# Ranking-Enhanced Unsupervised Sentence Representation Learning

**Yeon Seonwoo**[†*], **Guoyin Wang**[‡], **Changmin Seo**[‡], **Sajal Choudhary**[‡],
**Jiwei Li** [§], **Xiang Li** [‡], **Puyang Xu** [‡], **Sunghyun Park** [‡], **Alice Oh** [†]
[†]KAIST, [‡]Amazon, [§]Zhejiang Univeristy
yeon.seonwoo@kaist.ac.kr
{guoyiwan, changmis, sajalc, lixxiang, puyax, sunghyu}@amazon.com
jiwei_li@zju.edu.cn
alice.oh@kaist.edu

## Abstract

Unsupervised sentence representation learning has progressed through contrastive learning and data augmentation methods such as dropout masking. Despite this progress, sentence encoders are still limited to using only an input sentence when predicting its semantic vector. In this work, we show that the semantic meaning of a sentence is also determined by nearest-neighbor sentences that are similar to the input sentence. Based on this finding, we propose a novel unsupervised sentence encoder, **RankEncoder**. RankEncoder predicts the semantic vector of an input sentence by leveraging its relationship with other sentences in an external corpus, as well as the input sentence itself. We evaluate RankEncoder on semantic textual benchmark datasets. From the experimental results, we verify that 1) RankEncoder achieves 80.07% Spearman's correlation, a 1.1% absolute improvement compared to the previous state-of-the-art performance, 2) RankEncoder is universally applicable to existing unsupervised sentence embedding methods, and 3) RankEncoder is specifically effective for predicting the similarity scores of similar sentence pairs.[1]

## 1 Introduction

Sentence representation learning aims to encode sentences into a semantic vector space. This task has been a fundamental task in natural language processing (NLP), as universal sentence vectors are widely applicable to many NLP tasks (Kiros et al., 2015; Hill et al., 2016; Conneau et al., 2017; Logeswaran and Lee, 2018; Cer et al., 2018; Reimers and Gurevych, 2019). Recently, unsupervised sentence embedding methods have arisen as they have shown a potential to overcome limited labeled data with simple data augmentation methods (Gao et al., 2021; Wang et al., 2022; Yan et al., 2021; Liu et al., 2021; Wu et al., 2021; Izacard et al., 2021; Kim et al., 2021). These approaches minimize the distance between the vector representations of similar sentences, called positive pairs, while maximizing the distance between those of dissimilar sentences, called negative pairs. Many studies have focused on developing better positive and negative pair sampling methods. Data augmentation methods such as dropout masking (Gao et al., 2021), token shuffling (Yan et al., 2021), and sentence negation (Wang et al., 2022) have been proposed and achieved comparable semantic textual similarity performance to sentence encoders trained on human-annotated datasets.

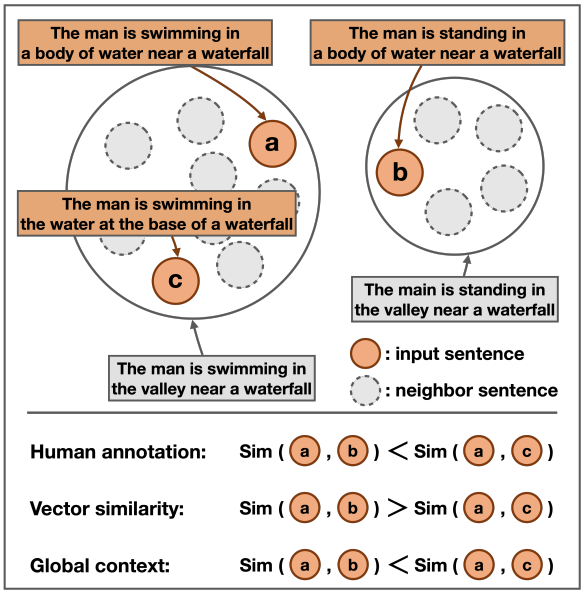



Figure 1: Vector representations of sentences and their neighbor sentences. The neighbor sentences reveal that $(a, c)$ share more semantic meanings than $(a, b)$. This captures more accurate semantic similarity scores than their vectors.

The semantic meaning of a sentence is not only determined by the words within the sentence itself but also by other sentences with similar meanings. However, previous unsupervised sentence embed-

[*]This work was done during an internship at Amazon.
[1]We provide the implementation of RankEncoder at https://github.com/yeonsw/RankEncoder.git



*Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*
*Volume 1: Long Papers*, pages 15783–15798
July 9-14, 2023 

ding methods use only the input sentence when predicting its semantic vector. Figure 1 shows example sentences and their semantic vector space. In this figure, the human-annotated similarity scores indicate that sentence pair $(a, c)$ is more similar than $(a, b)$. However, the similarity scores computed by their sentence vectors indicate the opposite result; the vector representations of $a$ and $b$ are closer than $a$ and $c$ as they have more overlapping words than $a$ and $c$. This problem can be alleviated by leveraging the distance between the input sentence and other sentences in a large corpus. The vectors of their neighbor sentences approximate the overall semantic distribution, and the semantic distribution reveals that sentences $a$ and $c$ are likely to share more semantic meanings than sentences $a$ and $b$. This facilitates the accurate prediction of semantic similarity between sentences.

In this paper, we propose **RankEncoder**, a novel unsupervised sentence encoder that leverages a large number of sentences in an external corpus. For a given corpus with $n$ sentences and an input sentence, RankEncoder computes a rank vector, an $n$-dimensional vector in which $i$'th element represents the distance between the input sentence and $i$'th sentence in the corpus; RankEncoder uses an existing unsupervised sentence encoder, $E$, to compute the distances. Then, two sentences that share the same neighbor sentences (e.g., sentence $a$ and $c$ in Fig 1) have similar rank vector representations. We verify that using the similarity between rank vectors captures better semantic similarity than their vector representation computed by the base encoder, $E$, (Fig 4) without further training. We further leverage the similarity scores predicted by the rank vectors to train another sentence encoder and achieve a better sentence encoder (Table 1).

From experiments on seven STS benchmark datasets, we verify that 1) rank vectors are effective for capturing the semantic similarity of similar sentences, 2) RankEncoder is applicable to any unsupervised sentence encoders, resulting in performance improvement, and 3) this improvement is also valid for the previous state-of-the-art sentence encoder and leads to a new state-of-the-art semantic textual similarity performance. First, we measure the performance of RankEncoder and the baselines on three sentence pair groups divided by their similarity scores. The experimental results show that RankEncoder is effective on similar sentence pairs. Second, we apply RankEncoder to the three unsupervised sentence encoders, SimCSE (Gao et al., 2021), PromptBERT (Jiang et al., 2022), and SNCSE (Wang et al., 2022), then verify that our approach brings performance improvement to each encoder. Third, we apply RankEncoder to the state-of-the-art unsupervised sentence encoder (Wang et al., 2022) and achieve a 1.1% improvement; the previous state-of-the-art is 78.97 Spearman's correlation, and we achieve 80.07 Spearman's correlation.

The contributions of this paper are three folds. First, we demonstrate that the semantic meaning of a sentence is also determined by its nearest-neighbor sentences as well as the words within the sentence itself. Second, we propose RankEncoder, which leverages a large number of sentences to capture the semantic meanings of sentences. Third, we achieve state-of-the-art STS performance and reduce the gap between supervised and unsupervised sentence encoders; the performances of our method and the state-of-the-art supervised sentence encoder (Jiang et al., 2022) are 80.07 and 81.97, respectively.

