2/9/2026 11:37:52 PM

Compare Results

Old File:

2023.emnlp-main.124.pdf

12 pages (10.53 MB)

versus

New File:

2023_emnlp-main_124.pdf

11 pages (299 KB)

2/9/2026 11:24:33 PM

Total Changes

25

Content

4 Replacements

10 Insertions

11 Deletions

Styling and Annotations

0 Styling

0 Annotations

Go to First Change (page 1)

Combining Denoising Autoencoders with Contrastive Learning to fine-tune Transformer Models

Alejo L'opez-Avila V'ictor Su'arez-Paniagua

Abstract

Recently, using large pre-trained Transformer models for transfer learning tasks has evolved to the point where they have become one of the flagship trends in the Natural Language Processing (NLP) community, giving rise to various outlooks such as prompt-based, adapters, or combinations with unsupervised approaches, among many others. In this work, we propose a 3-Phase technique to adjust a base model for a classification task. First, we adapt the model's signal to the data distribution by performing further training with a Denoising Autoencoder (DAE). Second, we adjust the representation space of the output to the conesponding classes by clustering through a Contrastive Leaming (CZ) method. In addition, we introduce a new data augmentation approach for Supervised Contrastive Learning to correct the unbalanced datasets. Third, we apply fine-tuning to delimit the predefined categories. These different phases provide relevant and complementary knowledge to the model to learn the final task. We supply extensive experimental results on several datasets to demonsffate these claims. Moreover, we include an ablation study and compare the proposed method against other ways of combining these techniques.

1 Introduction

The never-ending starvation of pre-trained Transformer models has led to fine-tuning these models becoming the most common way to solve target tasks. A standard methodology uses a pre-trained model with self-supervised learning on large data as a base model. Then, replace/increase the deep layers to learn the task in a supervised way by leveraging knowledge from the initial base model. Even though specialised Transformers, such as Pegasus (Zhang et a1.,2020) for summarisation, have appeared in recent years, the complexity and resources needed to train Transformer models of these scales make fine-tuning methods the most reasonable choice. This paradigm has raised interest in improving these techniques, either from the point of the architecture (Qin et a1.,2019), by altering the definition of the fine-tuning task (Wang et al.,2021b) or the input itself (Brown et a1.,2020), but also by revisiting and proposing different self-supervised training practices (Gao et a1.,2021). We decided to explore the latter and offer a novel approach that could be utilised broadly for NLP classification tasks. We combine some self-supervised methods with fine-tuning to prepare the base model for the data and the task. Thus, we will have a model adjusted to the data even before we start fine-tuning without needing to train a model from scratch, thus producing better results, as shown in the experiments.

A typical way to adapt a neural network to the input distribution is based on Autoencoders. These systems introduce a bottleneck for the input data distribution through a reduced layer, a sparse layer activation, or a restrictive loss, forcing the model to reduce a sample's representation at the narrowing. A more robust variant of this architecture is DAE, which corrupts the input data to prevent it from learning the identity function. The first phase of the proposed method is a DAE (Fig. 1a), replacing the final layer of the encoder with one more adapted to the data distribution.

Contrastive Learning has attracted the attention of the NLP community in recent years. This family of approaches is based on comparing an anchor sample to negative and positive samples. There are several losses in CL, like the triplet loss (Schroff et al., 2015), the contrastive loss (Chopra et al., 2005), or the cosine similarity loss. The second phase proposed in this work consists of a Contrastive Learning using the cosine similarity (as shown in Fig. 1b). Since the data is labelled, we use a supervised approach similar to the one presented in (Khosla et a1.,2020) but through Siamese Neural Networks. In contrast, we consider Contrastive Learning and fine-tuning the classifier (FT) as two distinct stages. We also add a new imbalance correction during data augmentation that avoids overfitting. This CL stage has a clustering impact since the vector representation belonging to the same class will tend to get closer during the training. We chose some benchmark datasets in classification tasks to support our claims. For the hierarchical ones, we can adjust labels based on the number of similar levels among samples.

