Adjusting Dropout in Contrastive Learning of Sentence Embeddings

Stanford CS224N Default Project

Guillermo Frontera Sánchez
Department of Computer Science
Stanford University
frontera@stanford.edu

Maurice Georgi
Department of Computer Science
Stanford University
maugeo@stanford.edu

Abstract

In our project, we implement the SimCSE and DiffCSE frameworks to improve sentence embeddings for multiple downstream tasks. Also, we analyse if we can improve the performance of SimCSE and DiffCSE by increasing dropout probabilities in early blocks of the BERT model, in order to increase the 'dropout-as-data-augmentation' effect of those frameworks. We are able to increase the performance of our network by 10% compared to our baseline and demonstrate the crucial role of preserving pre-trained features.

1 Introduction

In recent years, there have been many efforts devoted to learning universal sentence embeddings [1]. Pre-training models that capture rich semantic information has proven to perform very well in many NLP tasks by just adding one final output layer. Transformer architectures [2] like BERT [3] have made significant progress in providing useful sentence embeddings without a need for labeled text. As mentioned in [4], self-supervised learning (SSL) is one of the key ingredients in building transformer-based pre-trained language models. Several SSL methods have been developed to improve sentence embeddings, such as contrastive SSL. Contrastive SSL helps the model to learn by comparing semantically related and unrelated sentences. Obtaining labeled data for those comparisons is costly, whereas unlabeled data for NLP is available on large scales. Therefore, contrastive learning often relies on unsupervised methods, using data augmentation to generate the model input. However, most contrastive learning methods use discrete data augmentation (e.g., replacing one word for another) and changing the input data in NLP can easily change the meaning.

In this project, we further pre-train a baseline BERT [3] model (which has already been pre-trained using a large corpus) with SimCSE[1] [5] and DiffCSE [6] to improve sentence embeddings for three downstream tasks, namely sentiment analysis, paraphrase detection and semantic textual similarity. SimCSE is an unsupervised approach that takes an input sentence and predicts itself in a contrastive objective, with only standard dropout used as noise. DiffCSE builds on top of SimCSE and additionally learns sentence embeddings that are sensitive to the difference between the original sentence and an edited sentence. We specifically chose SimCSE and DiffCSE because both algorithms claim to improve sentence embeddings in general, i.e., they improve alignment and uniformity (alignment measures whether semantically similar sentence pairs are close together and uniformity measures how uniform the sentence embeddings are distributed). These further pre-trained networks are used as initialization for fine-tuning on all downstream tasks. In theory, a well-initialized model can leverage the knowledge incorporated by the pre-training to improve performance on each downstream task.

Also, we explore whether we can further improve SimCSE and DiffCSE by adjusting the dropout masks. In both papers, dropout is viewed as a form of data augmentation when computing contrastive loss. Normally, data augmentation is applied at the input level, i.e., the input is changed but nothing in the network changes. In this case, the dropout layers create random perturbations in every Transformer

[1] If not stated otherwise, we refer to unsupervised SimCSE when citing SimCSE in this report.

Stanford CS224N Natural Language Processing with Deep Learning

block, producing different sentence embeddings even for the same input sentence. The authors in both papers carefully searched for the best dropout probability hyperparameter, but applied the same value throughout the model. We apply different dropout probabilities for each block depth, with higher values at the earliest blocks and lower values at the final blocks, so that the data augmentation caused by dropout perturbations is closer to the input layer.

2 Related Work

Good sentence embeddings can be used for many different NLP tasks. Recent approaches use contrastive learning to fine-tune already pre-trained networks and improve the quality of sentence embeddings. However, defining related pairs is not trivial. For visual tasks, applying random transformations (e.g., cropping, resizing, or flipping) to images has demonstrated to be effective (see [7] as an example). In [8], they propose SimCLR where they use random transformations and use the contrastive loss to pull representations of semantically similar augmented images closer together. A similar approach called DeCLUTR [9], proposes a contrastive self-supervised algorithm that uses overlapped spans as positive examples and distant spans as negative examples for learning contrastive span representations. SimCSE solves the problem of creating related pairs by using dropout and feeding the same input through the network twice. The DiffCSE framework extends this principle by adding the Replaced Token Detection (RTD) task. This framework is heavily inspired by the ELECTRA [10] model, in which RTD is used to pre-train model embeddings instead of fine-tuning existing ones. In [11], a broad overview of existing contrastive pre-training methods in NLP is given.

