE. In such a situation, it is very important to know which loss function contributes to performance improvement. However, the effectiveness of the loss function has not been studied for a long time in KGE. This is because the performance of the trained model depends not only on the loss function but also on the scoring method.

In recent years, several studies, mainly in KGE, have investigated how the NS loss and SCE loss work well with different scoring methods. Ruffinelli et al. (2020); Ali et al. (2020) present a combination of the loss function and score function that improves performance by performing large-scale hyperparameter tuning against the combination of the scoring method, loss function, and other factors. The combinations they searched empirically show that choosing an appropriate loss function depends on the selected score function. Kamigaito & Hayashi (2021) investigate the relationship between the NS loss and SCE loss functions from a theoretical aspect. They showed that excluding the case of scoring methods that handle only non-negative values such as TransE (Wang et al., 2014) and RotatE (Sun et al., 2019), the SCE loss performs better than the NS loss.

Given the results obtained by the above studies, the current conclusion seems to be that the SCE loss may not be suitable for some scoring methods, and in such cases, we should use the NS loss. Therefore, even if the SCE loss is available for a task, the NS loss is still substantial. On the other hand, even though the NS loss function is important, it has not been theoretically analyzed how its basic hyper-parameters that are noise distribution and the number of samples affect the model during training.

To solve this problem, we focused on the derivatives of the NS loss function and analyzed its gradient. As a result of this theoretical analysis, we showed that the NS loss function is non-convex and also showed that the NS loss function has a convex region. Moreover, we also show that the inflection point that separates the convex region from the non-convex region is determined by the noise distribution and the number of samples. These results indicate that NS loss requires appropriate settings of the noise distribution, the number of samples, the initial value, and the learning rate in order to train the model in convex regions. As a result, we found that the condition is satisfied by using only non-negative values in the scoring method. We also showed that even if this condition is satisfied, there may be cases of NS loss where the model cannot be fitted to the training data. These theoretical results are not limited to KGE, but are applicable to general tasks.

In addition, we verified conditions for eliminating the existence of training cases that are difficult to fit, limited to the use in KGE. As a result, we clarified how to handle the noise distribution and the number of samples to satisfy this condition. Then, we derived a new variant of NS loss function that satisfies the verified conditions. We named the newly derived NS loss function Self-Smoothing Negative Sampling (SSNS) because it performs smoothing using the model's predictions under training. We also theoretically compared SSNS with Self-Adversarial Negative Sampling (SANS) (Sun et al., 2019), one of the variants of NS loss, to confirm the effectiveness of SSNS on datasets with a variety of relations other than 1-to-1 relations.

Experimental results using TransE and RotatE, which are scoring methods that output only non-negative values, showed that our proposed SSNS can improve the performance of KG completion against SANS in FB15k-237 (Toutanova & Chen, 2015), WN18RR (Dettmers et al., 2018c), and YAGO3-10 (Dettmers et al., 2018b). In addition, we observed a significant performance improvement of SSNS on YAGO3-10, which is a dataset that contains a variety of relations other than 1-to-1 relations. This result indicates that our theoretical explanation is also observed when training with the actual models and datasets.

Our contributions in this paper are as follows:

- After showing that the NS loss function is non-convex, we derive the conditions for the NS loss function to behave as a convex function.
- We show that the noise distribution, the number of samples, the initial value, and the learning rate must be set appropriately in order to train a model on a convex region in the NS loss function.
- We show that the NS loss function behaves as a convex function when we use a scoring method that outputs only non-negative values.
- We show that the existence of training examples that cannot be fit by the NS loss when using a scoring method that outputs only non-negative values.
- In the case of KGE, we show the conditions to deal with the unfit training examples in the NS loss.
- We derive a new variant of the NS loss, self-smoothing negative-sampling (SSNS), which can deal with the unfit training examples in KGE.

- Through experiments, we have confirmed that we can actually observe the theoretical advantages of SSNS in training on real datasets and models.

We organized this paper as follows: Sec. 2 briefly explains the NS loss function through derivation from Bregman divergence; Sec. 3 shows the gradient of the NS loss function to investigate its convexity; Sec. 4 derives the conditions for the NS loss function to behave as a convex function and derives a new NS loss function, SSNS, that satisfies these conditions; Sec. 5 examines whether we can observe the theoretical properties of the SSNS in the real datasets and models; Sec. 6 introduces related studies; Sec. 7 summarizes our conclusions on the convexity of NS loss obtained through this work.

2 Understanding Negative Sampling through Bregman Divergence

In this paper, in order to clarify the characteristics of the NS loss function, we follow the previous study (Kamigaito & Hayashi, 2021) and understand the NS loss function from the viewpoint of Bregman divergence. Through this understanding, we can clarify the problems that exist in the current NS loss function.

First, we explain the definition of Bregman divergence (Bregman, 1967). Let $\Psi(z)$ be a differentiable strictly convex function in its domain[1]. Bregman divergence between points f and g is defined as follows[2]:

$$d_{\Psi(z)}(f,g) = \Psi(f) - \Psi(g) - \nabla_g \Psi(g)^T (f-g). \tag{1}$$

By changing $\Psi(z)$, Bregman divergence can represent various distances. In the previous study (Kamigaito & Hayashi, 2021), they used the study on Noise Contrastive Estimation (NCE) (Gutmann & Hirayama, 2011) to derive each loss function. We denote a pair of an input x and its label y as (x,y). In order to minimize the distance across the entire observed data $D = \{(x_1,y_1),\cdots,(x_n,y_n)\}$ which follows a distribution $P_d(x,y)$, we fix f as the point followed by the training data and eliminate the terms that do not contain g used for representing an output value of a model, and then, we can define the expectation of Eq. (1) as follows:

$$B_{\Psi(z)}(f,g) = \sum_{(x,y)\in D} \left[-\Psi(g) + \nabla_g \Psi(g)^T g - \nabla_g \Psi(g)^T f\right] p_d(x,y). \tag{2}$$

The reason for eliminating terms that do not contain g is that the eliminated terms do not affect the loss function during training. In Eq. (2), $f = g$ is satisfied when $B_{\Psi(z)}(f,g) = 0$. In the following explanations, we explain the relationship between the loss functions through the derivation of each loss function using Eq. (2).

The next step is to derive the NS loss function from Eq. (2) similar to the previous study (Kamigaito & Hayashi, 2021). The NS loss function updates the model parameter θ of a scoring function $s_\theta(x,y)$ to make $s_\theta(x,y)$ return a higher score for $(x,y) \in D$ than that of $(x,y) \notin D$. Letting ν be a number of negative samples, $p_n(y|x)$ be a noise distribution, $f = \frac{\nu p_n(y|x)}{p_d(y|x)}$, $g = \frac{1}{\exp(s_\theta(x,y))}$, $\Psi(z) = z\log(z) - (1+z)\log(1+z)$, σ be a sigmoid function, we can induce the NS loss function as follows:

$$B_{\Psi(z)}\left(\frac{\nu p_n(y|x)}{p_d(y|x)}, \frac{1}{\exp(s_\theta(x,y))}\right) = -\frac{1}{|D|} \sum_{(x,y)\in D} \Big[\log(\sigma(s_\theta(x,y))) + \nu \mathop{\mathbb{E}}_{y_i \sim p_n} \log(\sigma(-s_\theta(x,y)))\Big]. \tag{3}$$

For a detailed derivation of this equation, see Appendix A.1. The right-hand side of Eq. (3) is similar to the orginal NS loss function proposed by Mikolov et al. (2013). From Eq. (3), we can understand that $\frac{\nu p_n(y|x)}{p_d(y|x)} = \frac{1}{\exp(s_\theta(x,y))}$ is satisfied when $B_{\Psi(z)}(\frac{\nu p_n(y|x)}{p_d(y|x)}, \frac{1}{\exp(s_\theta(x,y))})$ becomes 0, and thus, $\exp(s_\theta(x,y)) = \frac{p_d(y|x)}{\nu p_n(y|x)}$ is also satisfied.

The above derivations allow us to understand the relationship between $\frac{p_d(y|x)}{\nu p_n(y|x)}$, to which $\frac{1}{\exp(s_\theta(x,y))}$ converges in training, the number of samples, noise distribution, and gradient in the NS loss function. We focus on these relationships in the following discussion.