Finally, once we have adapted the model to the data distribution in the first phase and clustered the representations in the second, we apply fine-tuning at the very last. Among the different variants from fine-tuning, we use the traditional one for Natural language understanding (NLU), i.e., we add a small Feedforward Neural Network (FNN) as the classifier on top of the encoder with two layers. We use only the target task data without any auxiliary dataset, making our outlook self-contained. The source code is publicly available at GitHub[1]](https://github.com/vsuarezpaniagua/3-phase$_finetuning$).

To summarise, our contribution is fourfold:

1. We propose a 3-Phase fine-tuning approach to adapt a pre-trained base model to a supervised classification task, yielding more favourable results than classical fine-tuning.

2. We propose an imbalance correction method by sampling noised examples during the augmentation, which supports the Contrastive Learning approach and produces better vector representations.

3. We analyze possible ways of applying the described phases, including ablation and joint loss studies.

4. We perform experiments on several well-known datasets with different classification tasks to prove the effectiveness of our proposed methodology.

2 Related Work

One of the first implementations was presented in (Reimers and Gurevych, 2019), an application of Siamese Neural Networks using BERT (Devlin et al., 2018) to learn the similarity between sentences.[2]](https://github.com/vsuarezpaniagua/3-phase$_finetuning$)

Autoencoders were introduced in (Kramer, 1991) and have been a standard for self-supervised learning in Neural Networks since then. However, new modifications were created with the explosion of Deep Learning architectures, such as DAE (Vincent et al., 2010) and masked Autoencoders (Germain et al., 2015). The Variational Autoencoder (VAE) has been applied for NLP in (Miao et al., 2015) or (Li et al., 2019) with RNN networks or a DAE with Transformers in (Wang et al., 2021a). Transformers-based Sequential Denoising Auto-Encoder (Wang et al., 2021a) is an unsupervised method for encoding the sentence into a vector representation with limited or no labelled data, creating noise through an MLM task. The authors of that work evaluated their approach on the

[1] [https://github.com/vsuarezpaniagua/3-phase_finetuning

[2] [https://github.com/vsuarezpaniagua/3-phase_finetuning

following three tasks: Information Retrieval, Re-Ranking, and Paraphrase Identification, showing an increase of up to 6.4 points compared with previous state-of-the-art approaches. In (Savinov et al., 2021), the authors employed a new Transformer architecture called Step-unrolled Denoising Autoencoders. In the present work, we will apply a DAE approach to some Transformer models and extend its application to sentence-based and more general classification tasks.

The first work published on Contrastive Learning was (Chopra et al., 2005). After that, several versions have been created, like the triplet net in Facenet (Schroff et al., 2015) for Computer Vision. The triplet-loss compares a given sample and randomly selected negative and positive samples making the distances larger and shorter, respectively. One alternative approach for creating positive pairs is slightly altering the original sample. This method was followed by improved losses such as N-pairLoss (Sohn, 2016) and the Noise Contrastive Estimation (NCE) (Gutmann and Hyv"arinen, 2010), extending the family of CL techniques. In recent years, further research has been done on applying these losses, e.g. by supervised methods such as (Khosla et al., 2020), which is the one that most closely resembles one in our second phase.

Sentence-BERT (Reimers and Gurevych, 2019) employs a Siamese Neural Network using BERT with a pooling layer to encode two sentences into a sentence embedding and measure their similarity score. Sentence-BERT was evaluated on Semantic Textual Similarity (STS) tasks and the SentEval toolkit (Conneau and Kiela, 2018), outperforming other embedding strategies in most tasks. In our particular case, we also use Siamese Networks within the CL options. Similar to this approach, the Simple Contrastive Sentence Embedding (Gao et al., 2021) is used to produce better embedding representations. This unlabelled data outlook uses two different representations from the same sample, simply adding the noise through the standard dropout. In addition, they tested it using entailment sentences as positive examples and contradiction sentences as negative examples and obtained better results than SBERT in the STS tasks.

Whereas until a few years ago, models were trained for a target task, the emergence of pre-trained Transformers has changed the picture. Most implementations apply transfer learning on a Transformer model previously pre-trained on general NLU, on one or more languages, to the particular datasets and for a predefined task.