In addition to the unsupervised SimCSE, a supervised version of SimCSE is proposed in the same paper [5]. They use natural language inference (NLI) datasets to generate positive and negative samples. Examples of positive samples include question-answer pairs, two captions of the same image, and multiple translations for the same reference sentence.

3 Approach

Our pipeline is a two-stage process. For the first stage, we implement two contrastive learning methods, SimCSE and DiffCSE, to improve the baseline BERT model embeddings. For the second stage, we implement the multi-task classifier and fine-tune sentence embeddings for the three downstream tasks.

3.1 Contrastive learning

The objective of contrastive learning is to make semantically related representations closer in the embedding space, while pushing unrelated ones apart. To this end, a pre-trained model is fine-tuned to learn to discriminate between pairs (x, x^+) of semantically related sentences (positive samples) and pairs of unrelated sentences (negative samples). Finding a sufficient amount of positive and negative samples is an important challenge. The unsupervised SimCSE framework [5] constructs positive samples by feeding the same sentence *twice* (i.e., making $x^+ = x$) to the BERT encoder f to receive sentence embeddings $\mathbf{h} = f(x)$ and $\mathbf{h}^+ = f(x^+)$. Note that despite x and x^+ being equal, the corresponding embeddings $\mathbf{h}$ and $\mathbf{h}^+$ are not, because dropout layers provide randomly different masks in each forward pass. In this sense, dropout acts as a minimal form of data augmentation. Negative samples are constructed from different sentences from the same batch. The training objective for SimCSE is:

$$\mathcal{L}_{\text{contrast}} = -\log \frac{e^{\text{sim}\left(\mathbf{h}_i, \mathbf{h}_i^+\right)/\tau}}{\sum_{j=1}^{N} e^{\text{sim}\left(\mathbf{h}_i, \mathbf{h}_j^+\right)/\tau}}, \tag{1}$$

where N is the input batch size, $\text{sim}(\cdot, \cdot)$ is the cosine similarity function, and τ is a temperature hyperparameter.

The DiffCSE framework [6] is a more recent approach, which builds upon SimCSE by combining the regular contrastive loss $\mathcal{L}_{\text{contrast}}$ in equation (1) with an additional loss term $\mathcal{L}_{\text{RTD}}$. The loss $\mathcal{L}_{\text{RTD}}$ results from fine-tuning the pre-trained encoder f to provide better embeddings for performing the Replaced Token Detection (RTD) task.

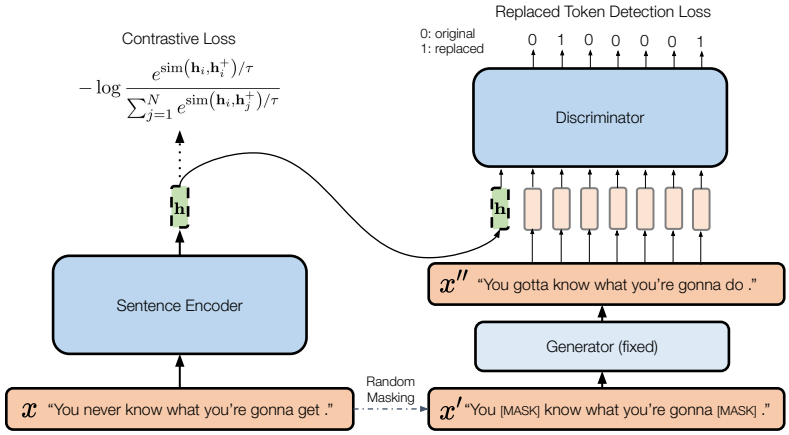

Figure 1: Illustration of DiffCSE obtained from [6].

Figure 1 provides an overview of DiffCSE. The left-hand side represents the standard SimCSE algorithm as described in [5]. The difference prediction model is on the right side. First, a random mask $m \in \{0,1\}^M$ (where M is the length of sentence x) is used to generate $x' = m \cdot x$. Then, a pre-trained masked language model (MLM) is used as generator to predict masked tokens in x' and obtain $x'' = G(x')$. Finally, a discriminator performs the RTD task, predicting whether each word in the sentence has been replaced or not. With all the predictions, $\mathcal{L}_{\text{RTD}}$ can finally be computed as:

$$\mathcal{L}_{\text{RTD}} = \sum_{i=1}^{N} \sum_{j=1}^{M} \left(-\mathbb{1}\left(x''^{(j)}_i = x^{(j)}_i\right) \log D\left(x''_i, \mathbf{h}, t\right) - \mathbb{1}\left(x''^{(j)}_i \neq x^{(j)}_i\right) \log\left(1 - D\left(x''_i, \mathbf{h}, t\right)\right) \right), \tag{2}$$

where $x^{(j)}_i$ is the j-th word in x_i, the i-th sentence in the batch; and $D(\cdot,\cdot,\cdot) \in [0,1]$ is the output of the discriminator. The final training objective for DiffCSE is $\mathcal{L} = \mathcal{L}_{\text{contrast}} + \lambda \cdot \mathcal{L}_{\text{RTD}}$, where λ is a hyperparameter.

For our implementation of DiffCSE, we use our own generator and discriminator, which are both built on top of the baseline BERT model.

3.2 Adjusting dropout

Although the authors of both SimCSE and DiffCSE were thorough in their search for the best dropout probability, there are no studies, to the best of our knowledge, that analyse the effect of having different dropout probabilities in different blocks of the BERT model. In the remainder of this report, we will refer to a combination of different dropout probabilities at different blocks as a *dropout mask*. In our work, we focus on the effect of dropout masks that increase probabilities at the early blocks of the model, to increase the 'dropout-as-data-augmentation' effect.

3.3 Multi-task classifier

The classifier uses a pre-trained BERT model to obtain the sentence embeddings provided by the pooler. We then add a different head for each of the three downstream tasks[2]. For sentiment classification, this head outputs five features, using a softmax function to determine the probabilities for each class. For both paraphrase detection and similarity prediction, the heads take the concatenation of the sentence embeddings for both sentences as input, and provide a single output each. In the case of paraphrase detection, this output is provided to a sigmoid function to compute the likelihood. In all three heads, the number of hidden layers is a hyperparameter.

We consider two different fine-tuning settings. In the first setting (full fine-tuning), we randomly initialize the classifier heads and then train all parameters of the network simultaneously. In the second setting (classifier training then full fine-tuning), we first freeze the pre-trained BERT backbone to only train the heads, and then unfreeze the BERT backbone and fine-tune all parameters.

[2]In our notation, a head may comprise more than just one linear layer.

The baseline for our analysis and improvements is the provided pre-trained model fine-tuned on all three downstream tasks. Also, we will compare our method improvements —i.e., using adjusted dropout masks— to the original methods trained with the same dropout probability for all Transformer blocks.

4 Experiments

4.1 Data

For fine-tuning and pre-training with SimCSE and DiffCSE we use the following four datasets: The Stanford Sentiment Treebank (SST) [12] consists of 11,855 single sentences from movie reviews with 5 different sentiment categories. The dataset is divided into 8,544 train, 1,101 dev and 2,210 test examples. The CFIMDB dataset consists of 2,434 highly polar movie reviews. The dataset is divided into 1,701 train, 245 dev and 488 test examples. The Quora dataset[3] consists of over 400,000 question pairs and each question pair is annotated with a binary value indicating whether the two questions are a paraphrase of each other. The dataset is divided into 141,506 train, 20,215 dev and 40,431 test examples. The SemEval STS Benchmark (STS) dataset [13] consists of 8,628 different sentence pairs of varying similarity. The dataset is divided into 6,041 train, 864 dev and 1,726 test examples. A specific description of how the datasets are used for pre-training and fine-tuning is given in section 4.3.

4.2 Evaluation method

Our main evaluation objective is to test the network on the development splits of the SST dataset for sentiment analysis, the Quora dataset for paraphrase detection, and the SemEval dataset for semantic textual similarity.

Besides the main evaluation objective, we analyse how changing the dropout probability for different layers affects the alignment and uniformity of our embeddings after pre-training with SimCSE and DiffCSE. Alignment and uniformity are two measures to identify key properties related to contrastive learning and were proposed by [14]. Alignment measures whether positive pairs are close together in the embeddings space, and is calculated as follows (assuming the outputs of f are already normalized):

$$\ell_{\text{align}} = \mathop{\mathbb{E}}_{(x,x^+)\sim p_{\text{pos}}} \|f(x) - f(x^+)\|^2.$$

Uniformity measures how uniform the distribution of all embeddings is, and is calculated as follows:

$$\ell_{\text{uniform}} = \log \mathop{\mathbb{E}}_{x,y\sim p_{\text{data}}} e^{-2\|f(x)-f(y)\|^2}.$$

In theory, good embeddings should align positive pairs and the embeddings should be evenly distributed in the embedding space.