[1] In this paper, we only use $\Psi(z)$ that satisfies the condition.

[2] Since we only considers the NS loss function, f and g are scalar values in this paper.

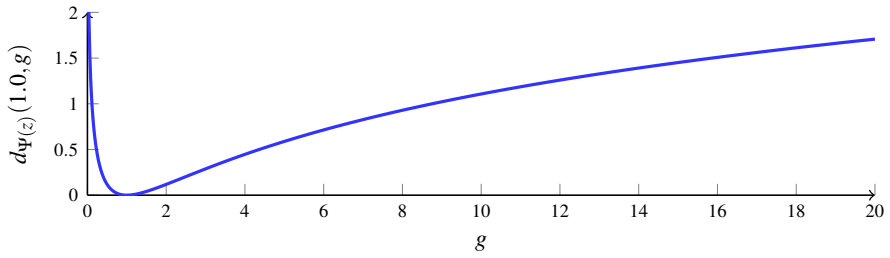


Figure 1: Divergence of the NS loss function of a given (x, y). This figure indicates the divergence between $f = 1.0$ and $g = \frac{1}{\exp(s_\theta(x,y))}$ in $d_{\Psi(z)}(f, g)$ when $\Psi(z) = z\log(z) - (1+z)\log(1+z)$.

3 Non Convexity of Negative Sampling

In this Section, we theoretically investigate how the NS loss affects the model and reveal its problem. $B_{\Psi(z)}(f, g)$ is the expected value when all (x, y) are considered in Eq. (2). Thus, for simplicity, we first investigate the influence of the NS loss on the model given each label pair (x, y). Therefore, in order to account for the NS loss in (x, y), we focus our on $d_{\Psi(z)}(f, g)$ in Eq. (1). Figure 1 shows the change in divergence of $d_{\Psi(z)}(1.0, g)$ when $\Psi(z) = z\log(z) - (1+z)\log(1+z)$ and g is varied. This figure shows that $d_{\Psi(z)}(1.0, g)$ reaches its minimum at $g = 1.0$ and is convex near that point, but after that, the gradient decreases as g increases. Based on this observation, we derive the following proposition:

Proposition 1.

(i) When $\Psi(z) = z\log(z) - (1+z)\log(1+z)$, $d_{\Psi(z)}(f, g)$ is a non-convex function which has an inflection point $f + \sqrt{f+1}$.

(ii) When $\Psi(z) = z\log(z) - (1+z)\log(1+z)$, the gradient of $d_{\Psi(z)}(f, g)$ monotonically decreases in the region where $g > f + \sqrt{f+1}$.

(iii) When $\Psi(z) = z\log(z) - (1+z)\log(1+z)$ and $f \geqq 0$, the inflection point $f + \sqrt{f+1}$ of $d_{\Psi(z)}(f, g)$ satisfies $f + \sqrt{f+1} \geqq 1$.

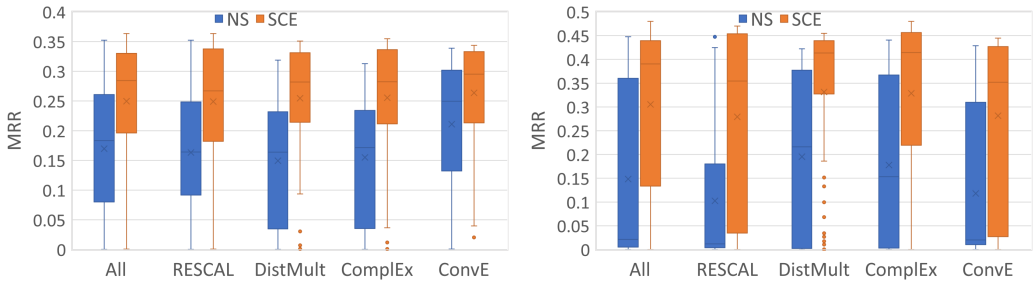
(iv) When $\Psi(z) = z\log(z) - (1+z)\log(1+z)$ and $f \geqq 0$, $d_{\Psi(z)}(f, g)$ is convex in the region $0 \leqq g \leqq 1$.

(v) When $s_\theta(x, y) \geqq 0$ is always satisfied, the NS loss behaves as a convex function.

We describe the proof of Prop. 1 in Appendix B.1. From Prop. 1, we can understand that the NS loss function is non-convex, but has a partially convex region. The gradient of the divergence decreases when $\exp(-s_\theta(x, y))$ is larger than the inflection point in the NS loss function. Therefore, in this case, it is desirable to use a larger learning rate as $\exp(-s_\theta(x, y))$ moves away from the region. On the other hand, when $\exp(-s_\theta(x, y))$ is smaller than the inflection point, the NS loss function is convex in this region, and the gradient of the divergence increases monotonically, so a small learning rate is desirable. Thus, the NS loss function has two regions with different properties in terms of the gradient. These characteristics show that it is difficult to set the learning rate appropriately when fitting a model to each (x, y) in training data using the NS loss.

Having two regions with different properties in the NS loss also causes a problem that the training is greatly affected by the initial value of $\exp(-s_\theta(x, y))$. If $\exp(-s_\theta(x, y))$ is placed in a region larger than the inflection point at the beginning of training, it requires a large number of updates to fit the model to the training data because the gradient is small in this region. On the other hand, if $\exp(-s_\theta(x, y))$ is initially placed in a region smaller than the inflection point, it can be fitted to the training data with fewer updates according to the convexity. As a result, depending on the initial value, an unbalanced model is learned in which the degree of fit of the model to each (x, y) differs greatly. In this case, the performance of the learned model may be degraded.

From Eq. (3) and Prop. 1 (i), the inflection point of the NS loss function is $f + \sqrt{f+1}$ with $f = \frac{\nu p_n(y|x)}{p_d(y|x)}$. Since the inflection point is affected by the noise distribution $p_n(y|x)$ and the number



(a) FB15k-237 *(b)* WN18RR

Figure 2: The box charts of the validation MRR for each model in FB15k-237 and WN18RR.

of samples ν in the NS loss, we can conclude that we should adjust the noise distribution, the number of samples, the initial value, and the learning rate appropriately to train the model in the convex region of the NS loss. Assuming a model consisting of a monotonically increasing activation function passing through the vicinity of the origin, initializing the model weights near zero and using a small learning rate may solve the problem since the model's output value may be in the convex region located near zero. However, since the output value of the model is affected by various factors, hyper-parameter tuning is still necessary.

To investigate whether careful hyper-parameter tuning is actually required in practical settings, we used publicly available large-scale hyper-parameter tuning results (Ruffinelli et al., 2020)[3]. This dataset contains the result of RESCAL (Bordes et al., 2011), DistMult (Yang et al., 2015), ComplEx (Trouillon et al., 2016), and ConvE (Dettmers et al., 2018a) trained by the NS and softmax cross-entropy (SCE) losses with different hyper-parameter settings in FB15k-237 and WN18RR. We divided the included 860 results for each dataset into two groups, learned by the SCE loss or the NS loss, and compared their Mean Reciprocal Rank (MRR). Figure 2 shows the results. The box charts show that the highest scores of the NS loss and SCE loss are close, but the NS loss only achieves the score close to the highest scores in fewer hyper-parameter settings than the SCE loss. Thus, achieving good performance in the NS loss function is difficult compared to a convex loss function such as the SCE loss function due to difficulty setting appropriate hyper-parameters.

Through the analysis in this section, we have found some problems in NS loss due to its non-convexity. In the next section, we discuss how to solve this problem.

4 Self-smoothing Negative Sampling

In this section, we derive the condition to solve the problem of non-convexity in the NS loss shown in the previous section and derive a variant of the NS loss that satisfies the conditions. We also examine the difference between the NS loss derived by us and an existing variant of the NS loss.

4.1 Conditions for Convexity

From Prop. 1. (v), we can understand that the NS loss behaves as a convex function when we use a score function that can only output non-negative values. This theoretical fact is along with the observation of Kamigaito & Hayashi (2021) that the NS loss can outperform the SCE loss only in the case of RotatE and TransE, which are non-negative scoring methods.