## 2 Related Works

Unsupervised sentence representation learning has progressed through contrastive learning with positive and negative sentence pair sampling methods (Gao et al., 2021; Jiang et al., 2022; Chuang et al., 2022; Wang et al., 2022). SimCSE (Gao et al., 2021) and ConSERT (Yan et al., 2021) apply data augmentation methods such as dropout masking, token shuffling, and adversarial attacks to an input sentence and sample a positive pair. However, these data augmentation methods often change the meaning of the input sentence and generate dissimilar positive pairs. A masked language modeling-based word replacement method has been proposed to alleviate this problem (Chuang et al., 2022). They train a sentence encoder to predict the replaced words and make the encoder aware of surface-level augmentations. Some studies adopt momentum contrastive learning to generate positive samples inspired by unsupervised visual representation learning (Zhang et al., 2021; Wu et al., 2021). Prompting (Jiang et al., 2022; Jiang and Wang, 2022) is another direction that is capable of generating positive pairs. Recently, a negative sampling method for data augmentation has been proposed (Wang et al., 2022). This approach takes the negation of an

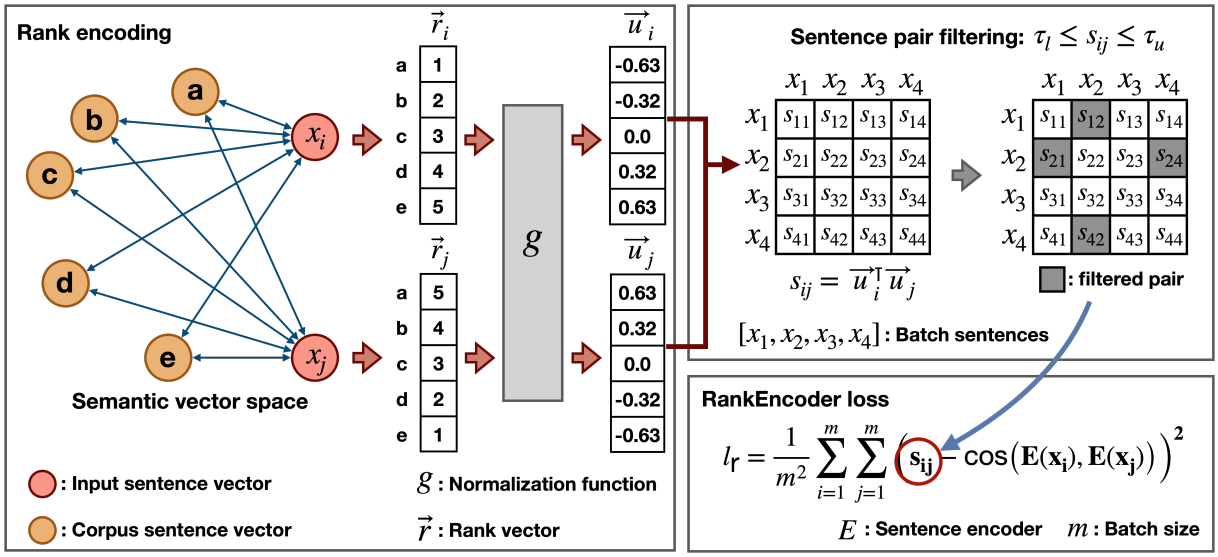



Figure 2: The overall illustration of RankEncoder. The left figure shows the process for computing rank vectors. For a given sentence pair, $(x_i, x_j)$, RankEncoder computes orders of sentences in the corpus by their similarity scores to the input sentence and normalizes these orders with the function $g$; the similarity scores are computed by the base encoder, $E_1$. The right figures show the training process of RankEncoder. For a batch of sentences, RankEncoder computes similarity scores of sentences with their rank vectors and trains the sentence encoder with the mean square error of these scores.

input sentence and uses this soft-negative sample in a contrastive learning framework. Compared to previous approaches focused on generating better positive and negative pairs, our work uses nearest-neighbor sentences to predict better semantic vectors of sentences, a novel approach that previous approaches have yet to cover. Related but different from our work, Trans-Encoder proposes the self-distillation method that gets supervision signals from itself (Liu et al., 2022). Trans-Encoder solves a slightly different problem from ours. They aim to solve an unsupervised sentence pair modeling problem, not unsupervised sentence embedding; although this work does not require any human-annotated similarity scores of sentence pairs for training, they need the sentence pairs of the STS datasets, which are not allowed to be used for training in unsupervised sentence representation learning.

## 3 Method

Leveraging the $k$-nearest-neighbor sentences helps a sentence encoder to approximate a more accurate semantic meaning of a sentence. For instance, when two input sentences have more common neighbors than other sentences, it is likely that they are semantically similar; we have provided an example in Figure 1. We extend this idea to leverage the entire sentences in the corpus, not just the neighbor sentences. Our unsupervised sentence encoder, RankEncoder, computes a rank vector for a given sentence. The rank vector is a list of ranks of all sentences in the corpus computed by their similarity scores to the input; for a given corpus with $n$ number of sentences, a rank vector is an $n$-dimensional vector, in which $i$'th element represents the rank of $i$'th sentence in the corpus. Thus, when two input sentences have common neighbor sentences, their rank vectors are similar. We found that rank vectors capture more accurate semantic similarity than previous unsupervised sentence encoders. Since rank vectors predict better semantic similarity scores between sentences, we use these scores for training another sentence encoder to further increase its STS performance. We provide the overall illustration of RankEncoder in Figure 2.

### 3.1 Contrastive Learning for Base Encoder

The first step of our framework is to learn a base sentence encoder $E_1$ via the standard contrastive learning approach (Chen et al., 2020). Given an input sentence $x_i$, we first create a positive example $x_i^+$ which is semantically similar to $x_i$ (Gao et al., 2021; Chuang et al., 2022); we apply each data augmentation method used by existing unsupervised sentence representation learning studies (Gao et al., 2021; Jiang et al., 2022; Wang et al., 2022) and verify that our approach works in all cases. Then, a text encoder, e.g., BERT (Devlin et al., 2019) and

RoBERTa (Liu et al., 2019), predicts their sentence vectors, $\vec{\boldsymbol{v}}_i$ and $\vec{\boldsymbol{v}}_i^+$. Given a batch of $m$ sentences $\{x_i\}_{i=1}^m$, the contrastive training objective for the sentence $x_i$ with in-batch negative examples is as follows:

$$l_i = -\log \frac{e^{\cos(\vec{\boldsymbol{v}}_i, \vec{\boldsymbol{v}}_i^+)/\tau}}{\sum_{j=1}^m e^{\cos(\vec{\boldsymbol{v}}_i, \vec{\boldsymbol{v}}_j^+)/\tau}}, \quad (1)$$

where $\cos(\cdot)$ is the cosine similarity function and $\tau$ is the temperature hyperparameter. We then get the overall contrastive loss for the whole batch by summing over all the sentences; $l_{cl} = \sum_{i=1}^m l_i$. Note that the training objective $l_{cl}$ can be further enhanced by adding other relevant losses (Chuang et al., 2022), transforming the input sentences (Jiang et al., 2022; Gao et al., 2021), or modifying the standard contrastive loss (Zhou et al., 2022). For simplicity, we use $l_{cl}$ to represent all the variants of contrastive learning loss in this paper. By optimizing $l_{cl}$, we obtain a coarse-grained sentence encoder $E_1$ for the following steps.