4.3 Experimental details

As stated before, our training pipeline consists of two stages. First, we pre-train the baseline BERT model with SimCSE or DiffCSE, and then we fine-tune the model on the three downstream tasks. For pre-training and fine-tuning we use the AdamW optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

Pre-training A suitable dataset for pre-training a model with either SimCSE or DiffCSE is assembled by combining all sentences in the SST, CFIMDB, Quora and STS datasets. For the Quora and SST datasets, which have two sentences for every data point, each of the sentences is treated as an independent sample[4]. We exclude sentences with more than 200 words to be able to use larger batch sizes and handle computational limits.

[3] `https://quoradata.quora.com/First-Quora-Dataset-Release-Question-Pairs`

[4] Note here that the two sentences of those data points can be very similar to each other. In theory, this could lead to a negative effect in the contrastive loss function [5] when two similar sentences are used as negative instances. But as we randomly pick sentences for each batch and the total number of sentences ($\sim$ 300K) is large enough, this effect is marginal.

	Mask 1	Mask 2	Mask 3	Mask 4	Mask 5	Mask 6	Mask 7	Mask 8	Mask 9	Mask 10	Mask 11	Mask 12
Trans. block 1	0.2	0.3	0.4	0.5	0.2	0.3	0.4	0.5	0.5	0.4	0.3	0.2
Trans. block 2	0.2	0.3	0.4	0.5	0.2	0.3	0.4	0.5	0.4	0.3	0.2	0.1
Trans. block 3	0.2	0.3	0.4	0.5	0.2	0.3	0.4	0.5	0.3	0.2	0.1	0.1
Trans. block 4	0.2	0.3	0.4	0.5	0.2	0.3	0.4	0.5	0.2	0.1	0.1	0.1
Trans. block 5	0.1	0.1	0.1	0.1	0.2	0.3	0.4	0.5	0.1	0.1	0.1	0.1
Trans. block 6	0.1	0.1	0.1	0.1	0.2	0.3	0.4	0.5	0.1	0.1	0.1	0.1

Table 1: Dropout probabilities applied to each Transformer block (smaller numbers refer to earlier blocks in the network). All other blocks not listed in the table have dropout probability 0.1.

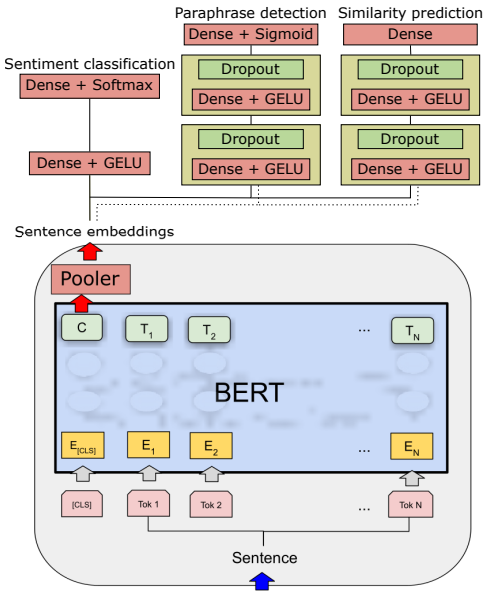

Figure 2: Architecture of multi-task classifier used for fine-tuning on all three downstream tasks.

We pre-train the baseline BERT implementation with SimCSE, with no additional heads and making no modifications to the network. For SimCSE we use the best-performing hyperparameters in [5]: we train with a batch size of 64, a learning rate of 3×10^{-5}, and a temperature parameter τ of 0.05. Instead of training for one epoch as in [5], we train for two epochs to compensate for our smaller dataset. We use a dropout probability of 0.1 for all layers as a baseline, and then run several experiments with different dropout masks. See Table 1 for all tested masks. Masks 1 to 4 change the dropout probability of the first four Transformer blocks, while masks 5 to 8 adjust the dropout probabilities of the first six Transformer blocks to analyse if changing more blocks makes a difference. Finally, masks 9 to 12 gradually decrease the dropout probability in the first blocks to analyse if a smoother transition is beneficial. We only apply mask 1-4 and 9-12 for DiffCSE due to computational limits.