This new paradigm has guided the search for the best practice in each of the trending areas such as prompting strategies (Brown et al., 2020), Few Shot Learning (Wang et al., 2021b), meta-learning (Finn et al., 2017), also some concrete tasks like Intent Detection (Qin et al., 2019), or noisy data (Siddhant Garg, 2021). Here, we present a task-agnostic approach, outperforming some competitive methods for their corresponding tasks. It should also be noted that our method benefits from using mostly unlabelled data despite some little labelled data. In many state-of-the-art procedures, like (Sun et al., 2020a), other datasets than the target dataset are used during training. In our case, we use only the target dataset.

3 Model

In this section, we describe the different proposed phases: a Denoising Autoencoder that makes the inputs robust against noise, a Contrastive Learning approach to identify the similarities between different samples in the same class and the dissimilarities with the other class examples together with a novel imbalance correction, and finally, the traditional fine-tuning of the classification model. In the last part of the section, we describe another approach combining the first two phases into one loss.

3.1 DAE: Denoising Autoencoder phase

The Denoising Autoencoder is the first of the proposed 3-Phase approach, shown in Fig. 1a. Like any other Autoencoder, this model consists of an encoder and a decoder, connected through a bottleneck. We use two Transformer models as encoders and decoders simultaneously: RoBERTa (Liu et al., 2019), and all-MiniLM-L12-v2 (Wang et al., 2020). The underlying idea is that the bottleneck represents the input according to the general distribution of the whole dataset. A balance needs to be found between preventing the information from being memorised and having sufficient sensitivity to be reconstructed by the decoder, forcing the bottleneck to learn the general distribution. We add noise to the Autoencoder to prevent the bottleneck from memorising the data. We apply a Dropout on the input to represent the noise. Formally, for a sequence of tokens $X : x_0, \ldots, x_n$ coming from a data distribution D, we define the loss as [$\mathrm{L}_{DAE} = \mathbb{E} * p[\log P * \theta(\tilde{X} \mid X)]]whereisthesequenceXafteraddingthenoise.Tosupportwithanexample, maskingatokenatthepositioniwouldpr : \mathrm{x}_0, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n$. The distribution of P_θ in this Cross-Entropy loss corresponds to the composition of the decoder with the encoder. We consider the ratio of noise like another hyper-parameter to tune. More details can be found in Appendix A.1, and the results section 5. Instead of applying the noise to the dataset and running a few epochs over it, we apply the noise on the fly, getting a very low probability of a repeated input.

Once the model has been trained, we extract the encoder and the bottleneck, resulting in DAE (Fig. 1a), which will be the encoder for the next step. Each model's hidden size has been chosen as its bottleneck size (768 in the case of RoBERTa). The key point of the first phase is to adapt the embedding representation of the encoder to the target dataset distribution, i.e., this step shifts the distribution from the general distribution learned by the pre-trained Transformers encoder into one of the target datasets. It should be noted here that this stage is in no way related to the categories for qualification. The first phase was implemented using the SBERT library.[3]](https://www.sbert.net/index.html)

3.2 CL: Contrastive Learning phase

The second stage will employ Contrastive Learning, more precisely, a Siamese architecture with cosine similarity loss. The contrastive techniques are based on comparing pairs of examples or anchor-positive-negative triplets. Usually, these methods have been applied from a semi-supervised point of view. We decided on a supervised outlook where the labels to train in a supervised manner are the categories for classification, i.e., we pick a random example. Then, we can get a negative input by sampling from the other categories or a positive one by sampling from the same category. We combine this process of creating pairs with the imbalance correction explained below to get pairs of vector outputs (z, z'). Given two inputs u and v and a label label $* u, v$ based on their class similarity, [label*u,v $= \begin{cases} 1, & \text{if } u \text{ and } v \text{ are in the same class } 0, \\ \text{otherwise.} \end{cases}$]

We use a straightforward extension for the hierarchical dataset (AGNews) by normalising these weights along the number of levels. In the case of two levels, we assign 1 for the case where all the labels match, 0.5 when only the first one matches, and 0 if none. After applying the encoder, we obtain u and v. We define the loss over these outputs as the Mean Squared Error (MSE): [$\mathrm{L}_{CL} = \|\mathrm{label}_{u,v} - \mathrm{CosineSim}(u, v)\|^2$.]