However, there is one problem in learning with the NS loss that satisfies the condition of the Prop. 1. (v). That is, under this condition, $\frac{\nu p_n(y|x)}{p_d(y|x)} \geqq 0$ in Eq. (3), while $0 \leqq \frac{1}{\exp(s_\theta(x,y))} \leqq 1$. Therefore, the problem is that there are cases where it is difficult to fit a model to a training data where $\frac{\nu p_n(y|x)}{p_d(y|x)} > 1$. To deal with this problem, it is necessary to select ν and $p_n(y|x)$ appropriately. However, since $p_d(y|x)$ changes depending on the target training data, it is difficult to derive a general condition. Therefore, in this paper, we only discuss the appropriate ν and $p_n(y|x)$ for the case of KGE.

[3]https://github.com/uma-pi1/kge-iclr20

To explain knowledge graph embedding, we define a tuple consisting of a relation r_k between entities e_i and e_j as (e_i, e_j, r_k). In KGE, a model predicts such a tuple. Specifically, for queries of the form $(e_i, e_j, ?)$ or $(?, e_j, r_k)$, the model predicts the entity at the position ?. In the following explanations, all such queries are denoted by x, and entities to be predicted are denoted by y. In a knowledge graph, the frequency of each tuple is at most one. Therefore, we can assume that $p_d(x,y)$ in the knowledge graph is uniformly distributed. In this case, we can derive the following relations.

Proposition 2. *In a knowledge graph, when $p_d(x,y)$ and $p_n(y|x)$ are uniformly distributed, $p_d(y|x) \geqq p_n(y|x)$ is satisfied.*

We describe the proof of Prop. 2 in Appendix B.2. From Prop. 2, we can induce that the condition for satisfying $\frac{\nu p_n(y|x)}{p_d(y|x)} \leqq 1$ is setting $p_n(y|x)$ to be uniformly distributed and $\nu = 1$. In this condition, there are no unfitting cases in KGE when we use a score function that can only output non-negative values in the NS loss. However, it is not realistic to set the number of samples to 1, because it would make learning unstable. Therefore, in the next Section, we derive a variant of the NS loss that can satisfy our induced conditions without setting ν as 1.

4.2 Derivation

In this Section, we derive a variant of the NS loss that satisfies the conditions shown in the previous Section for eliminating unfit cases when using a score function that outputs only non-negative values. For this purpose, there are two problems to be solved as follows:

1. Although setting an appropriate number of samples is important for efficient learning as explained in Section3, we have to set the number of samples $\nu = 1$.

2. Even though Kamigaito & Hayashi (2021) show that appropriate noise distribution is important in smoothing the learning results, we cannot choose a noise distribution other than the uniform distribution.

About the first problem, since the link prediction of KGE only focuses on the rank of predicted values, we can cancel the constant value ν from $\frac{\nu p_n(y|x)}{p_d(y|x)}$ with keeping proportional relationships of the predicted values for each (x,y) in terms of the optimal solution (Kamigaito & Hayashi, 2021). Canceling ν from $\frac{\nu p_n(y|x)}{p_d(y|x)}$ makes an additional advantage that the inflection point is no longer affected by ν in training. The second problem can be dealt with by performing smoothing in a way other than changing the noise distribution $p_n(y|x)$. Instead, we directly impose a smoothing value α to $p_d(y|x)$ as $p_d(y|x)^\alpha$. Based on the above solution, we derive a variant of the NS loss from $B_{\Psi(z)}(\frac{p_n(y|x)}{p_d(y|x)^\alpha}, \frac{1}{\exp(s_\theta(x,y))})$. Letting $G(y|x;\theta) = \frac{1}{\exp(s_\theta(x,y))}$, $p_n(y|x)$ is uniform, $u = (x,y)$, $f(u) = \frac{p_n(y|x)}{p_d(y|x)^\alpha}$, $g(u) = G(y|x;\theta)$, $\Psi(g(u)) = g(u)\log(g(u)) - (1+g(u))\log(1+g(u))$, and $\nabla_g \Psi(g(u)) = \log(g(u)) - \log(1+g(u))$, we derive the variant of NS loss function as follows:

$$
\begin{aligned}
&B_{\Psi(z)}(\frac{p_n(y|x)}{p_d(y|x)^\alpha}, \frac{1}{\exp(s_\theta(x,y))}) = \sum_{x,y}\Big[-\Psi(g(u)) + \nabla_g\Psi(g(u))g(u) - \nabla_g\Psi(g(u))f(u)\Big]p_d(x,y) \\
=&\sum_{x,y}\Big[-g(u)\log(1+g(u)) + (1+g(u))\log(1+g(u)) \\
&+\log(g(u))g(u) + \log(1+g(u))g(u) - \log(g(u))f(u) + \log(1+g(u))f(u)\Big]p_d(x,y) \\
=&\sum_{x,y}\Big[\log(1+g(u)) - \log(g(u))f(u) + \log(1+g(u))f(u)\Big]p_d(x,y) \\
=&\sum_{x,y}\left[\log(1+g(u)) + \log(1+\frac{1}{g(u)})f(u)\right]p_d(x,y)
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{x,y} p_d(y|x) \log(1+g(u)) \frac{1}{p_d(y|x)} p_d(x,y) + \sum_{x,y} p_n(y|x) \log(1+\frac{1}{g(u)}) \frac{1}{p_d(y|x)^{\alpha}} p_d(x,y) \\
&= \sum_{x,y} p_d(y|x) \log(1+G(y|x;\theta)) p_d(x) + \sum_{x,y} p_n(y|x) \log(1+\frac{1}{G(y|x;\theta)}) p_d(y|x)^{1-\alpha} p_d(x) \\
&= \frac{1}{|D|} \sum_{(x,y)\in D} \log(1+G(y|x;\theta)) + \frac{1}{\nu|D|} \sum_{(x,y)\in D} p_d(y|x)^{1-\alpha} \sum_{i=1,y_i\sim p_n}^{\nu} \log(1+\frac{1}{G(y_i|x;\theta)}) \\
&= -\frac{1}{|D|} \sum_{(x,y)\in D} \log(\frac{1}{1+G(y|x;\theta)}) - \frac{1}{\nu|D|} \sum_{(x,y)\in D} p_d(y|x)^{1-\alpha} \sum_{i=1,y_i\sim p_n}^{\nu} \log(\frac{G(y_i|x;\theta)}{1+G(y_i|x;\theta)}) \\
&= -\frac{1}{|D|} \sum_{(x,y)\in D} \log(\sigma(s_\theta(x,y))) - \frac{1}{\nu|D|} \sum_{(x,y)\in D} p_d(y|x)^{1-\alpha} \sum_{i=1,y_i\sim p_n}^{\nu} \log(\sigma(-s_\theta(x,y))) \\
&= -\frac{1}{|D|} \sum_{(x,y)\in D} \left(\log(\sigma(s_\theta(x,y))) + \frac{p_d(y|x)^{1-\alpha}}{\nu} \sum_{i=1,y_i\sim uniform}^{\nu} \log(\sigma(-s_\theta(x,y))) \right) \qquad (4)
\end{aligned}
$$

Our new variant of the NS loss is represented by Eq. (4). This expression contains $p_d(y|x)$ on the right-hand side. However, since $p_d(y|x)$ is considered to be the true distribution behind the all training examples, we cannot use $p_d(y|x)$ directly. Therefore, based on the idea of Sun et al. (2019), we use the output of the model under training. Specifically, we approximate $p_d(y|x)$ as follows:

$$
p_d(y|x) \approx \tilde{p}_d(y|x) = \frac{\exp(s_\theta(x,y))}{\sum_{y'\in Y'} \exp(s_\theta(x,y'))}, \qquad (5)
$$

Y' is the set of the positive examples and the negative examples obtained by sampling during training. Note that $\tilde{p}_d(y|x)$ is close to $p_d(y|x)$ through training when $p_n(y|x)$ is a uniform distribution (Kamigaito & Hayashi, 2021). We name the method derived in Eq. (4) *self-smoothing negative-sampling* (SSNS) because it uses the predictions of the training model for the smoothing.