For the discriminator and the generator of DiffCSE, we use the same BERT model used for the sentence encoder. For the generator, we freeze the BERT model and train the generator head on the pre-training dataset for 20 epochs. For this training, we use a batch size of 64 and a learning rate of 10^{-3}. For DiffCSE pre-training, we use the same hyperparameters as for SimCSE. In this case, we use $\lambda = 0.005$ for the loss, as in [6]. As for SimCSE, we train with a dropout probability of 0.1 as baseline and then compare this to different dropout masks.

Fine-tuning For fine-tuning the network on the three downstream tasks, we use their respective datasets: SST, Quora, and STS. The multi-task classifier is depicted in Figure 2. The number of hidden layers for each of the classifier heads is the combination that performs best after tuning (see section A.1 for details).

Model	PD Accuracy	STS Correlation	SST Accuracy	Average
Baseline BERT	0.829	0.546	0.510	0.628
SimCSE Baseline	0.807	0.552	0.501	0.620
SimCSE Mask 1	0.815	0.406	0.484	0.568
SimCSE Mask 2	0.822	0.519	0.502	0.614
SimCSE Mask 3	0.825	0.492	0.499	0.605
SimCSE Mask 4	0.819	0.519	0.510	0.616
SimCSE Mask 5	0.822	0.509	0.510	0.614
SimCSE Mask 6	0.819	0.520	0.494	0.611
SimCSE Mask 7	0.818	0.530	0.482	0.610
SimCSE Mask 8	0.833	0.508	0.494	0.612
SimCSE Mask 9	0.828	0.494	0.481	0.601
SimCSE Mask 10	0.826	0.485	0.475	0.595
SimCSE Mask 11	0.824	0.572	0.488	0.628
SimCSE Mask 12	0.795	0.461	0.510	0.589
DiffCSE Baseline	0.813	0.472	0.456	0.580
DiffCSE Mask 1	0.805	0.561	0.486	0.617
DiffCSE Mask 2	0.822	0.525	0.484	0.610
DiffCSE Mask 3	**0.835**	0.480	0.467	0.594
DiffCSE Mask 4	0.815	**0.582**	0.492	**0.630**
DiffCSE Mask 9	0.822	0.542	0.478	0.614
DiffCSE Mask 10	0.786	0.545	0.497	0.609
DiffCSE Mask 11	0.814	0.476	**0.511**	0.600
DiffCSE Mask 12	0.804	0.492	0.507	0.601

Table 2: Experimental results for full fine-tuning on the development splits. The best results for each metric are highlighted.

To fine-tune on all three datasets, we draw a batch from one of the three datasets at each iteration and use it to train the appropriate head. Since the Quora dataset is much larger than the other two datasets, we add a balancing factor so that for every 17 batches of the Quora dataset we draw 2 batches of the SST dataset and 4 batches of the SemEval dataset. We use cross entropy loss for sentiment classification, binary cross-entropy loss for paraphrase detection and mean squared error for similarity prediction. We update the parameters after each batch[5].

As previously described in section 3.3, we consider two different fine-tuning settings: full fine-tuning, and classifier training then full fine-tuning.

We performed extensive hyperparameter tuning to achieve the best results when performing full fine-tuning on the baseline BERT model. Results of this analysis can be found in section A.1. We achieve the best performance using a batch size of 32, a learning rate of 3×10^{-5}, a weight decay of 10^{-3} and a hidden dropout probability of 0.2. We use these tuned hyperparameters for full fine-tuning on all models pre-trained with SimCSE or DiffCSE to measure the effect of additional pre-training. We train for 10 epochs and report the numbers for the checkpoint with the best average performance.

For classifier training then full fine-tuning, we also perform some hyperparameter tuning, although not as extensive as with full fine-tuning (mainly due to time constraints). During classifier training, we use the same parameters as in full fine-tuning: a learning rate of 3×10^{-5}, a weight decay of 10^{-3} and a dropout probability of 0.2. Then, during full fine-tuning, we use the same parameters except for the learning rate, which becomes 10^{-5}.

4.4 Results

Table 2 shows the results for full fine-tuning. The average best performance is achieved by DiffCSE with mask 4. That being said, the difference between the best-performing model and the baseline BERT model is marginal. In general, the performance of all models is very similar and we cannot see a clear benefit of contrastive learning when using full fine-tuning.

Table 3 shows the results of classifier training then full fine-tuning. The best performance is achieved by SimCSE with mask 5 and improves the performance by over 10% compared to the baseline BERT model. For classifier training then full fine-tuning, the effect of doing additional pre-training compared to the baseline BERT model is clearly measurable for SimCSE and DiffCSE. All models

[5]We also try adding losses of multiple batches to optimize a mutual loss for all three tasks but don't receive better results.