We apply this Contrastive Learning to the encoder DAE, i.e., the encoder and the bottleneck from the previous step. Again, we add an extra layer to this encoder model, producing our following final embedding. We denote this encoder after applying CL over DAE as DAECL (Fig. 1b).

[3] [https://www.sbert.net/index.html

Similarly, we chose the hidden size per model as the embedding size. CL plays the role of soft clustering for the embedding representation of the text based on the different classes. This will make fine-tuning much easier as it will be effortless to distinguish representations from distinct classes. The second phase was also implemented through the SBERT library.

3.2.1 Imbalance correction

As mentioned in the previous section, we are using a Siamese network and creating the pairs in a supervised way. It is a common practice to augment the data before applying CL. In theory we can make as many example combinations as possible. In the case of the Siamese models, we have a total of $n(n-1)/2$ unique pair combinations that correspond to the potential number of pairs based on symmetry. Typically, data can be increased as much as one considers suitable. The problem is that one may fall short or overdo it and produce overfitting in the smaller categories. Our augmentation is based on two pillars. We start with correcting the imbalance in the dataset by selecting underrepresented classes in the dataset more times but without repeating pairs. Secondly, on the classes that have been increased, we apply noise on them to provide variants that are close but not the same to define more clearly the cluster to which they belong.

We balance the dataset by defining the number of examples that we are going to use per class based on the most significant class (where max is its size) and a range of ratios, from $minratio$, the minimum and $maxratio$ the maximum. These upper and lower bounds for the ratios are hyper-parameters, and we chose 4 and 1.5 as default values. Formally, the new ratio is defined by the function: [f(x) = log! $\left(\frac{max}{x}\right)^{\frac{minratio}{\log maxratio}}$.]

where x refers to the initial size of a given class. After adding a lower bound for the ratio, we get the final amount [newratio$_k$ = min!$(maxratio, f(class_k))][newclass_k = newratio_k \times class_k] for$k = 1,2,. . . ,K$in a classification problem with K classes, where$class$_k$ and $newclass_k$ are the cardinalities of the class k, before and after the resizing, respectively. As we can observe, the function gives $f(1) = maxratio$ for one example, so we will never get something bigger than the maximum. The shape of this function is similar to a negative log-likelihood, giving a high ratio to the small classes and around the minimum for medium or bigger. A simple computation shows that the ratio 1 in function f is obtained at [x = max$^{\frac{maxratio-1}{maxratio}}$]$ormax^{7/8}$ in our default case.

We duplicate this balanced dataset and shuffle both to create the pairs. Since many combinations exist, we only take the unique ones without repeating pairs, broadly preserving the proportions. Even so, if just a few examples represent one class, the clustering process from CL can be affected by the augmentation because the border between them would be defined just for a few examples. To avoid this problem, we add a slight noise to the tokens when the text length is long enough. This noise consists of deleting some of the stop-words from the example. Usually, this noise was added to create positive samples and produced some distortion to the vector representation. Adding it twice, in the augmentation and the CL, would produce too much distortion. However, since we are using a supervised approach, this does not negatively affect the model, as shown in the ablation section 5.

3.3 FT: fine-tuning phase

Our final stage is fine-tuning, obtained by employing DAECL as our base model (as indicated in Fig. 1b). We add a two-layer MLP on top as a classifier. We tried both to freeze and not to freeze the previous neurons from DAECL.

As the final activation, we use Softmax, which is a sigmoid function for the binary cases. More formally, for K classes, softmax corresponds to [$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \quad i = 1, 2, \ldots, K.$]

As a loss function for the classification, we minimize the use of Cross-Entropy: [$L_{FT} = -\sum_{k=1}^{K} y_k \log(p_k)] where p_k$ is the predicted probability for the class k and y_k for the target. For binary classification datasets this can be further expanded as [L = -(y log(p) + (1 − y) log(1 − p)).]

3.3.1 Joint

We wanted to check if we could benefit more when combining losses, i.e. by creating a joint loss based on the first and the second loss, (1 and 3), respectively. [$L_{Joint} = L_{DAE} + L_{CL}$.]

This training was obtained as a joint training of stages one and two. By adding the classification head, like in the previous section, for fine-tuning, we got the version we denote as Joint (see Table 3).