4.3 Differences from Self-Adversarial Negative Sampling

In this Section, we explain the difference between our derived SSNS explained in the previous Section and Self-Adversarial Negative Sampling (SANS), which also uses the prediction results of the model under training. Letting $|Y|$ be the number of labels and $p_{sans}(y|x) = \frac{\nu}{|Y|}\tilde{p}_d(y|x)$, we can derive SANS in the form of Eq. (3) as follows:

$$
B_{\Psi(z)}(\frac{p_{sans}(y|x)}{p_d(y|x)}, \frac{1}{\exp(s_\theta(x,y))}) = -\frac{1}{|D|} \sum_{(x,y)\in D} \Big[\log(\sigma(s_\theta(x,y))) + \sum_{i=1,y_i\sim uniform}^{\nu} \tilde{p}_d(y|x) \log(\sigma(-s_\theta(x,y))) \Big]. \qquad (6)
$$

We describe the detailed derivation of Eq. (6) in Appendix A.1. From this derivation, we can see that in the SANS loss, when the loss is zero, $\frac{1}{\exp(s_\theta(x,y))} = \frac{p_{sans}(y|x)}{p_d(y|x)}$ is satisfied. Different from the SSNS loss function, the SANS loss functions is influenced by the number of negative sampling ν. Unlike the NS loss, the SANS loss is not directly affected by ν, since it is divided by $|Y|$. This division is effective to maintain $\frac{\nu p_n(y|x)}{p_d(y|x)} \leqq 1$ for decreasing the hard-to-fit training examples. Through training, $\tilde{p}_d(y|x)$ gets closer to $p_d(y|x)$. However, if (x,y) is an uncertain example (Chang et al., 2017) that increases the variance of $\tilde{p}_d(y|x)$, there is a possibility that $\tilde{p}_d(y|x)$ becomes large and $\frac{p_{sans}(y|x)}{p_d(y|x)} > 1$, which leads to the existence of hard-to-fit cases and it causes under-fitting. In KGE, relations other than 1-to-1, such as 1-to-N, M-to-1, and M-to-N relations are the uncertain examples. Therefore, SANS may not work well on datasets including such the relations, while SSNS may work well. However, since these properties are affected by various factors during training, it is difficult to verify this only from the theoretical aspect. Therefore, we need to verify the existence of such a relationship through actual experiments.

5 Analysis

This Section compares the SANS and SSNS loss functions on various datasets to verify whether SSNS is robust to datasets that contain a variety of relations other than 1-to-1 relations. We use

Table 1: The numbers of each instance for each dataset.

Dataset	Entities	Relations	Tuples		
			Train	Valid	Test
FB15k-237	14,541	237	272,115	17,535	20,466
WN18RR	40,943	11	86,835	3,034	3,134
YAGO3-10	123,182	37	1,079,040	4978	4982

Table 2: Results for each loss function and model in each dataset. The bold value indicates the best score. † indicates the score is significantly different from the second best score[4].

Dataset	Loss	RotatE				TransE			
		MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
FB15k-237	NS	29.9	19.9	33.1	50.5	28.2	17.8	31.5	49.8
	SANS	33.4	23.8	37.1	**52.5**	**32.8**	23.1	**36.6**	**52.5**
	SSNS	**33.6**	**24.5**	**37.1**	51.7	32.6	**23.5**	36.2	50.9
WN18RR	NS	46.8	42.6	48.4	54.8	21.8	1.3	39.5	51.0
	SANS	47.2	43.1	48.6	55.2	22.3	1.5	**39.9**	52.2
	SSNS	**47.5**	**43.4**	**48.9**	**55.3**	**22.5**	**2.6**†	39.0	**52.1**
YAGO3-10	NS	47.7	38.1	53.2	66.2	47.7	37.7	53.7	66.2
	SANS	49.0	39.7	54.6	66.2	49.0	39.2	55.0	66.6
	SSNS	**50.8**†	**41.9**†	**56.3**†	**67.0**	**49.9**†	**40.5**†	**55.7**	**66.8**

three datasets for validation: WN18RR (Dettmers et al., 2018c); FB15k-237 (Toutanova & Chen, 2015); YAGO3-10 (Dettmers et al., 2018b). Table 1 shows the statistics for each dataset. As we can see from these statistics, WN18RR and YAGO3-10 are datasets that consist of a small number of relations for many entities. In particular, since YAGO3-10 is a dataset designed to contain at least ten relations for one entity, it is suitable for verifying our assumption. We used the MRR, Hits@1, Hits@3, and Hits@10 as evaluation metrics.

We used two typical non-negative scoring functions, TransE (Wang et al., 2014) and RotatE (Sun et al., 2019), as the models. We used LibKGE as our implementation. For the hyperparameters for each dataset, we used the values used in the previous study Sun et al. (2019). We chose the uniform distribution for training the NS loss. For the SANS loss, we used the weights learned by the NS loss with the same configuration to our baseline as initial values only for the training of RotatE in YAGO3-10 because the learning did not progress well with random initialization. Since the SSNS loss requires the output of a model close to the true distribution for training, we used the weights of the model trained by SANS with the same configuration to our baselines as initial weights. For a fair comparison with the SANS loss, we also performed fine-tuning for the SANS loss using the weights trained with the SANS loss, and we selected the model with the highest MRR in the developed data. For selecting the training results, we trained up to 800 epochs, evaluated the training results every five epochs, and selected the model with the highest MRR. On the other hand, we found that the convergence time of SSNS loss was faster for FB15k-237 and YAGO3-10 than the other losses, so we trained up to five epochs in the SSNS loss, evaluated the training results every epoch, and selected the model with the maximum MRR in the development dataset. We also tuned the α with the development dataset since it has different properties from the temperature parameter of the SANS loss. Please refer to the Appendix for the detailed description of the tuning.

Table 2 shows the experimental results. We can see from the experimental results that both TransE and RotatE perform better when using the SANS loss than when using the NS loss. As we explained in Section 4.3, the SANS loss is less sensitive to the large number of samples commonly used in KGE. This result is along with our theoretical expectation explained in subsection 4.3.

Next, we discuss the performance of the SANS loss and the SSNS loss. From the results of FB15k-237 and WN18RR, we can observe the similar performance of the SSNS loss in RotatE. This is

[4]We use the student's t-test with a confidence interval of 95%. We describe the details in Appendix C.2.

along with our theoretical expectation that the two losses are robust to the large number of samples commonly used in KGE. On the other hand, in YAGO3-10, the SSNS loss achieves a significant performance improvement over the SANS loss in both RotatE and TransE. Since YAGO3-10 is a dataset designed to contain at least ten relations per entity, this result is consistent with our expectation that the SSNS loss is more effective against the SANS loss on a dataset containing a variety of relations other than the 1-to-1 relations as we explained in subsection 4.3.

6 Related Work

Mikolov et al. (2013) proposed the NS loss function to approximate the softmax cross-entropy loss function to reduce the computational cost for training a word representation model word2vec. Initially, due to its similarity to noise contrastive estimation (NCE) (Gutmann & Hyvärinen, 2010), a theoretical analysis of the differences between the NS loss and the NCE loss was conducted by (Dyer, 2014), but this study did not reveal the clear properties of the NS loss. In particular, the work of Levy & Goldberg (2014) attracted attention because it showed that the result using the NS loss in word2vec is equivalent to the decomposition of the PMI matrix. However, this equivalence is only valid when the unigram distribution is used as the noise distribution, and thus it is not valid for other than the word representation task.

Since then, due to its usefulness, the NS loss has been used in various tasks such as word representation (Mikolov et al., 2013; Bojanowski et al., 2017), language modeling (Melamud et al., 2017), pretrained language model (Clark et al., 2020b;a), and knowledge graph embedding (KGE) (Trouillon et al., 2016). In particular, NS loss has recently been widely used in KGE. However, since various scoring methods and loss functions are used in combination in knowledge graph embedding, the properties of the loss function are not clear. Ruffinelli et al. (2020); Ali et al. (2020) explored these combinations in a large-scale experiment and empirically derived which combination is better. On the other hand, Kamigaito & Hayashi (2021) clarified the properties of the SCE loss function and the NS loss function by theoretically identifying the differences between them. They also pointed out that the score function that outputs only non-negative values is unsuitable for the SCE loss. Our paper is novel in focusing on the non-convexity and the existence of inflection points in the NS loss function and clarifying the problems of the NS loss function. Furthermore, our paper is useful not only for clarifying the problem but also for proposing a method to solve the problem in KGE.