Model	PD Accuracy	STS Correlation	SST Accuracy	Average
Baseline BERT	0.829	0.542	0.500	0.624
SimCSE Baseline	0.855	0.793	0.520	0.723
SimCSE Mask 1	0.865	0.790	0.495	0.717
SimCSE Mask 2	0.865	0.788	0.499	0.717
SimCSE Mask 3	0.866	0.790	0.505	0.720
SimCSE Mask 4	0.865	0.792	0.496	0.718
SimCSE Mask 5	0.864	0.805	0.509	**0.726**
SimCSE Mask 6	**0.867**	0.791	0.505	0.721
SimCSE Mask 7	0.861	**0.807**	0.501	0.723
SimCSE Mask 8	0.864	0.791	0.494	0.716
SimCSE Mask 9	0.864	0.800	0.504	0.723
SimCSE Mask 10	0.865	0.797	0.493	0.718
SimCSE Mask 11	0.867	0.794	0.490	0.717
SimCSE Mask 12	0.864	0.790	0.492	0.715
DiffCSE Baseline	0.850	0.773	0.509	0.711
DiffCSE Mask 1	0.857	0.776	0.501	0.711
DiffCSE Mask 2	0.855	0.777	0.502	0.711
DiffCSE Mask 3	0.856	0.784	0.496	0.712
DiffCSE Mask 4	0.855	0.784	0.513	0.717
DiffCSE Mask 9	0.852	0.777	**0.526**	0.718
DiffCSE Mask 10	0.859	0.789	0.500	0.716
DiffCSE Mask 11	0.860	0.795	0.501	0.719
DiffCSE Mask 12	0.860	0.801	0.512	0.724

Table 3: Experimental results for classifier training then full fine-tuning on the development splits. The best results for each metric are highlighted.

Model	PD Accuracy	STS Correlation	SST Accuracy	Average
SimCSE Mask 5	0.860	0.702	0.511	0.691

Table 4: Experimental results on the test splits for the overall best-performing model on the development splits.

with additional pre-training achieve better average performance, mostly due to the paraphrase detection and semantic textual similarity tasks, where performance increases significantly. This aligns with our expectations. SimCSE and DiffCSE should have a positive effect on paraphrase detection and semantic textual similarity as better alignment and uniformity should help the model to capture the relationship of two sentences better. For sentiment analysis we only observe marginal or no improvements by doing pre-training.

Although the best performance for both pre-training methods is achieved with a specific dropout mask compared to the respective baselines, we don't observe any significant benefit by changing the dropout masks in general. In fact, the opposite seems to be the case. The performance of both methods seems to be robust to the choice of the dropout masks.

Our submission to both the test and dev set leaderboards corresponds to the model pre-trained using SimCSE and dropout mask 5, using classifier training then full fine-tuning. The scores obtained in the test leaderboard is shown in Table 4.

5 Analysis

Our experiments show that the training procedure plays an important role in the success of SimCSE and DiffCSE. Only when doing classifier training and then fine-tuning we see a significant impact of SimCSE and DiffCSE. When directly fine-tuning all parameters of BERT, the further pre-trained features of the backbone seem to get erased in the process. The reason for that can be that the randomly initialized heads lead to large gradients that get propagated through the network. By doing classifier training first, gradients might be smaller and already existing features are preserved.

To better understand why SimCSE and DiffCSE improve the performance in the case of classifier training then full fine-tuning we analyse alignment and uniformity. Figure 3 shows alignment and uniformity of all pre-trained networks we use for our experiments.

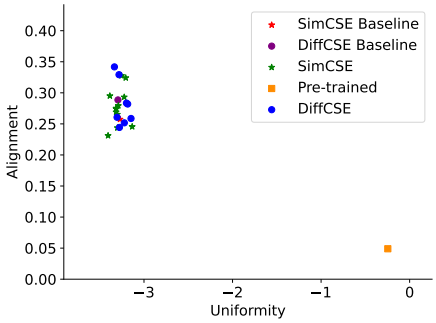

Figure 3: Visualization of alignment and uniformity of SimCSE and DiffCSE compared to the pre-trained network. Each marker in the plot corresponds to one experiment run with a specific pre-training and dropout mask.