4 Experimental Setup

5 Experimental Setup

The datasets for the experiments, the two base models used and the metrics employed are detailed below.

5.1 Datasets

We have chosen several well-known datasets to carry out the experiments:

- Intent recognition on SNIPS (SNIPS Natural Language Understanding benchmark) (Coucke et al., 2018). For this dataset, we found different versions, the first one with just 327 examples and 10 categories. This one was obtained from the huggingface library[4]](https://huggingface.co/datasets/snips$_built_i$

- The second version for SNIPS from Kaggle[5]](https://www.kaggle.com/datasets/weipengfei/atis-snips) containing more samples and split into 7 classes that we call SNIPS2 is the most common one.

- The third one is commonly used for slot prediction (Qin et al., 2019), although here we only consider task intent recognition.

- We used SST2 and SST5 from (Socher et al., 2013) for classification containing many short text examples.

- We add AGNews (Zhang et al., 2015) to our list, a medium size dataset that shows our method over long text.

- We complement the experiments with IMDB (Maas et al., 2011), a big dataset with long inputs for binary classification in sentiment analysis. The length and size of this data made us consider only the RoBERTa as the base model.

We used Hugging Face API to download all the datasets apart from the second version of SNIPS. In some cases, there was no validation dataset, so we used 20

[4] https://huggingface.co/datasets/snips_built_in_intents
[5] https://www.kaggle.com/datasets/weipengfei/atis-snips

5.2 Models and Training

We carried out experiments with two different models, a small model for short text all-MiniLM-L12-v2 (Wang et al., 2020)[6]](https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2), more concretely, a version fine-tuned for sentence similarity and a medium size model RoBERTa-base (Liu et al., 2019)[7]](https://huggingface.co/roberta-base) which we abbreviate as RoBERTa. The ranges for the hyper-parameters below, as well as the values of the best accuracy, can be found in Appendix A.1.

We use Adam as the optimiser. We test different combinations of hyper-parameters, subject to the model size. We tried batch sizes from 2 to 128, whenever possible, based on the dataset and the base model. We tested the encoder with both frozen and not frozen weights — almost always getting better results when no freezing is in place. We tested two different truncations' lengths based either on the maximum length in the training dataset plus a 20

5.3 Metrics

We conducted experiments using the datasets previously presented in Section 4.1. We used the standard macro metrics for classification tasks, i.e., Accuracy, F1-score, Precision, Recall, and the confusion matrix. We present only the results from Accuracy in the main table to compare against other works. The results for other metrics can be found in Appendix A.2.

6 Results

7 Results

We assess the performance of the three methods against the several prominent publicly available datasets. Thus, here we evaluate the 3-Phase procedure compared to Joint and FT approaches. We report the performance of these approaches in terms of accuracy in Table 3.

We observe a noticeable improvement of at least 1

We did not have a bigger model for those datasets with long input, so we tried to compare against approaches with similar base models. We truncated the output for the datasets with the longest inputs, which may reflect smaller values in our case. Since the advent of (Sun et al., 2020a), several current techniques are based on combining different datasets in the first phase through multi-task learning and then fine-tuning each task in detail. Apart from (Sun et al., 2020a), this is the case for the prompt-base procedure from EFL (Wang et al., 2021b) as well. Our method focuses on obtaining the best results for a single task and dataset. Several datasets to pre-train the model could be used as a phase before all the others. However, we doubt the possible advantages of this as the first and second phases would negatively affect this learning, and those techniques focused on training the classifier with several models would negatively affect the first two phases.

7.1 Ablation study

We conducted ablation experiments on all the datasets, choosing the same hyper-parameters and base model as the best result for each one. The results can be seen in Table 4.

We start the table with the approaches mentioned: 3-Phase and Joint. We wanted to see if combining the first two phases could produce better results as those would be learned simultaneously.

[6] [https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2

[7] [https://huggingface.co/roberta-base

The results show that merging the two losses always leads to worse results, except for one of the datasets where they give the same value.