### 3.2 RankEncoder

RankEncoder computes the orders of sentences in the corpus with their similarity scores to the input sentence. For a given corpus with $n$ sentences, $\mathcal{C} = [x_1, ..., x_n]$, and a given base encoder, $E_1$, RankEncoder first computes the vector representation of each sentence in the corpus, $\mathcal{V} = [\vec{\boldsymbol{v}}_1, ..., \vec{\boldsymbol{v}}_n]$, with $E_1$. Then computes the rank vector of an input sentence, $x$, by their orders as follows:

$$\text{RankEncoder}_{E_1}(x, \mathcal{V}) = g(< r_1, r_2, ..., r_n >), \quad (2)$$

where $r_i$ is the order of sentence $x_i$. We use cosine similarity scores between $\mathcal{V}$ and the vector representation of $x$, $E_1(x)$. The function $g$ is a normalization function defined as follows:

$$g(\vec{\mathbf{r}}) = \frac{\vec{\mathbf{r}} - \frac{1}{n}\sum_{i=1}^n r_i \cdot \vec{\mathbf{1}}}{\sqrt{n} \times \sigma([r_i]_{i=1}^n)}, \quad (3)$$

where $\sigma$ is the standard deviation of the input values, $\vec{\mathbf{1}}$ is a $n$-dimensional vector of ones. By applying this function to rank vectors, the inner product of two normalized rank vectors becomes equivalent to Spearman's rank correlation, and the similarity is scaled between -1 and 1. We describe the connection between normalization function $g$ and Spearman's rank correlation in Appendix A.1.

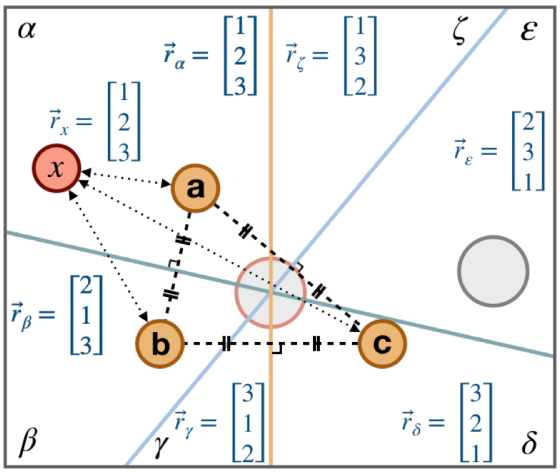



Figure 3: The rank vector space of RankEncoder with corpus $\mathcal{C} = \{a, b, c\}$. The sentence vectors are computed by the base encoder, $E$. Each element of $\vec{\mathbf{r}}$ corresponds to the rank of each sentence within $\mathcal{C}$; the first element in the vector is the rank of sentence $a$. Each line represents the boundary that two rank variables are converted. For instance, all vectors on the left of the yellow line are closer to $b$ than $c$. Sentence vectors in the same area have the same rank vector; the rank vector $\vec{r}_x$ of sentence $x$ is the same as $\vec{r}_\alpha$ as it is in the $\alpha$ area.

### 3.3 Semantic Vector Space of Rank Vectors

The similarity between rank vectors is affected mainly by their neighbor sentences, even though we use the entire sentences in a given corpus. Figure 3 shows a simplified example of RankEncoder's semantic space when the corpus has three sentences. Each solid line represents the boundary that two rank variables are converted. For instance, the yellow line is the boundary that reverses the orders of sentences $b$ and $c$; all the vectors located in the left part of the yellow line are closer to sentence $b$ than $c$. Since we have three sentences in this corpus, we get six rank vectors, and all vectors in each region have the same rank vector. In this figure, we see that the vectors in the red area are more affected by sentences $a$, $b$, and $c$ than vectors in the grey area. For a given sentence, if its sentence representation lies in the central area, i.e., the red area, then its corresponding rank vector can be easily changed by a small modification of its sentence vector. For vectors having a larger distance from these sentences, e.g., the vectors in the gray area, the corresponding rank vectors are much less sensitive to modification of the input's sentence vector. This pattern also holds when we increase the size of the corpus as well; we demonstrate this in Section 5.5.

### 3.4 Model Training

We use similarities predicted by rank vectors to train another sentence encoder, $E_2$[2]; rank vectors capture a better semantic similarity than their vector representation computed by base encoder $E_1$. For a given unsupervised sentence encoder $E_1$ and corpus $\mathcal{C}$, we compute similarity scores of all sentence pairs in a batch with their rank vectors computed from $E_1$. The similarity scores are computed by the inner product of these rank vectors. Then, we define the loss function as the mean square error of RankEncoder's similarity scores as follows:

$$l_{\text{r}} = \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}\Big(\vec{\mathbf{u}}_i^{\intercal}\vec{\mathbf{u}}_j - \cos\big(E_2(x_i), E_2(x_j)\big)\Big)^2, \tag{4}$$

where $\{x_i\}_{i=1}^{m}$ are the sentences in the batch, $E_2$ is the sentence encoder in training, $\vec{\mathbf{u}}_i$ is a rank vector of $x_i$ computed by RankEncoder$_{E_1}$, and $\cos(\cdot)$ is the cosine similarity function. Then, we combine the RankEncoder loss, $l_r$, with the standard contrastive loss, $l_{\text{cl}}$, in the form of the hinge loss as follows:

$$l_{\text{total}} = max(\lambda_{\text{train}} \times l_r, l_{\text{cl}}), \tag{5}$$

where $\lambda_{\text{train}}$ is a weight hyperparameter.

### 3.5 Sentence Pair Filtering

Previous unsupervised sentence encoders randomly sample sentences to construct a batch, and randomly sampled sentence pairs are mostly dissimilar pairs. This causes sentence encoders to learn mostly on dissimilar pairs, which is less important than similar sentence pairs. To alleviate this problem, we filter dissimilar sentence pairs with a similarity under a certain threshold[3]. Also, it is unlikely that randomly sampled sentence pairs have the same semantic meaning. We regard sentence pairs with high similarity as noisy samples and filter these pairs with a certain threshold. The final RankEncoder loss function with sentence pair filtering is as follows:

$$l_r = \sum_{i=1}^{m}\sum_{j=1}^{m}\Bigg\{\frac{\mathbb{1}[\tau_l \le \vec{\mathbf{u}}_i^{\intercal}\vec{\mathbf{u}}_j \le \tau_u]}{\sum_{p=1}^{m}\sum_{q=1}^{m}\mathbb{1}[\tau_l \le \vec{\mathbf{u}}_p^{\intercal}\vec{\mathbf{u}}_q \le \tau_u]} \times \Big(\vec{\mathbf{u}}_i^{\intercal}\vec{\mathbf{u}}_j - \cos\big(E_2(x_i), E_2(x_j)\big)\Big)^2\Bigg\}, \tag{6}$$

where $\tau_l$ and $\tau_u$ are the thresholding parameters, and $\mathbb{1}$ is the indicator function that returns 1 when the condition is true and returns 0 otherwise.

### 3.6 Inference

We can further utilize RankEncoder in inference stage. Given a sentence pair $(x_i, x_j)$, we compute the similarity between the two sentences as follows:

$$\begin{aligned}\text{sim}(x_i, x_j) &= \lambda_{\text{inf}} \cdot \vec{\mathbf{z}}_i^{\intercal}\vec{\mathbf{z}}_j \\ &\quad + (1-\lambda_{\text{inf}}) \cdot \cos\big(E_2(x_i), E_2(x_j)\big),\end{aligned} \tag{7}$$

where $E_2$ is a sentence encoder trained by Eq 5, $\lambda_{\text{inf}}$ is a weight parameter, and $\vec{\mathbf{z}}_i$ and $\vec{\mathbf{z}}_j$ are the rank vectors of $x_i$ and $x_j$ computed by RankEncoder$_{E_2}$.

## 4 Experimental Setup

### 4.1 Base Encoder $E_1$ & Corpus $\mathcal{C}$

RankEncoder computes rank vectors using corpus $\mathcal{C}$ and base encoder $E_1$. We use 100,000 sentences sampled from Wikipedia[4] as the corpus $\mathcal{C}$[5], and we use the following unsupervised sentence encoders for $E_1$, SimCSE (Gao et al., 2021), PromptBERT (Jiang et al., 2022), and SNCSE (Wang et al., 2022). SimCSE is a standard unsupervised sentence encoder that uses a standard contrastive learning loss with the simple data augmentation method. We use SimCSE as it is effective to show the efficacy of RankEncoder. We use PromptBERT and SNCSE, the state-of-the-art unsupervised sentence encoders, to verify whether RankEncoder is effective on more complex models.