Model	PD Accuracy	STS Correlation	SST Accuracy	Average
SimCSE Baseline Wikipedia	0.862	0.792	0.503	0.719
DiffCSE Baseline Wikipedia	0.858	0.745	0.457	0.687

Table 5: Experimental results for classifier training then full fine-tuning with Wikipedia data on the development splits.

To calculate both metrics we use the development split of the STS dataset. In [5] they observe that uniformity of sentence embeddings improves with SimCSE while alignment stays constant (smaller values are better for both metrics). We can only reproduce the improvement of uniformity for SimCSE and DiffCSE[6]. We also train SimCSE with the same Wikipedia sentences used in [5] and [6], but receive similar values for uniformity and alignment as for our training dataset. We assume that our pre-trained network has low alignment and high uniformity compared to the pre-trained network used in [5] and [6], and that this explains the difference. Also, we receive similar values for alignment and uniformity after applying SimCSE and DiffCSE as reported in [5], which hints that SimCSE and DiffCSE create a certain level of uniformity and alignment.

Also, we analyse if changing the training data for pre-training with SimCSE and DiffCSE has an impact on the performance. Table 5 shows the results using the same Wikipedia data that is used in [5] and [6] for pre-training. For SimCSE we achieve similar performance as with our data. DiffCSE with the Wikipedia data used for pre-training performs slightly worse than using our data. This might be because using our data for pre-training already involves data from the target domain, but a more thorough analysis is needed to measure the exact effect of the data used during pre-training.

In general, we don't see a significant benefit of DiffCSE compared to SimCSE in our experiments. The performance of DiffCSE is (among other things) dependant on the generator that is used. A more thorough analysis is needed to test different generators and the effect on the performance of DiffCSE.

6 Conclusion

Our analysis shows that further pre-training using SimCSE and DiffCSE can significantly improve the performance of BERT on multiple downstream tasks. We achieve 10% better average performance over all three downstream tasks compared to our baseline. We find out that the training procedure plays a crucial role in utilizing the additional knowledge that is incorporated into the model by pre-training with SimCSE and DiffCSE. Training the task-specific heads before fine-tuning all parameters enables the model to preserve the improved features of SimCSE and DiffCSE. Also, in our analysis, the pre-training methods are robust against changes in the dropout masks. Contrary to [5], SimCSE and DiffCSE only improve uniformity of sentence embeddings independent of the training data, showing that these observations depend on the already pre-trained network used for both methods. We also show the effect of the training data used for both methods and that the choice of training data affects the performance of DiffCSE.

[6]Note that we use the same dataset to evaluate uniformity and alignment as in [5] and [6].

References

[1] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680. Association for Computational Linguistics, September 2017.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, Long Beach, United States, December 2017. Curran Associates, Inc.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[4] Katikapalli Subramanyam Kalyan, Ajit Rajasekharan, and Sivanesan Sangeetha. AMMUS: A survey of transformer-based pretrained models in natural language processing, August 2021. arXiv:2108.05542 [cs].

[5] Tianyu Gao, Xingcheng Yao, and Danqi Chen. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Punta Cana, Dominican Republic, May 2022. Association for Computational Linguistics.

[6] Yung-Sung Chuang, Rumen Dangovski, Hongyin Luo, Yang Zhang, Shiyu Chang, Marin Soljačić, Shang-Wen Li, Wen-tau Yih, Yoon Kim, and James Glass. DiffCSE: Difference-based contrastive learning for sentence embeddings. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4207–4218, Seattle, United States, July 2022. Association for Computational Linguistics.

[7] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin A. Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. *CoRR*, abs/1406.6909, 2014.

[8] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, pages 1597–1607. JMLR.org, July 2020.

[9] John M. Giorgi, Osvald Nitski, Gary D. Bader, and Bo Wang. Declutr: Deep contrastive learning for unsupervised textual representations. *CoRR*, abs/2006.03659, 2020.

[10] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *Proceedings of the 8th International Conference on Learning Representations*, April 2020.

[11] Nils Rethmeier and Isabelle Augenstein. A primer on contrastive pretraining in language processing: Methods, lessons learned, and perspectives. *ACM Computing Surveys*, 55(10):1–17, February 2023.

[12] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[13] Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. *SEM 2013 shared task: Semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, Atlanta, Georgia, USA, June 2013. Association for Computational Linguistics.

[14] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9929–9939. PMLR, July 2020.