7 Conclusion

In this paper, we focused on the non-convexity of the NS loss function and theoretically investigated why learning does not work well when using the NS loss. As a result, we found that we should adjust the noise distribution, the number of samples, the initial value, and the learning rate appropriately to learn a model on a convex region in the NS loss function.

We also showed conditions for training a model on the convex region in the NS loss and showed that the NS loss is suitable for a score function that outputs only non-negative values. On the other hand, we also indicated that there are cases where the NS loss cannot properly fit the model to the training examples under these conditions. Furthermore, we proposed a new variant of the NS loss function, the self-smoothing negative sampling (SSNS) loss function, which can solve the problem by focusing on KGE. Since we theoretically expected the proposed SSNS loss function to improve the performance of a dataset containing a variety of relations other than 1-to-1 relations, we performed experiments using the actual models and datasets.

Experimental results using the non-negative scoring methods TransE and RotatE showed that the proposed SSNS could improve the performance of SANS on the FB15k-237, WN18RR, and YAGO3-10 datasets. In particular, we observed a significant performance improvement on YAGO3-10, a dataset that contains a variety of relations other than the 1-to-1 relations. This observation is consistent with our expectation in terms of the theoretical aspect.

This paper dealt with the NS loss's inability to properly fit the model to the training examples by focusing on KGE. Thus, we would like to explore ways to solve a similar task outside of KGE, such as item recommendation, by using a non-negative scoring method in the future.

REFERENCES

Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. Pykeen 1.0: A python library for training and evaluating knowledge graph emebddings. *arXiv preprint arXiv:2007.14175*, 2020.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. doi: 10.1162/tacl_a_00051. URL https://aclanthology.org/Q17-1010.

Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, pp. 301–306. AAAI Press, 2011.

L.M. Bregman. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7(3):200 – 217, 1967. ISSN 0041-5553. doi: https://doi.org/10.1016/0041-5553(67)90040-7. URL http://www.sciencedirect.com/science/article/pii/0041555367900407.

Haw-Shiuan Chang, Erik Learned-Miller, and Andrew McCallum. Active bias: Training more accurate neural networks by emphasizing high variance samples. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/2f37d10131f2a483a8dd005b3d14b0d9-Paper.pdf.

Kevin Clark, Minh-Thang Luong, Quoc Le, and Christopher D. Manning. Pre-training transformers as energy-based cloze models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 285–294, Online, November 2020a. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.20. URL https://www.aclweb.org/anthology/2020.emnlp-main.20.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*, 2020b. URL https://openreview.net/forum?id=r1xMH1BtvB.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI*, pp. 1811–1818, 2018a. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17366.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Thirty-second AAAI conference on artificial intelligence*, 2018b.

Tim Dettmers, Minervini Pasquale, Stenetorp Pontus, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the 32th AAAI Conference on Artificial Intelligence*, pp. 1811–1818, February 2018c. URL https://arxiv.org/abs/1707.01476.

Chris Dyer. Notes on noise contrastive estimation and negative sampling. *arXiv preprint arXiv:1410.8251*, 2014.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 297–304, 2010.

Michael U. Gutmann and Jun-ichiro Hirayama. Bregman divergence as general framework to estimate unnormalized statistical models. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, UAI'11, pp. 283–290, Arlington, Virginia, USA, 2011. AUAI Press. ISBN 9780974903972.

Hidetaka Kamigaito and Katsuhiko Hayashi. Unified interpretation of softmax cross-entropy and negative sampling: With case study for knowledge graph embedding. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint*

Conference on Natural Language Processing (Volume 1: Long Papers), pp. 5517–5531, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.429. URL https://aclanthology.org/2021.acl-long.429.

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pp. 2177–2185, Cambridge, MA, USA, 2014. MIT Press.

Oren Melamud, Ido Dagan, and Jacob Goldberger. A simple language model based on PMI matrix approximations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1860–1865, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1198. URL https://www.aclweb.org/anthology/D17-1198.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You can teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BkxSmlBFvr.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HkgEQnRqYQ.

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*, pp. 57–66, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-4007. URL https://www.aclweb.org/anthology/W15-4007.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *ICML*, pp. 2071–2080, 2016. URL http://proceedings.mlr.press/v48/trouillon16.html.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, pp. 1112–1119. AAAI Press, 2014.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6575.

A DERIVATIONS

This section describes the detailed derivations of the loss functions discussed in this paper.

A.1 DERIVATION FOR THE NS LOSS FUNCTION

In this subsection, we describe the details of the derivation for the NS loss function in Eq. ((3). Letting θ be the model parameter, $s_\theta(x,y)$ be a scoring function to make $s_\theta(x,y)$, ν be a number of negative samples, $p_n(y|x)$ be a noise distribution, σ be a sigmoid function, $\Psi(z) = z\log(z) - (1+z)\log(1+z)$, $G(y|x;\theta) = \frac{1}{\exp(s_\theta(x,y))}$, $u = (x,y)$, $f(u) = \frac{p_n(y|x)}{p_d(y|x)}$, $g(u) = G(y|x;\theta)$, $\Psi(g(u)) = g(u)\log(g(u)) - (1+g(u))\log(1+g(u))$, and $\nabla_g \Psi(g(u)) = \log(g(u)) - \log(1+g(u))$, we can derive the NS loss function as follows:

$$
\begin{aligned}
& B_{\Psi(z)}\Big(\frac{\nu p_n(y|x)}{p_d(y|x)}, \frac{1}{\exp(s_\theta(x,y))}\Big) \\
=& \sum_{x,y} \Big[-\Psi(g(u)) + \nabla_g \Psi(g(u)) g(u) - \nabla_g \Psi(g(u)) f(u) \Big] p_d(x,y) \\
=& \sum_{x,y} \Big[-g(u)\log(1+g(u)) + (1+g(u))\log(1+g(u)) \\
& + \log(g(u))g(u) + \log(1+g(u))g(u) - \log(g(u))f(u) + \log(1+g(u))f(u) \Big] p_d(x,y) \\
=& \sum_{x,y} \Big[\log(1+g(u)) - \log(g(u))f(u) + \log(1+g(u))f(u) \Big] p_d(x,y) \\
=& \sum_{x,y} \left[\log(1+g(u)) + \log(1+\frac{1}{g(u)})f(u) \right] p_d(x,y) \\
=& \sum_{x,y} p_d(y|x)\log(1+g(u))\frac{1}{p_d(y|x)} p_d(x,y) + \sum_{x,y} p_n(y|x)\log(1+\frac{1}{g(u)})\frac{1}{p_d(y|x)} p_d(x,y) \\
=& \sum_{x,y} p_d(y|x)\log(1+G(y|x;\theta)) p_d(x) + \sum_{x,y} p_n(y|x)\log(1+\frac{1}{G(y|x;\theta)}) p_d(x) \\
=& \frac{1}{|D|} \sum_{(x,y)\in D} \log(1+G(y|x;\theta)) + \frac{1}{|D|} \sum_{(x,y)\in D} \sum_{i=1, y_i \sim p_n}^{\nu} \log(1+\frac{1}{G(y_i|x;\theta)}) \\
=& -\frac{1}{|D|} \sum_{(x,y)\in D} \log(\frac{1}{1+G(y|x;\theta)}) - \frac{1}{|D|} \sum_{(x,y)\in D} \sum_{i=1, y_i \sim p_n}^{\nu} \log(\frac{G(y_i|x;\theta)}{1+G(y_i|x;\theta)}) \\
=& -\frac{1}{|D|} \sum_{(x,y)\in D} \log(\sigma(s_\theta(x,y))) - \frac{1}{|D|} \sum_{(x,y)\in D} \sum_{i=1, y_i \sim p_n}^{\nu} \log(\sigma(-s_\theta(x,y))) \\
=& -\frac{1}{|D|} \sum_{(x,y)\in D} \Big[\log(\sigma(s_\theta(x,y))) + \sum_{i=1, y_i \sim p_n}^{\nu} \log(\sigma(-s_\theta(x,y))) \Big] \\
=& -\frac{1}{|D|} \sum_{(x,y)\in D} \Big[\log(\sigma(s_\theta(x,y))) + \nu \mathop{\mathbb{E}}_{y_i \sim p_n} \log(\sigma(-s_\theta(x,y))) \Big]. \qquad (7)
\end{aligned}
$$