[2]It is also possible to train $E_1$ continuously with the rank vector similarities. However, this approach yields slightly lower performance than training another sentence encoder, $E_2$.

[3]This method increases RankEncoder's STS performance by 0.17% Spearman correlation, resulting in a performance of 80.07 Spearman correlation shown in Table 1.

[4]We use Wikipedia sentences since they are generally effective on STS datasets. However, for optimal results, it is best to use a corpus that closely aligns with the specific domain of the inputs.

[5]The performance of RankEncoder increases as the number of sentences in the corpus increases, and the performance converges at a size of 10,000. There is a slight but negligible improvement beyond this point. We sample more than 10,000 sentences to push the performance boundary as much as possible.

| Model | STS12 | STS13 | STS14 | STS15 | STS16 | STS-B | SICK-R | AVG |
|---|---|---|---|---|---|---|---|---|
| ConSERT (Yan et al., 2021) | 64.64 | 78.49 | 69.07 | 79.72 | 75.95 | 73.97 | 67.31 | 72.74 |
| SimCSE (Gao et al., 2021) | 68.40 | 82.41 | 74.38 | 80.91 | 78.56 | 76.85 | 72.23 | 76.25 |
| DCLR (Zhou et al., 2022) | 70.81 | 83.73 | 75.11 | 82.56 | 78.44 | 78.31 | 71.59 | 77.22 |
| ESimCSE (Wu et al., 2021) | 73.40 | 83.27 | 77.25 | 82.66 | 78.81 | 80.17 | 72.30 | 78.27 |
| DiffCSE (Chuang et al., 2022) | 72.28 | 84.43 | 76.47 | 83.90 | 80.54 | 80.59 | 71.23 | 78.49 |
| PromptBERT (Jiang et al., 2022) | 71.56 | 84.58 | 76.98 | **84.47** | **80.60** | **81.60** | 69.87 | 78.54 |
| SNCSE (Wang et al., 2022) | 70.67 | 84.79 | 76.99 | 83.69 | 80.51 | 81.35 | 74.77 | 78.97 |
| RankEncoder | **74.88** | **85.59** | **78.61** | 83.50 | 80.56 | 81.55 | **75.78** | **80.07** |

Table 1: Semantic textual similarity performance of RankEncoder and baselines in an unsupervised setting. Following previous sentence embedding studies, we measure the Spearman's rank correlation between the human annotated scores and the model's predictions. The results of the baselines are from the original paper. RankEncoder uses SNCSE as base encoder $E_1$.

### 4.2 Datasets & Evaluation Metric

We evaluate RankEncoder on seven semantic textual similarity benchmark datasets: STS2012-2016 (Agirre et al., 2012, 2013, 2014, 2015, 2016), STS-B (Cer et al., 2017), and SICK-Relatedness (Marelli et al., 2014). Each dataset consists of sentence pairs with human-annotated similarity scores. For each sentence pair, sentence encoders predict the similarity, and we measure the Spearman's rank correlation between the predicted similarities and the human-annotated similarities.

### 4.3 Training Details & Hyper-Parameter Settings

We train RankEncoder on $10^6$ sentences from Wikipedia, following existing unsupervised sentence embedding studies. We use two NVIDIA V100 GPUs for training. The running time for training RankEncoder is approximately 1.5 hours, which takes an hour more than the training time of SimCSE, and its inference takes slightly more time, about 1.8% more than SimCSE based on BERT-base. We provide the details in Appendix A.7. We find the best hyperparameter setting on the development sets of the STS-B and SICKRelatedness datasets. We set $\lambda_{\text{train}} = 0.05$, $\lambda_{\text{inf}} = 0.1$, $\tau_l = 0.5$, and $\tau_u = 0.8$. We provide more analysis on the hyper-parameter, $\lambda_{\text{train}}$, in Appendix A.2. For other hyperparameters, we follow the base encoder's setting provided by the authors of each base encoder, $E_1$.

## 5 Results and Discussions

In this section, we demonstrate that 1) RankEncoder is effective for capturing the semantic similarity scores of similar sentences, 2) RankEncoder is universally applicable to existing unsupervised sentence encoders, and 3) RankEncoder achieves state-of-the-art semantic textual similarity (STS) performance. We describe the detailed experimental results in the following sections.

### 5.1 Semantic Textual Similarity Performance

We apply RankEncoder to an existing unsupervised sentence encoder and achieve state-of-the-art STS performance. We use SNCSE (Wang et al., 2022) fine-tuned on BERT-base (Devlin et al., 2019) as the base encoder, $E_1$. Table 1 shows the STS performance of RankEncoder and unsupervised sentence encoders on seven STS datasets and their average performance (AVG). RankEncoder increases the AVG performance of SNCSE by 1.1 and achieves state-of-the-art STS performance.

RankEncoder brings a significant performance gain on STS12, STS13, STS14, and SICK-R, but a comparably small improvement on STS16 and STS-B. We conjecture that this is because RankEncoder is specifically effective on similar sentence pairs. The STS12, 13, 14, and SICK-R datasets contain similar sentence pairs more than dissimilar pairs; we show the similarity distribution of each dataset in Appendix A.3. This pattern is aligned with the performance gain on each STS dataset in Table 1.

### 5.2 Universality of RankEncoder

RankEncoder applies to any unsupervised sentence encoders. We apply RankEncoder to SimCSE (Gao et al., 2021), PromptBERT (Jiang et al., 2022), and SNCSE (Wang et al., 2022). SimCSE represents the vanilla contrastive learning-based sentence encoder, and PromptBERT and SNCSE represent the state-of-the-art unsupervised sentence encoders. We evaluate each encoder's average per-

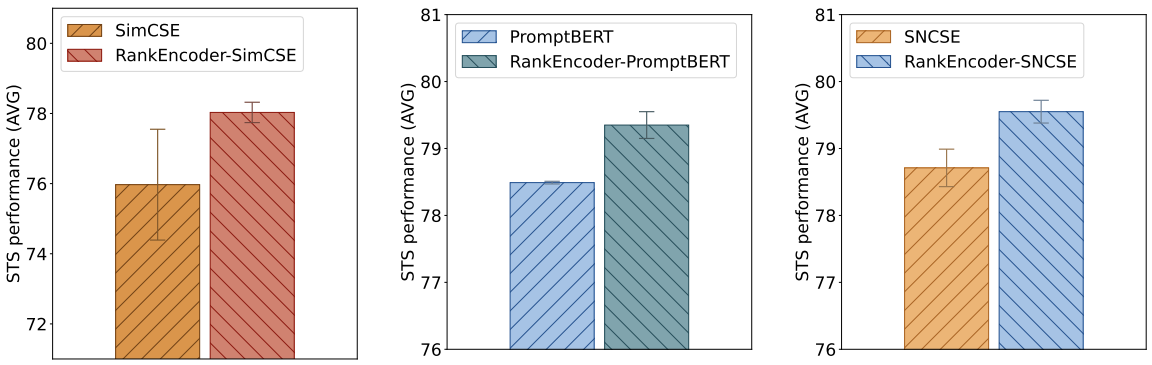



Figure 4: STS performance of three unsupervised sentence encoders and RankEncoder. We report the mean performance and standard deviation of three separate trials with different random seeds. RankEncoder brings improvement on all base encoders. This result implies that our approach is generally applicable to other unsupervised sentence embedding approaches.