We start the ablations by considering only DAE right before fine-tuning, denoting it as DAE+FT. In this case, we assume that the fine-tuning will carry all the class information. One advantage of this outlook is that it still applies to models that employ a small labelled fraction of all the data (i.e., the unlabelled data represents the majority). The next column, CL+FT, replaces DAE with Contrastive Learning, concentrating the attention on the classes and not the data distribution. Considering only the classes and fine-tuning in CL+FT, we get better results than in DAE+FT, but still lower than the 3-Phase in almost all the datasets. Right after, we add two extreme cases of the imbalance correction, where Extra Imb. increases the upper bound for the ratio and No Imb. excludes the imbalance method. Both cases generally produce lower accuracies than 3-Phase, being No Imb. slightly lower. The last column corresponds to fine-tuning FT.

All these experiments proved that the proposed 3-Phase approach outperformed all the steps independently, on its own, or combined the Denoising Autoencoder and the Contrastive Learning as one loss.

8 Conclusion

The work presented here shows the advantages of fine-tuning a model in different phases with an imbalance correction, where each stage considers certain aspects, either as an adaptation to the text characteristics, the class differentiation, the imbalance, or the classification itself. We have shown that the proposed method can be equally or more effective than other methods explicitly created for a particular task, even if we do not use auxiliary datasets. Moreover, in all cases, it outperforms classical fine-tuning, thus proving that classical fine-tuning only partially exploits the potential of the datasets. Squeezing out all the juice from the data requires adapting to the data distribution and grouping the vector representations according to the task before the fine-tuning, which in our case is targeted towards classification.

9 Future work

The contrastive training phase benefits of data augmentation, i.e., we can increase the number of examples simply through combinatorics. However, this can lead to space deformation for small datasets, even with the imbalance correction, as fewer points are considered. Therefore, overfitting occurs despite the combinatoric strategy. Another advantage of this phase is balancing the number of pairs with specific values. This practice allows us, for example, to increase the occurrence of the underrepresented class to make its cluster as well defined as those of the more represented categories (i.e. ones with more examples). This is a partial solution for the imbalance problem.

In the future, we want to address these problems. For the unbalanced class in the datasets, seek a solution to avoid overfitting to the under-represented classes and extend our work to support a few shot learning settings (FSL). To do so, we are going to analyze different data augmentation techniques. Among others, Variational Autoencoders. Recent approaches for text generation showed that hierarchical VAE models, such as stable diffusion models, may produce very accurate augmentation models. One way to investigate this is to convert the first phase into a VAE model, allowing us to generate more examples from underrepresented classes and generally employ them all in the FSL setting.

Finally, we would like to combine our approach with other fine-tuning procedures, like prompt methods. Adding a prompt may help the model gain previous knowledge about the prompt structure

instead of learning the prompt pattern simultaneously during the classification while fine-tuning.

Limitations

[MISSING]

Acknowledgments

[MISSING]

References

References

[1] [ILLEGIBLE]

[2] [ILLEGIBLE]

[3] [ILLEGIBLE]

[4] [ILLEGIBLE]

[5] [ILLEGIBLE]

[6] [ILLEGIBLE]

[7] [ILLEGIBLE]

[8] [ILLEGIBLE]

[9] [ILLEGIBLE]

[10] [ILLEGIBLE]

[11] [ILLEGIBLE]

[12] [ILLEGIBLE]

[13] [ILLEGIBLE]

[14] [ILLEGIBLE]

[15] [ILLEGIBLE]

[16] [ILLEGIBLE]

[17] [ILLEGIBLE]

[18] [ILLEGIBLE]

[19] [ILLEGIBLE]

[20] [ILLEGIBLE]

[21] [ILLEGIBLE]

[22] [ILLEGIBLE]

[23] [ILLEGIBLE]

[24] [ILLEGIBLE]

[25] [ILLEGIBLE]

[26] [ILLEGIBLE]

[27] [ILLEGIBLE]

[28] [ILLEGIBLE]

[29] [ILLEGIBLE]

[30] [ILLEGIBLE]

[31] [ILLEGIBLE]

[32] [ILLEGIBLE]

A Appendix

A.1 Hyper-parameters

This appendix section shows the final hyper-parameters from the best results in Table 3. The column at the end on the right contains the search space used for training. Some of the values were not used for training, either because of computational limitations or because they were not realistic for some datasets.

[ILLEGIBLE]

A.2 Other metrics

[ILLEGIBLE]