A.2 Derivation for the SANS Loss Function

In this subsection, we describe the details of the derivation for the SANS loss function in Eq. ((6). Letting θ be the model parameter, $s_\theta(x,y)$ be a scoring function to make $s_\theta(x,y)$, ν be a number of negative samples, $|Y|$ be the number of labels, Y' be sampled examples, $\tilde{p}_d(y|x) = \frac{\exp(s_\theta(x,y))}{\sum_{y'\in Y'}\exp(s_\theta(x,y'))}$, $p_{sans}(y|x) = \frac{\nu}{|Y|}\tilde{p}_d(y|x)$, σ be a sigmoid function, $\Psi(z) = z\log(z) - (1+z)\log(1+z)$, $G(y|x;\theta) = \frac{1}{\exp(s_\theta(x,y))}$, $u = (x,y)$, $f(u) = \frac{p_n(y|x)}{p_d(y|x)}$, $g(u) = G(y|x;\theta)$, $\Psi(g(u)) = g(u)\log(g(u)) - (1+g(u))\log(1+g(u))$, and $\nabla_g\Psi(g(u)) = \log(g(u)) - \log(1+g(u))$, we can derive the NS loss function as follows:

$$
\begin{aligned}
&B_{\Psi(z)}(\frac{p_{sans}(y|x)}{p_d(y|x)}, \frac{1}{\exp(s_\theta(x,y))}) \\
=&\sum_{x,y}\Big[-\Psi(g(u)) + \nabla_g\Psi(g(u))g(u) - \nabla_g\Psi(g(u))f(u)\Big]p_d(x,y) \\
=&\sum_{x,y}\Big[-g(u)\log(1+g(u)) + (1+g(u))\log(1+g(u)) \\
&+\log(g(u))g(u) + \log(1+g(u))g(u) - \log(g(u))f(u) + \log(1+g(u))f(u)\Big]p_d(x,y) \\
=&\sum_{x,y}\Big[\log(1+g(u)) - \log(g(u))f(u) + \log(1+g(u))f(u)\Big]p_d(x,y) \\
=&\sum_{x,y}\left[\log(1+g(u)) + \log(1+\frac{1}{g(u)})f(u)\right]p_d(x,y) \\
=&\sum_{x,y}p_d(y|x)\log(1+g(u))\frac{1}{p_d(y|x)}p_d(x,y) + \sum_{x,y}p_{sans}(y|x)\log(1+\frac{1}{g(u)})\frac{1}{p_d(y|x)}p_d(x,y) \\
=&\sum_{x,y}p_d(y|x)\log(1+G(y|x;\theta))p_d(x) + \sum_{x,y}p_{sans}(y|x)\log(1+\frac{1}{G(y|x;\theta)})p_d(x) \\
=&\frac{1}{|D|}\sum_{(x,y)\in D}\log(1+G(y|x;\theta)) + \frac{1}{|D|}\sum_{(x,y)\in D}\sum_{i=1,y_i\sim uniform}^{\nu}\tilde{p}_d(y|x)\log(1+\frac{1}{G(y_i|x;\theta)}) \\
=&-\frac{1}{|D|}\sum_{(x,y)\in D}\log(\frac{1}{1+G(y|x;\theta)}) - \frac{1}{|D|}\sum_{(x,y)\in D}\sum_{i=1,y_i\sim uniform}^{\nu}\tilde{p}_d(y|x)\log(\frac{G(y_i|x;\theta)}{1+G(y_i|x;\theta)}) \\
=&-\frac{1}{|D|}\sum_{(x,y)\in D}\log(\sigma(s_\theta(x,y))) - \frac{1}{|D|}\sum_{(x,y)\in D}\sum_{i=1,y_i\sim uniform}^{\nu}\tilde{p}_d(y|x)\log(\sigma(-s_\theta(x,y))) \\
=&-\frac{1}{|D|}\sum_{(x,y)\in D}\Big[\log(\sigma(s_\theta(x,y))) + \sum_{i=1,y_i\sim uniform}^{\nu}\tilde{p}_d(y|x)\log(\sigma(-s_\theta(x,y)))\Big] \\
=&-\frac{1}{|D|}\sum_{(x,y)\in D}\Big[\log(\sigma(s_\theta(x,y))) + \nu\mathop{\mathbb{E}}_{y_i\sim uniform}\tilde{p}_d(y|x)\log(\sigma(-s_\theta(x,y)))\Big].
\end{aligned}
\tag{8}
$$

Note that Kamigaito & Hayashi (2021) used $\tilde{p}_d(y|x)$ as a noise distribution for inducing Eq. (8), different from $\frac{1}{|Y|}\tilde{p}_d(y|x)$, we used. This is because $\frac{1}{|Y|}$ is constant and it is canceled in the softmax function.

B PROOFS

B.1 PROPOSITION. 1

Table 3: A derivative sign chart of $d_{\Psi(z)}(f,g)$ when $\Psi(z) = z\log(z) - (1+z)\log(1+z)$.

g	0		f		$f+\sqrt{f+1}$		∞
$d_{\Psi(z)}(f,g)$	$+$	$+$	0	$+$	$+$	$+$	$+$
$\nabla_g d_{\Psi(z)}(f,g)$	$-$	$-$	0	$+$	$+$	$+$	0
$\nabla_g^2 d_{\Psi(z)}(f,g)$	$+$	$+$	$+$	$+$	0	$-$	$-$

Proof. Calculating $\nabla_g d_{\Psi(z)}(f,g)$ and $\nabla_g^2 d_{\Psi(z)}(f,g)$ under $\Psi(z) = z\log(z) - (1+z)\log(1+z)$ leads to the derivative sign chart shown in Table 3. From this derivative sign chart, we can confirm that the Prop. 1. (i) and 1. (ii) are true.

Since $f+\sqrt{f+1}$ is a monotonically increasing function and this function is 1 when $f=0$, we can conclude that the Prop. 1. (iii) is true.

From the Prop. 1. (iii) and the Table 3, we can confirm that the Prop. 1. (iv) is satisfied since the second-order derivative is always positive in the domain $0 \leqq g \leqq 1$.

From Eq. (3) and Prop. 1. (iv), the NS loss behaves as a convex function if $0 \leqq \frac{1}{\exp(s_\theta(x,y))} \leqq 1$ is always satisfied. Then, since $\frac{1}{\exp(x)}$ is a monotonically decreasing function for x, $\frac{1}{\exp(s_\theta(x,y))}$ satisfies $0 \leqq \frac{1}{\exp(s_\theta(x,y))} \leqq 1$ when $s_\theta(x,y) \leqq 0$. Therefore, we can derive Prop. 1. (v). □

B.2 PROPOSITION. 2

Proof. We define the number of differences in x as $|X|$, the number of differences in y as $|Y|$, and the frequency of occurrence of x as $\#x$. In this case, $p_d(y|x) = \frac{p_d(x,y)}{p_d(x)} = \frac{1}{\#x}$ is satisfied. Here, $\#x \leqq |Y|$ is also satisfied, because (x,y) that indicates a tuple in a knowledge graph can appear only once in the knowledge graph. Therefore, since $\frac{1}{\#x} \geqq \frac{1}{|Y|}$ is satisfied, $p_d(y|x) \geqq p_n(y|x)$ is also satisfied. □

C EXPERIMENTAL DETAILS

C.1 SETTINGS

Table 4 and 5 show the hyparparameters of each model for each dataset.

Table 4: Hyparparameters of RotatE for each dataset.