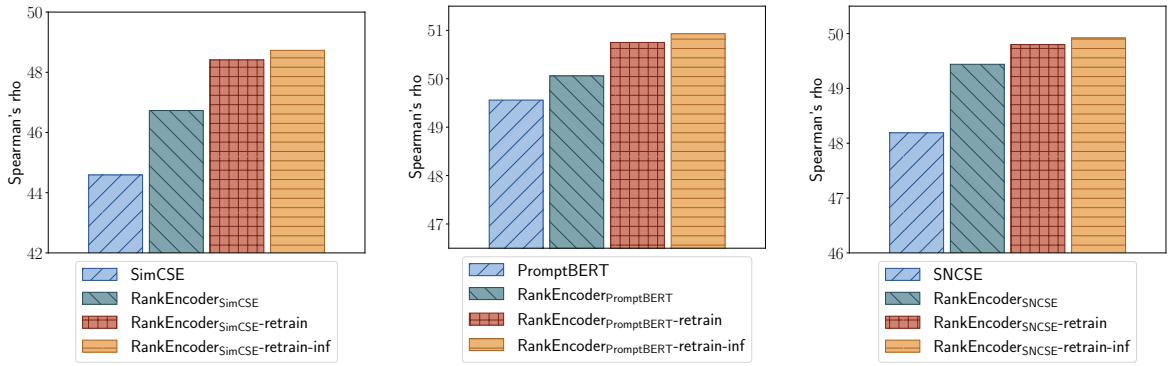



Figure 5: STS performance of three unsupervised sentence encoders and RankEncoder on sentence pairs with high similarity scores; we select sentence pairs with a similarity between 0.67 and 1.0 (0.0-1.0 scale) in the STS-B dataset. We ablate the two components of RankEncoder, re-training (Eq. 5) and inference (Eq. 7).

formance (AVG) on seven STS datasets. We train each encoder in three separate trials and report the mean and the standard deviation of the AVG performances in Figure 4; the error bar shows the standard deviation. This figure shows that RankEncoder increases the average STS performance on each unsupervised sentence encoder; the improvements on SimCSE, PromptBERT, and SNCSE are 2.1, 0.9, and 0.9, respectively. We report detailed experimental results in Appendix A.5. This result implies that RankEncoder is a universal method that applies to any unsupervised sentence encoder.

### 5.3 Overlapping Neighbor Sentences

In Section 3.3, we conjecture that the RankEncoder is specifically effective for similar sentence pairs as they have more overlapping neighbor sentences, which are used to approximate their semantic similarity. To support this supposition, we show the relation between the performance gain caused by RankEncoder and the number of overlapping neighbor sentences of the input sentences. We group sentence pairs in the STS-B dataset by cosine similarity scores of their sentence vectors, then compare the STS performance of SimCSE and RankEncoder (Eq. 2 without re-training) on each group; we use SimCSE as the base encoder, $E_1$. We also report the average number of overlapping neighbor sentences of each group; for each sentence pair, we count the number of sentences in the intersection of their top 100 nearest neighbor sentences and take the average. Figure 6 shows one expected result of our supposition; the performance gain correlates with the number of overlapping neighbor sentences.

### 5.4 Performance on similar sentence pairs

It is more important to accurately predict the similarities between similar texts than those between dissimilar ones. This is because many NLP down-

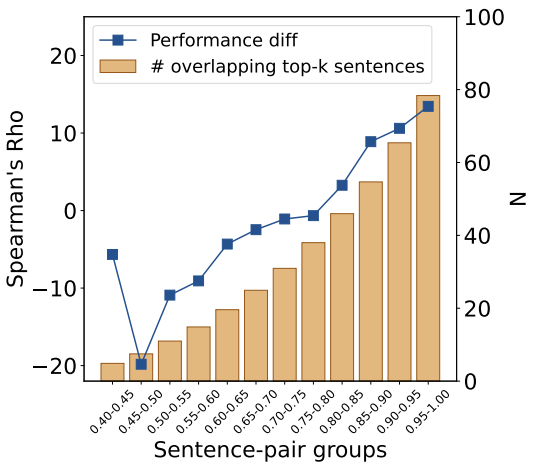
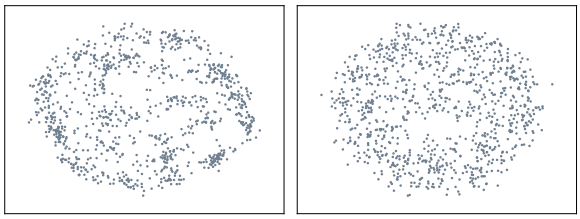



Figure 6: The performance difference between SimCSE and RankEncoder (blue line) and the number of overlapping neighbor sentences (yellow bar) on each sentence pair group. We group sentence pairs in the STS-B dataset based on the cosine similarity of their vector representations computed by SimCSE. The number of overlapping neighbor sentences is the number of sentences in the intersection of each sentence's top 100 neighbor sentences.



(a) PromptBERT (b) RankEncoder

Figure 7: Semantic vector spaces of PromptBERT and RankEncoder. We randomly sample 1000 sentences from the STS-B dataset and visualize the vector representations of these sentences (grey dots). We use PromptBERT as the base encoder of RankEncoder. We use the following equation to compute the distances between vectors; $\text{dist}(\vec{\mathbf{v}}_i, \vec{\mathbf{v}}_\mathbf{j}) = 1 - \cos(\vec{\mathbf{v}}_i, \vec{\mathbf{v}}_\mathbf{j})$.

stream tasks, e.g., retrieval and reranking, aim to find the most relevant/similar text (or texts) from candidate texts, and we can easily filter out dissimilar texts with simple approaches such as a lexical retriever; we only need rough similarity scores to identify the dissimilar texts. In this section, we demonstrate the efficacy of our approach on similar sentence pairs. We divide sentence pairs in the STS-B dataset into three groups by their human-annotated similarity scores and use the group with the highest similarity. The similarity range of each group is 0.0-0.33 for the dissimilar groups, 0.33-0.67 for the intermediate group, and 0.67-1.0 for the similar group; we normalize the scores to a 0.0-1.0 scale. Figure 5 shows the performance of three unsupervised sentence encoders and the performance gain brought by each component of RankEncoder. RankEncoder$_E$ is the model with Eq. 2 that uses $E$ as the base encoder. RankEncoder$_E$-retrain is the model with re-training (Eq. 5). RankEncoder$_E$-retrain-inf is the model with re-training and weighted inference (Eq. 7). From the comparison between $E$ and RankEncoder$_E$, we verify that rank vectors effectively increase the base encoder's performance on similar sentence pairs. This improvement is even more significant when using rank vectors for retraining and inference. We report the detailed results in Appendix A.6.

### 5.5 The Vector Space of RankEncoder

In Section 3.3, we show that rank vectors become more distinguishable as the number of sharing neighbor sentences increases. In this section, we demonstrate that this pattern holds for a larger corpus as well. Figure 7 shows the vector representations of randomly sampled 1,000 sentences in the STS-B dataset; Figure 7a is the vector space of PromptBERT, and Figure 7b is the rank vector space. We see that the dense sub-spaces in Figure 7a expand as shown in 7b, and their representations become more distinguishable.

Rank vectors improve the uniformity of the semantic vector space with negligible degradation in alignment. Uniformity and alignment are metrics for measuring the quality of embedding vectors (Gao et al., 2021; Wang and Isola, 2020). Uniformity is a measure of the degree of evenness of the embedding vectors. Alignment is a measure of the degree of closeness of the embedding vectors of positive pairs (e.g., sentence pairs with a similarity score higher than 4.0 in the STS-B dataset). We show the uniformity and alignment of each base encoder, $E_1$, and their corresponding rank vectors, RankEncoder$_{E_1}$, in Table 2. For each base encoder, their rank vectors largely improve uniformity, which is aligned with the results shown in Figure 7. These results also show that rank vectors bring degradation in alignment. However, this degradation is relatively negligible compared to the improvement in uniformity. From these results, we conjecture that the performance improvement on the STS benchmark datasets shown in Figure 4 is mostly related to the improvement in uniformity

| | Uniformity | Alignment |
|---|---|---|
| SimCSE | -2.42 | 0.21 |
| +RankEncoder | -3.23 | 0.23 |
| PromptBERT | -1.49 | 0.11 |
| +RankEncoder | -3.31 | 0.22 |
| SNCSE | -2.21 | 0.16 |
| +RankEncoder | -3.20 | 0.21 |

Table 2: Uniformity and alignment of base encoders and RankEncoder. Lower is better.

rather than alignment.