A Appendix

A.1 Hyperparameter tuning

In order to optimize the performance of our model on the three downstream tasks, we conduct an exhaustive hyperparameter tuning process. We evaluate a range of hyperparameters including the learning rate, batch size, dropout probability, L2 weight decay, dataset ratios, classifier depth, and loss weights. The dataset ratios determine how frequently we sample from each dataset relative to the others, while the classifier depth refers to the number of layers applied on top of the pooler layer, with separate values for paraphrase detection, similarity prediction, and sentiment analysis (the first value refers to the number of layers we apply on top of the pooler layer for all tasks). The loss weights indicate the degree to which each loss is prioritized relative to the others. The outcomes of this hyperparameter tuning process are presented in Table 6.

LR	BS	DP	WD	DS ratios	Classifier depths	Loss weights	Avg. score
3×10^{-5}	16	0.3	10^{-2}	17 4 2	0 0 0 0	1 1 1	0.540
3×10^{-5}	16	0.3	10^{-2}	17 4 2	0 1 1 0	1 1 1	0.604
3×10^{-5}	16	0.3	10^{-2}	17 4 2	0 2 2 0	1 1 1	0.614
3×10^{-5}	16	0.3	10^{-2}	17 4 2	0 3 3 0	1 1 1	0.595
3×10^{-5}	16	0.3	10^{-2}	17 4 2	0 4 4 0	1 1 1	0.553
3×10^{-5}	32	0.3	10^{-2}	17 4 2	0 0 0 0	1 1 1	0.549
3×10^{-5}	32	0.3	10^{-2}	17 4 2	0 1 1 0	1 1 1	0.593
3×10^{-5}	32	0.3	10^{-2}	17 4 2	0 2 2 0	1 1 1	0.605
3×10^{-5}	32	0.3	10^{-2}	17 4 2	0 3 3 0	1 1 1	0.598
3×10^{-5}	32	0.3	10^{-2}	17 4 2	0 4 4 0	1 1 1	0.595
3×10^{-5}	32	0.3	10^{-2}	17 4 2	0 1 2 0	1 1 1	0.603
3×10^{-5}	32	0.3	10^{-2}	17 4 2	1 1 2 0	1 1 1	0.580
3×10^{-5}	32	0.3	0	17 4 2	0 2 2 0	1 1 1	0.610
3×10^{-5}	32	0.3	10^{-3}	17 4 2	0 2 2 0	1 1 1	0.612
3×10^{-5}	32	0.3	0.1	17 4 2	0 2 2 0	1 1 1	0.600
3×10^{-5}	32	0.3	1	17 4 2	0 2 2 0	1 1 1	0.577
3×10^{-5}	32	0.3	10	17 4 2	0 2 2 0	1 1 1	0.411
10^{-5}	32	0.3	10^{-3}	17 4 2	0 2 2 0	1 1 1	0.572
10^{-4}	32	0.3	10^{-3}	17 4 2	0 2 2 0	1 1 1	0.310
3×10^{-5}	32	0.1	10^{-3}	17 4 2	0 2 2 0	1 1 1	0.621
$\mathbf{3 \times 10^{-5}}$	**32**	**0.2**	$\mathbf{10^{-3}}$	**17 4 2**	**0 2 2 0**	**1 1 1**	**0.628**
3×10^{-5}	32	0.2	10^{-3}	17 4 2	0 2 2 0	1 1 1	0.617
3×10^{-5}	32	0.4	10^{-3}	17 4 2	0 2 2 0	1 1 1	0.584
3×10^{-5}	32	0.5	10^{-3}	17 4 2	0 2 2 0	1 1 1	0.606
3×10^{-5}	32	0.2	10^{-3}	17 4 2	0 2 2 0	1 1 0.3	0.622
3×10^{-5}	32	0.2	10^{-3}	17 4 2	0 2 2 0	1 1 0.1	0.568
3×10^{-5}	32	0.2	10^{-3}	17 4 4	0 2 2 0	1 1 1	0.602
3×10^{-5}	32	0.2	10^{-3}	35 4 4	0 2 2 0	1 1 1	0.613
3×10^{-5}	32	0.2	10^{-3}	35 3 4	0 2 2 0	1 1 1	0.605
3×10^{-5}	32	0.2	10^{-3}	35 3 2	0 2 2 0	1 1 1	0.610

Table 6: Hyperparameter tuning results on the development splits. Best parameter combination is highlighted. LR = learning rate; BS = batch size; DP = dropout probability; WD = weight decay; DS ratios = dataset ratios; Avg. score = mean score of paraphrase detection, sentiment analysis and similarity prediction.