RotatE in FB15k-237	
Batch size	1024
Dimension	1000
Number of samples	256
α	NS: None, SANS: 1.0, SSNS: 1.25
Initialize	NS: Xavier uniform w/ gain=1.0, SANS: Pretrained by SANS, SSNS: Pretrained by SANS
Regularize	None
Dropout	None
Optimizer	Adam
Learning rate	0.00005
Decay	0.95
Patience of learning rate	NS: 5 epochs, SANS: 5 epochs, SSNS: 1 epoch
Max epoch	NS: 800 epochs, SANS: 800 epochs, SSNS: 5 epochs
Validation	NS: Every 5 epoch, SANS: Every 5 epoch, SSNS: Every 1 epoch
Patience of learning	NS: 10 epochs, SANS: 10 epochs, SSNS: 5 epoch
RotatE in WN18RR	
Batch size	512
Dimension	500
Number of samples	1024
α	NS: None, SANS: 0.5, SSNS: 1.5
Initialize	NS: Xavier uniform w/ gain=1.0, SANS: Pretrained by SANS, SSNS: Pretrained by SANS
Regularize	None
Dropout	None
Optimizer	Adam
Learning rate	0.00005
Decay	0.95
Patience of learning rate	NS: 5 epochs, SANS: 5 epochs, SSNS: 5 epoch
Max epoch	NS: 800 epochs, SANS: 800 epochs, SSNS: 800 epochs
Validation	NS: Every 5 epoch, SANS: Every 5 epoch, SSNS: Every 5 epoch
Patience of learning	NS: 10 epochs, SANS: 10 epochs, SSNS: 10 epoch
RotatE in YAGO3-10	
Batch size	1024
Dimension	500
Number of samples	400
α	NS: None, SANS: 0.5, SSNS: 0.75
Initialize	NS: Xavier uniform w/ gain=1.0, SANS: Pretrained by NS, SSNS: Pretrained by SANS
Regularize	None
Dropout	None
Optimizer	Adam
Learning rate	0.0002
Decay	0.95
Patience of learning rate	NS: 5 epochs, SANS: 5 epochs, SSNS: 1 epoch
Max epoch	NS: 800 epochs, SANS: 800 epochs, SSNS: 5 epochs
Validation	NS: Every 5 epoch, SANS: Every 5 epoch, SSNS: Every 1 epoch
Patience of learning	NS: 10 epochs, SANS: 10 epochs, SSNS: 5 epoch

Table 5: Hyparparameters of TransE for each dataset.

TransE in FB15k-237	
Batch size	1024
Dimension	1000
Number of samples	256
α	NS: None, SANS: 1.0, SSNS: 0.75
Initialize	NS: Xavier uniform w/ gain=1.0, SANS: Pretrained by SANS, SSNS: Pretrained by SANS
Regularize	None
Dropout	None
Optimizer	Adam
Learning rate	0.00005
Decay	0.95
Patience of learning rate	NS: 5 epochs, SANS: 5 epochs, SSNS: 1 epoch
Max epoch	NS: 800 epochs, SANS: 800 epochs, SSNS: 5 epochs
Validation	NS: Every 5 epoch, SANS: Every 5 epoch, SSNS: Every 1 epoch
Patience of learning	NS: 10 epochs, SANS: 10 epochs, SSNS: 5 epoch
TransE in WN18RR	
Batch size	512
Dimension	500
Number of samples	1024
α	NS: None, SANS: 0.5, SSNS: 0.5
Initialize	NS: Xavier uniform w/ gain=1.0, SANS: Pretrained by SANS, SSNS: Pretrained by SANS
Regularize	None
Dropout	None
Optimizer	Adam
Learning rate	0.00005
Decay	0.95
Patience of learning rate	NS: 5 epochs, SANS: 5 epochs, SSNS: 5 epoch
Max epoch	NS: 800 epochs, SANS: 800 epochs, SSNS: 800 epochs
Validation	NS: Every 5 epoch, SANS: Every 5 epoch, SSNS: Every 5 epoch
Patience of learning	NS: 10 epochs, SANS: 10 epochs, SSNS: 10 epoch
TransE in YAGO3-10	
Batch size	1024
Dimension	500
Number of samples	400
α	NS: None, SANS: 0.5, SSNS: -64.0
Initialize	NS and SANS: Xavier uniform w/ gain=1.0, SSNS: Pretrained by SANS
Regularize	None
Dropout	None
Optimizer	Adam
Learning rate	0.0002
Decay	0.95
Patience of learning rate	NS: 5 epochs, SANS: 5 epochs, SSNS: 1 epoch
Max epoch	NS: 800 epochs, SANS: 800 epochs, SSNS: 5 epochs
Validation	NS: Every 5 epoch, SANS: Every 5 epoch, SSNS: Every 1 epoch
Patience of learning	NS: 10 epochs, SANS: 10 epochs, SSNS: 5 epoch

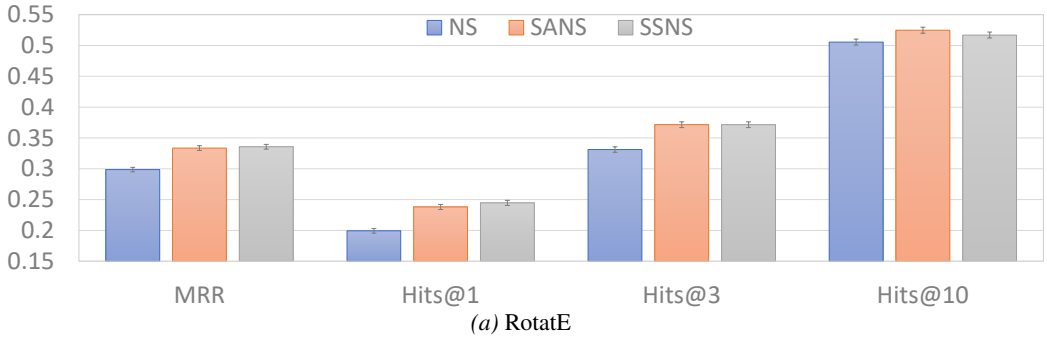
When tuning α in SSNS, we chose α from $[0.5, 0.75, 1.0, 1.25, 1.5]$ excluding the case of TransE in YAGO3-10. In this excluded case, due to the flat output of TransE, we needed to consider minus values for α in SSNS. Therefore, we chose α in SSNS for TransE in YAGO3-10 from $[0.0, -1.0, -4.0, -16.0, -64.0]$.

C.2 RESULTS

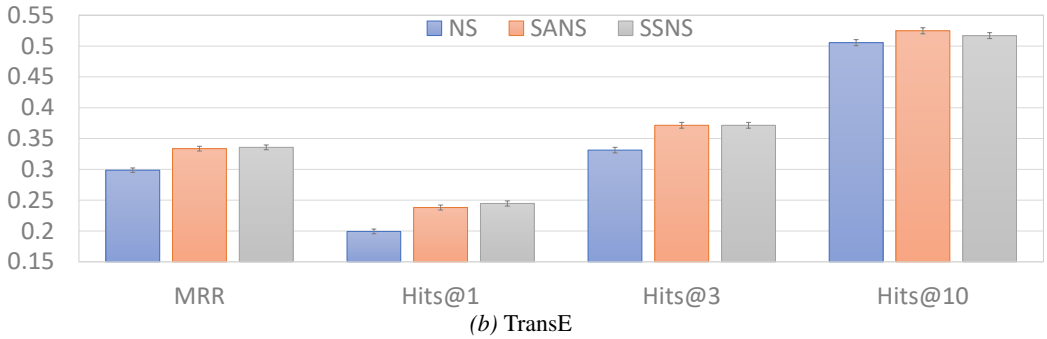
Table 6 shows the scores and their confidence intervals for each setting. Figure 3, 4, and 5 show the scores and their confidence intervals in Table 6 as bars and error bars, respectively.

Table 6: The scores and their confidence intervals of each model in each dataset.