## 6 Conclusion

In this study, we showed that the semantics of a sentence is also determined by its similar sentence, not just the words within the sentence itself. We proposed RankEncoder to overcome the limitation of the previous sentence representation learning approaches, which are limited to using only the input sentence. RankEncoder leverages the distance between the input sentence and the sentences in a corpus to predict its semantic vector. RankEncoder is universally applicable to any unsupervised sentence encoder, resulting in performance improvement, and we demonstrated this with three unsupervised sentence encoders. We achieved state-of-the-art semantic textual similarity performance by applying our approach to the previous best sentence encoder. We also showed that our approach is specifically effective for capturing the semantic similarities of similar sentences.

## 7 Limitations

This work has been studied on the Wikipedia corpus, following the standard experimental setting used in previous unsupervised sentence representation learning studies. We expect to see many important findings by investigating sentence representation learning on various corpora in different domains such as Bookcorpus (Zhu et al., 2015) and the C4 corpus (Raffel et al., 2019).

## Acknowledgements

This research was supported by the Engineering Research Center Program through the National Research Foundation of Korea (NRF) funded by the Korean Government MSIT (NRF-2018R1A5A1059921)

## References

Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Inigo Lopez-Gazpio, Montse Maritxalar, Rada Mihalcea, et al. 2015. Semeval-2015 task 2: Semantic textual similarity, english, spanish and pilot on interpretability. In *SemEval*.

Eneko Agirre, Carmen Banea, Claire Cardie, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Weiwei Guo, Rada Mihalcea, German Rigau, and Janyce Wiebe. 2014. SemEval-2014 task 10: Multilingual semantic textual similarity. In *SemEval*.

Eneko Agirre, Carmen Banea, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, Rada Mihalcea, German Rigau, and Janyce Wiebe. 2016. SemEval-2016 task 1: Semantic textual similarity, monolingual and cross-lingual evaluation. In *SemEval*.

Eneko Agirre, Daniel Cer, Mona Diab, and Aitor Gonzalez-Agirre. 2012. Semeval-2012 task 6: A pilot on semantic textual similarity. In *SemEval*.

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. * sem 2013 shared task: Semantic textual similarity. In *SemEval*.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *SemEval*.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv*.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *ICML*.

Yung-Sung Chuang, Rumen Dangovski, Hongyin Luo, Yang Zhang, Shiyu Chang, Marin Soljačić, Shang-Wen Li, Wen-tau Yih, Yoon Kim, and James Glass. 2022. Diffcse: Difference-based contrastive learning for sentence embeddings. *arXiv*.

Alexis Conneau and Douwe Kiela. 2018. Senteval: An evaluation toolkit for universal sentence representations. *arXiv*.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *EMNLP*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing*.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *EMNLP*.

Felix Hill, Kyunghyun Cho, and Anna Korhonen. 2016. Learning distributed representations of sentences from unlabelled data. In *NAACL*.

Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *KDD*.

Gautier Izacard, Mathild Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning. *arXiv*.

Ting Jiang, Shaohan Huang, Zihan Zhang, Deqing Wang, Fuzhen Zhuang, Furu Wei, Haizhen Huang, Liangjie Zhang, and Qi Zhang. 2022. Promptbert: Improving bert sentence embeddings with prompts. *arXiv*.

Yuxin Jiang and Wei Wang. 2022. Deep continuous prompt for contrastive learning of sentence embeddings. *arXiv*.

Taeuk Kim, Kang Min Yoo, and Sang-goo Lee. 2021. Self-guided contrastive learning for bert sentence representations. In *ACL*.

Ryan Kiros, Yukun Zhu, Russ R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. *NeurIPS*.

Fangyu Liu, Yunlong Jiao, Jordan Massiah, Emine Yilmaz, and Serhii Havrylov. 2022. Trans-encoder: Unsupervised sentence-pair modelling through self- and mutual-distillations. In *ICLR*.

Fangyu Liu, Ivan Vulić, Anna Korhonen, and Nigel Collier. 2021. Fast, effective, and self-supervised: Transforming masked language models into universal lexical and sentence encoders. In *EMNLP*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv*.

Lajanugen Logeswaran and Honglak Lee. 2018. An efficient framework for learning sentence representations. In *ICLR*.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. 2014. A SICK cure for the evaluation of compositional distributional semantic models. In *LREC*.

Bo Pang and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *ACL*.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *ACL*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP-IJCNLP*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

Ellen M Voorhees and Dawn M Tice. 2000. Building a question answering test collection. In *SIGIR*.

Hao Wang, Yangguang Li, Zhen Huang, Yong Dou, Lingpeng Kong, and Jing Shao. 2022. Sncse: Contrastive learning for unsupervised sentence embedding with soft negative samples. *arXiv*.

Tongzhou Wang and Phillip Isola. 2020. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *ICML*.

Janyce Wiebe, Theresa Wilson, and Claire Cardie. 2005. Annotating expressions of opinions and emotions in language. *Language resources and evaluation*.

Xing Wu, Chaochen Gao, Liangjun Zang, Jizhong Han, Zhongyuan Wang, and Songlin Hu. 2021. Esimcse: Enhanced sample building method for contrastive learning of unsupervised sentence embedding. *arXiv*.

Yuanmeng Yan, Rumei Li, Sirui Wang, Fuzheng Zhang, Wei Wu, and Weiran Xu. 2021. Consert: A contrastive framework for self-supervised sentence representation transfer. In *ACL*.

Yan Zhang, Ruidan He, Zuozhu Liu, Lidong Bing, and Haizhou Li. 2021. Bootstrapped unsupervised sentence representation learning. In *ACL*.

Kun Zhou, Beichen Zhang, Xin Zhao, and Ji-Rong Wen. 2022. Debiased contrastive learning of unsupervised sentence representations. In *ACL*.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *ICCV*.

# A Appendix

## A.1 The connection between the normalization function $g$ and Spearman's Rank Correlation

The Spearman's rank correlation of two lists of variables, $u =< u_1, ..., u_n >$ and $v =< v_1, ..., v_n >$,

is the Pearson correlation coefficient, $\rho$, of their ranks, $r^u$ and $r^v$, as follows:

$$\rho(r^u, r^v) = \frac{\sum_{i=1}^{n} \left\{ \frac{1}{n}(r_i^u - \overline{r}^u) \times (r_i^v - \overline{r}^v) \right\}}{\sigma(r^u) \times \sigma(r^v)}, \tag{8}$$

where $\overline{r}^u$ and $\overline{r}^v$ are the mean of the rank variables, $\sigma(r)^u$ and $\sigma(r)^v$ are the standard deviations of ranks. Then, this can be re-written as follows:

$$\rho(r^u, r^v) = \\ \Big(\frac{1}{\sqrt{n}}(r^u - \overline{r}^u)/\sigma(r^u)\Big)^\intercal \Big(\frac{1}{\sqrt{n}}(r^v - \overline{r}^v)/\sigma(r^v)\Big). \tag{9}$$

Thus, the inner product of the two rank vectors after normalization with g is equivalent to the Spearman's rank correlation of the rank variables.

### A.2 $\lambda_{\text{train}}$ Analysis

The RankEncoder loss, $l_r$, brings a large effect to RankEncoder's re-training process even when the weight parameter, $\lambda_{\text{train}}$, is set to a small value. In this section, we show that the two losses, $l_{\text{cl}}$ and $l_r$, similarly affect to the total loss, $l_{\text{total}}$ in Eq. 5, when $\lambda_{\text{train}} = 0.05$, which is the default setting we use for all experiments in this paper. Figure 8 shows the training loss curves of RankEncoder and SimCSE-unsup with the same random seed. We show the two losses, $l_{\text{cl}}$ and $l_r$, of RankEncoder separately. SimCSE-unsup's loss rapidly decreases at the beginning, and converges to a value less than 0.001. We see a similar pattern in the contrastive loss of RankEncoder, which is the same loss function as SimCSE-unsup. In contrast, $\lambda_{\text{train}} \times l_r$ starts from a much lower value than $l_{\text{cl}}$; even without the weight parameter, $l_r$ is still much lower than $l_{\text{cl}}$. After few training steps, $\lambda_{\text{train}} \times l_r$ converges close to the value of $l_{\text{cl}}$. Given that $\lambda_{\text{train}}$ determines the scale of two losses of our hinge loss function (Eq. 5), we expect that increasing $\lambda_{\text{train}}$ brings RankEncoder's loss curve converged to higher than SimCSE's loss. This result shows that $\lambda_{\text{train}} = 0.05$ is optimal value that maintaining the RankEncoder's loss curve similar to the base encoder's loss curve, while balancing the weights of the two losses, $l_{\text{cl}}$ and $l_r$.

The loss curve of a supervised sentence encoder provides a reference point for comparison between the loss curves of unsupervised sentence encoders. In Figure 8, all unsupervised sentence encoders' loss curves show a rapidly decreasing pattern, which implies overfitting in training. To verify whether this pattern comes from unsupervised training, we show the loss curve of the supervised sentence encoder, SimCSE-sup, in Figure 8. In this experiment, we measure the same contrastive loss used in unsupervised sentence encoders but in the SimCSE-sup's fully supervised training process. We see the same pattern also holds for SimCSE-sup and verify that the rapidly decreasing pattern is not the problem that only occurs in unsupervised training.

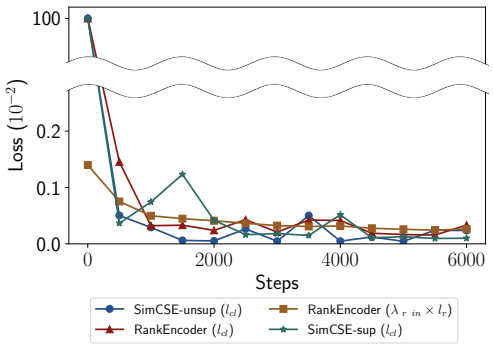



Figure 8: The training loss curves of SimCSE and RankEncoder. X-axis represents a training step, and Y-axis is a scaled loss. After few training steps, the three losses converge in a similar value. Setting $\lambda_{\text{train}}$ to a small value, 0.05, results in similar weights on the two loss functions of RankEncoder while maintaining the loss curve of the base encoder.

### A.3 Similarity Distribution of STS Benchmark Datasets

Semantic textual similarity datasets have different similarity distributions. Since RankEncoder is specifically effective for similar sentence pairs, we expect that RankEncoder brings a more performance increase on datasets with more similar sentence pairs. We show the similarity distribution of each STS dataset in Figure 9. In this figure, we normalize the similarity scores between 0 and 1. The result shows that the similarity distributions of STS12, STS14, and SICK-R are skewed to a high similarity score and STS13's similarity distribution has a distinct peak at a high similarity score. From the results in Table 1, we see that RankEncoder is more effective on STS12, STS13, STS14, SICK-R, and show the relation between the performance increase and the similarity distribution of

| Model | MR | CR | SUBJ | MPQA | SST | TREC | MRPC | AVG |
|---|---|---|---|---|---|---|---|---|
| SimCSE | 81.18 | 86.46 | 94.45 | 88.88 | 85.50 | 89.80 | 74.43 | 85.81 |
| SimCSE w/ MLM | **82.92** | 87.23 | **95.71** | 88.73 | **86.81** | 87.01 | **78.07** | 86.64 |
| RankEncoder-SimCSE w/ MLM | 82.14 | **87.31** | 95.35 | **89.05** | 86.66 | **91.00** | 76.06 | **86.80** |

Table 3: Transfer task results of baselines and RankEncoder. We use RankEncoder with base encoder SimCSE. MLM represents that the model is trained by both loss functions: its loss function and the masked language modeling loss used in pre-trained language models such as BERT. We set the weight parameter of the MLM loss function to 0.1.

| Model | STS12 | STS13 | STS14 | STS15 | STS16 | STS-B | SICK-R | AVG |
|---|---|---|---|---|---|---|---|---|
| SimCSE | 68.1±3.3 | 81.4±1.6 | 73.8±2.4 | 81.8±1.4 | 78.3±0.6 | 77.3±2.3 | 71.0±0.4 | 76.0±1.5 |
| + RankEncoder | **75.0**±0.6 | **82.0**±0.7 | **75.2**±0.2 | **83.0**±0.1 | **79.8**±0.1 | **80.4**±0.6 | **71.1**±1.2 | **78.1**±0.1 |
| PromptBERT | 72.1±0.2 | 84.6±0.3 | 76.8±0.1 | 84.2±0.3 | 80.4±0.3 | 81.8±0.3 | 69.5±0.2 | 78.5±0.0 |
| + RankEncoder | **74.2**±0.3 | **85.2**±0.2 | **77.7**±0.2 | **84.4**±0.3 | **80.7**±0.5 | **82.1**±0.4 | **71.2**±0.2 | **79.4**±0.2 |
| SNCSE | 70.2±0.5 | 84.1±0.5 | 77.1±0.4 | **83.2**±0.5 | 80.7±0.1 | 80.7±0.6 | 75.0±0.1 | 78.7±0.3 |
| + RankEncoder | **73.9**±0.6 | **84.5**±0.5 | **78.0**±0.3 | 83.0±0.5 | **81.0**±0.2 | **81.2**±0.2 | **75.3**±0.1 | **79.6**±0.2 |

Table 4: Semantic textual similarity performance of sentence encoders. We measure Spearman's rank correlation between the human-annotated scores and the model's predictions. We report the mean performance and standard deviation of three separate trials with different random seeds.

each dataset.

### A.4 Transfer Tasks

We verify that applying our approach to an existing unsupervised sentence encoder increases the performance on transfer tasks. We use the following seven transfer tasks to evaluate sentence embeddings: MR (Pang and Lee, 2005), CR (Hu and Liu, 2004), SUBJ (Pang and Lee, 2004), MPQA (Wiebe et al., 2005), SST (Socher et al., 2013), TREC (Voorhees and Tice, 2000), and MRPC (Dolan and Brockett, 2005). These transfer tasks employ an additional single-layer neural network to transform sentence embeddings into the appropriate output format for a given task. The single-layer neural network is trained with the training set of each task. We use the SentEval toolkit (Conneau and Kiela, 2018) for evaluation. Table 3 shows the performance of SimCSE and RankEncoder on these transfer tasks. We use SimCSE as the base encoder of RankEncoder. Recently, SimCSE (Gao et al., 2021) has shown that training sentence encoders with auxiliary masked language modeling (MLM) loss enhances their performance on transfer tasks. Inspired by this finding, we use MLM loss when training RankEncoder. The experimental results show that our approach increases the average performance on transfer tasks by 0.16%p. This performance gain is relatively small when we compare it with the performance gain on STS benchmark datasets shown in Table 1. This is because the sentence embedding quality is not directly connected to the objective of transfer tasks (Gao et al., 2021).