RotatE in FB15k-237												
	MRR			Hits@1			Hits@3			Hits@10		
	Value	Conf. Interval		Value	Conf. Interval		Value	Conf. Interval		Value	Conf. Interval	
		From	To		From	To		From	To		From	To
NS	0.299	0.295	0.302	0.199	0.195	0.203	0.331	0.327	0.336	0.505	0.501	0.510
SANS	0.334	0.330	0.337	0.238	0.234	0.242	0.371	0.367	0.376	0.525	0.520	0.530
SSNS	0.336	0.332	0.340	0.245	0.241	0.249	0.371	0.367	0.376	0.517	0.512	0.522
TransE in FB15k-237												
	MRR			Hits@1			Hits@3			Hits@10		
	Value	Conf. Interval		Value	Conf. Interval		Value	Conf. Interval		Value	Conf. Interval	
		From	To		From	To		From	To		From	To
NS	0.282	0.278	0.285	0.178	0.175	0.182	0.315	0.311	0.320	0.498	0.493	0.503
SANS	0.328	0.324	0.332	0.231	0.227	0.235	0.366	0.361	0.370	0.525	0.520	0.530
SSNS	0.326	0.323	0.330	0.235	0.231	0.239	0.362	0.358	0.367	0.509	0.505	0.514
RotatE in WN18RR												
	MRR			Hits@1			Hits@3			Hits@10		
	Value	Conf. Interval		Value	Conf. Interval		Value	Conf. Interval		Value	Conf. Interval	
		From	To		From	To		From	To		From	To
NS	0.468	0.456	0.479	0.426	0.414	0.438	0.484	0.472	0.496	0.548	0.536	0.561
SANS	0.472	0.460	0.483	0.431	0.418	0.443	0.486	0.474	0.499	0.552	0.540	0.564
SSNS	0.475	0.463	0.487	0.434	0.422	0.447	0.489	0.476	0.501	0.553	0.540	0.565
TransE in WN18RR												
	MRR			Hits@1			Hits@3			Hits@10		
	Value	Conf. Interval		Value	Conf. Interval		Value	Conf. Interval		Value	Conf. Interval	
		From	To		From	To		From	To		From	To
NS	0.218	0.212	0.224	0.013	0.010	0.016	0.395	0.383	0.407	0.510	0.498	0.522
SANS	0.223	0.217	0.229	0.015	0.012	0.018	0.399	0.386	0.411	0.522	0.509	0.534
SSNS	0.225	0.219	0.231	0.026	0.022	0.030	0.390	0.378	0.402	0.521	0.508	0.533
RotatE in YAGO3-10												
	MRR			Hits@1			Hits@3			Hits@10		
	Value	Conf. Interval		Value	Conf. Interval		Value	Conf. Interval		Value	Conf. Interval	
		From	To		From	To		From	To		From	To
NS	0.477	0.469	0.486	0.381	0.371	0.390	0.532	0.522	0.542	0.662	0.652	0.671
SANS	0.490	0.482	0.499	0.397	0.387	0.407	0.546	0.536	0.555	0.662	0.652	0.671
SSNS	0.508	0.499	0.516	0.419	0.409	0.429	0.563	0.553	0.572	0.670	0.661	0.679
TransE in YAGO3-10												
	MRR			Hits@1			Hits@3			Hits@10		
	Value	Conf. Interval		Value	Conf. Interval		Value	Conf. Interval		Value	Conf. Interval	
		From	To		From	To		From	To		From	To
NS	0.477	0.469	0.486	0.377	0.367	0.386	0.537	0.528	0.547	0.662	0.653	0.671
SANS	0.490	0.481	0.498	0.392	0.383	0.402	0.550	0.540	0.560	0.666	0.656	0.675
SSNS	0.499	0.491	0.508	0.405	0.395	0.414	0.557	0.547	0.567	0.668	0.659	0.677

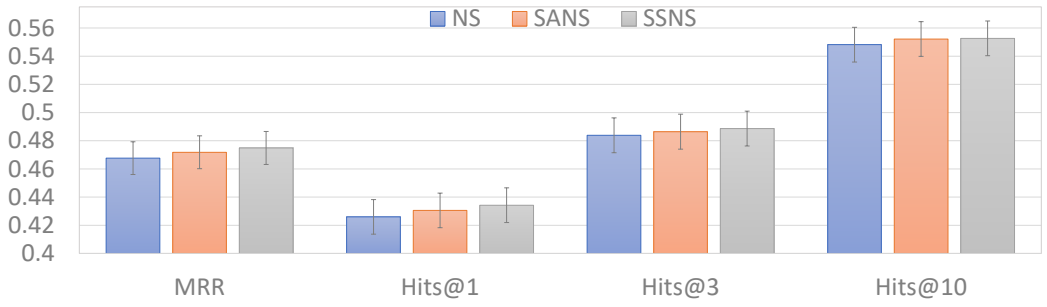


(a) RotatE

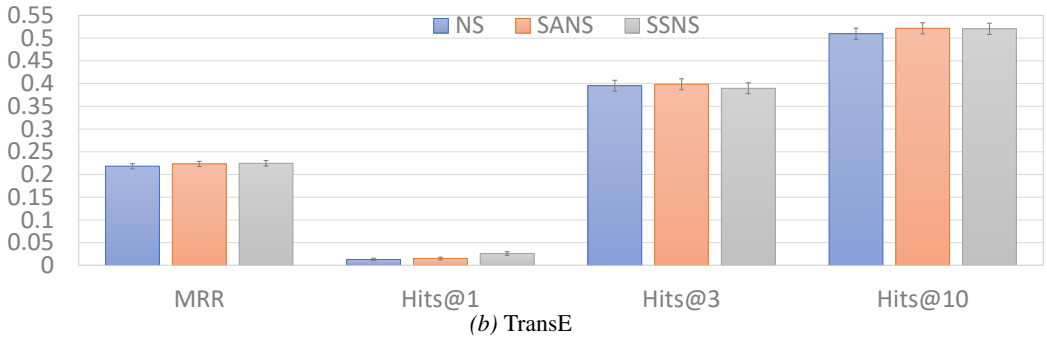


(b) TransE

Figure 3: The results of RotatE and TransE in FB15k-237. The error bar indicates the 95% confidence interval.

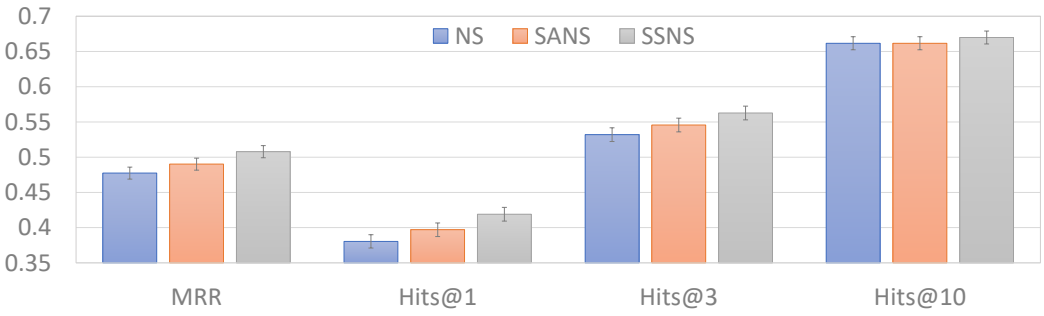


(a) RotatE

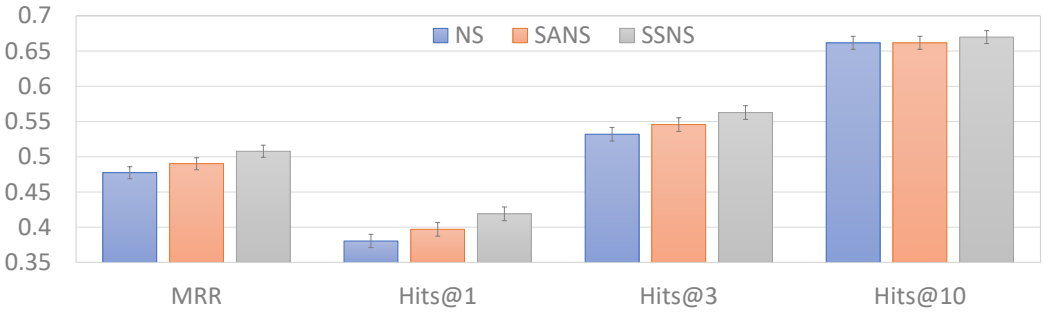


(b) TransE

Figure 4: The results of RotatE and TransE in WN18RR. The notations are the same as Fig. 3.



(a) RotatE



(b) TransE

Figure 5: The results of TransE in YAGO3-10. The notations are the same as Fig. 3.