### A.5 Universality of RankEncoder

In this section, we report the detailed experimental results of Figure 4. Table 4 shows the results.

### A.6 The performance of RankEncoder on Similar Sentence Pairs

We report the detailed results of Figure 5 in Table 5.

### A.7 Computational Cost

In this section, we describe the details of the computational efficiency of RankEncoder.

**Pre-Computation:** We pre-compute sentence vectors of corpus $\mathcal{C}$ for training and inference. This takes a few seconds on a single V100 GPU.

**Training:** Most of the additional training time comes from calculating a rank vector similarity matrix (the matrix in Figure 2). First, we calculate a rank vector for every sentence in a given batch. The time complexity of calculating a rank vector is

| | Base Encoder $E_1$ | | |
|---|---|---|---|
| | SimCSE | PromptBERT | SNCSE |
| $E_1$ | 44.59 | 49.56 | 48.19 |
| RankEncoder$_{E_1}$ | 46.73 | 50.06 | 49.44 |
| RankEncoder$_{E_1}$ − retrain | 48.41 | 50.75 | 49.80 |
| RankEncoder$_{E_1}$ − retrain − inf | 48.73 | 50.93 | 49.92 |

Table 5: Semantic textual similarity performance of variations of RankEncoder. $E$ is the base encoder. RankEncoder$_E$ is RankEncoder with Eq. 2. RankEncoder − retrain is RankEncoder with Eq. 5. RankEncoder − retrain − inf is RankEncoder with Eq. 7.

$O(N \times D)$, where $N$ is the number of pre-indexed vectors, and $D$ is the dimension of the vectors. Assuming a batch size of $B$, the time complexity of this step is $O(B \times N \times D)$. Second, we calculate a $B \times B$ rank vector similarity matrix. This is $O(B \times B \times N)$ since the dimension size of the rank vector is $N$. The total time complexity is $O(B \times D \times N + B \times B \times N)$, which is $O(B \times D \times N)$, assuming the batch size is much smaller than the dimension size. As a result, the total training time is 1.5 hours, 0.5 hours (the base encoder's training time) + 1.0 hours (the additional training time brought by our approach).

**Inference:** RankEncoder's inference process comprises two steps: 1) predicting the sentence vector of an input sentence and 2) computing similarity scores between the input sentence vector and the pre-indexed vectors; we exclude the indexing time since indexing is completed before inference. The first step takes the same inference time as BERT-base (0.07 seconds for a given sentence on a single V100 GPU) as RankEncoder uses BERT-base. The second step entails matrix multiplication of an $N \times D$ matrix and a $D \times 1$ matrix (this takes 0.0013 seconds), which takes 1.8% of the whole inference time. Thus, our method increases the inference time by 1.8%.

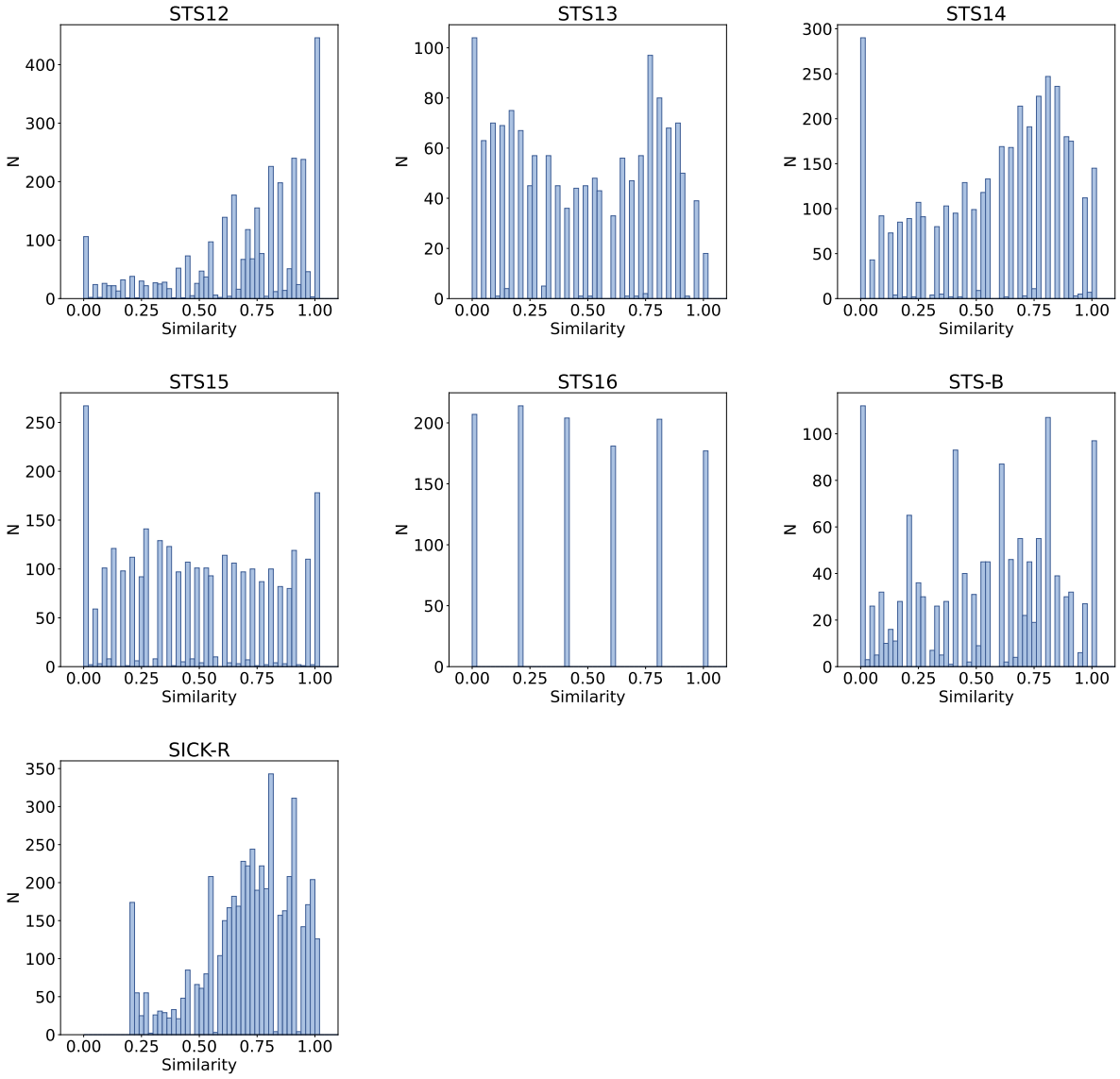



Figure 9: Similarity distributions of semantic textual similarity benchmark datasets. We scale the similarity scores between 0.0 and 1.0.

## ACL 2023 Responsible NLP Checklist

### A For every submission:

☑ A1. Did you describe the limitations of your work?
*Section 7*

☐ A2. Did you discuss any potential risks of your work?
*Not applicable. Left blank.*

☑ A3. Do the abstract and introduction summarize the paper's main claims?
*Abstract, Section 1*

☒ A4. Have you used AI writing assistants when working on this paper?
*Left blank.*

### B ☑ Did you use or create scientific artifacts?

*Section 3, Section 4*

☑ B1. Did you cite the creators of artifacts you used?
*Section 3, Section 4*

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
*Section 3, Section 4*

☑ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
*Section 3, Section 4*

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
*Not applicable. Left blank.*

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
*Not applicable. Left blank.*

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
*Section 4*

### C ☑ Did you run computational experiments?

*Section 5*

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
*Section 4, Section 5*

*The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.*

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
*Section 4*

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
*Section 5*

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
*Section 4*

**D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?**

*Left blank.*

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
*No response.*

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
*No response.*

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
*No response.*

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
*No response.*

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
*No response.*