SETFIT (Sentence Transformer Fine Tuning)*

Lorenc Zhuka$^{1\,[1112405]}$

University of Applied Sciences, Darmstadt

Abstract. Recent advances in few-shot learning have led to impressive results in scenarios where only a small amount of labeled data is available. Techniques such as standard fine-tuning and few-shot learning with large language models have demonstrated their effectiveness. However, these methods are challenging in practice because they rely on manually generated prompts and require large language models with billions of parameters to achieve high accuracy. SETFIT overcomes these limitations. SETFIT is a prompt-free and efficient framework specifically designed for few-shot fine-tuning of Sentence Transformers (ST). The core idea behind SETFIT is to first fine-tune a pre-trained ST on a small set of text pairs in a contrastive Siamese manner. This process leads to a refined model capable of generating comprehensive text embeddings. Subsequently, these embeddings are utilized to train a classification head. My experiments show that SETFIT achieves comparable results to standard fine-tuning on few-shot training data as well as on the entire dataset, while being trained an order of magnitude faster. I also propose my Hard Negative/Positive Sampling approach to generate the data used to fine-tune the Sentence Transformer, and compare its performance to the Random Sampling currently used to implement SETFIT.

Keywords: SETFIT · Transformer · Sentence Transformer · Sentence Embeddings · HuggingFace · BERT · fine-tuning · few-shot learning · in-context learning

1 Introduction

In NLP, a language model for learning a particular task is usually fine-tuned using our own data. However, the standard fine-tuning approach has some drawbacks, including the need for a large data set. In standard fine-tuning, all parameters of the model are updated, and since language models have millions or billions of parameters, this can lead to long training times. Also, we need computational resources to train and use these language models. There are several few-shot learning techniques that overcome some of these limitations.

Few-shot learning techniques have gained popularity due to their effectiveness in scenarios where labeled data is limited and annotation is time-consuming. These methods are designed to process a small number of labeled training examples and involve adapting pre-trained language models (PLMs) for specific tasks.

* Supported by University of Applied Sciences, Darmstadt

Currently, there are several approaches to few-shot learning with PLMs, including in-context learning, and parameter-efficient fine-tuning. A limitation of these approaches is that they rely on large language models to achieve high performance, as shown in Fig. 1. The size of models like GPT-3, with billions of parameters, presents a challenge.

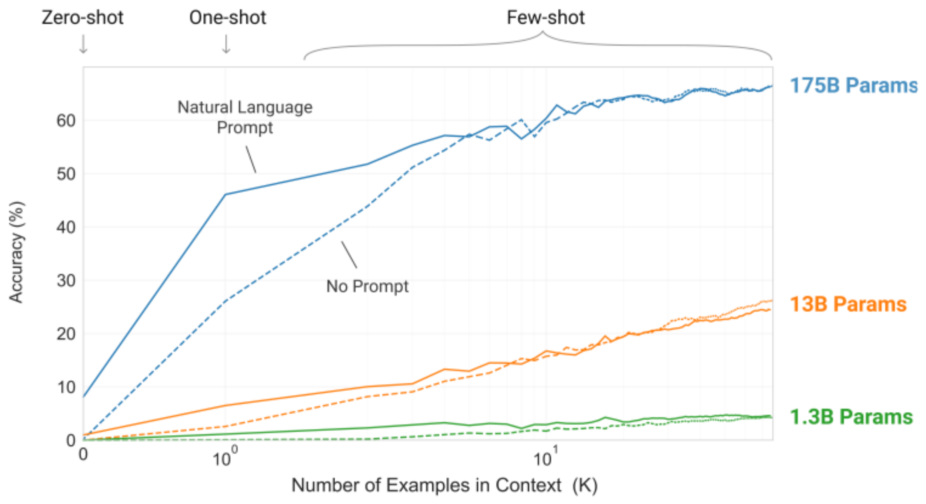
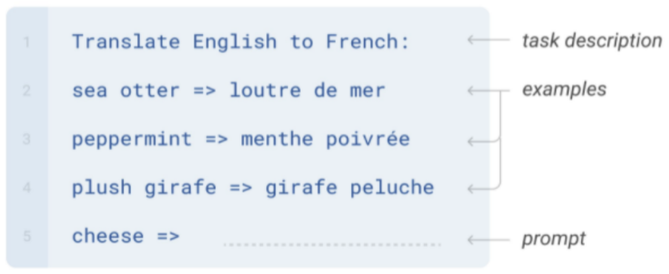


Fig. 1: Performance of in-context learning increases with the scale of the language model.

Furthermore, training and use of these methods often require specialized infrastructure, making them less accessible. In addition, few-shot learning techniques rely on manually created prompts during training, leading to different results depending on how the prompts are designed. One example of in-context learning is shown in Fig. 2.



```
Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>
```

Fig. 2: Example of designing few-shot examples in language translation.

To address these issues, SETFIT is an alternative approach based on Sentence Transformers (ST) that eliminates the need for prompts and does not rely on large-scale PLMs to achieve high accuracy. SETFIT is competitive with standard fine-tuning at larger training sets, although the standard fine-tuned model is larger. The effectiveness of SETFIT is demonstrated using several text classification tasks on different NLP datasets, including distillation and non-English data. SETFIT is compared to standard fine-tuning on few-shot training data and on entire data.

2 SETFIT - Sentence Transformer Fine Tuning

SETFIT uses a two-step training methodology, that begins with fine-tuning a Sentence Transformer (ST) and then training a classifier head. In the first step, the ST is fine-tuned to sentence pairs from the input data using a contrastive Siamese approach. This involves encoding the sentence pairs and adjusting the parameters of ST to minimize the distance between semantically similar pairs and maximize the distance between dissimilar pairs. In the subsequent step, a text classification head is trained with the encoded training data generated by the fine-tuned ST from the first step. This classification head learns to classify text based on the encoded representations provided by the ST. These two steps are visually represented in Fig. 3

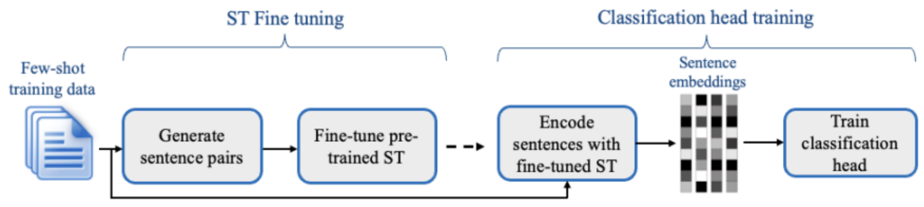


Fig. 3: Architecture of SETFIT.

2.1 Data generation for fine-tuning the Sentence Transformer

To fine-tune a Sentence Transformer model, we should generate sentence pairs/triple pairs labeled 1 (similar) or 0 (not similar). Due to the limited amount of labeled training data in a few-shot scenario, our goal is to generate additional training examples to augment the available data.

Formally, let us consider the case of a binary classification problem. Give a small collection of M labeled instances denoted as $D = \{(s_i, l_i)\}$, where s_i represents sentences and l_i represents their corresponding class labels. For each sentence s_i, we randomly choose a "positive" s_j where $l_i == l_j$ and a "negative" s_k where $l_i\ != l_k$. This produces two triples $\{(s_i, s_j, 1), (s_i, s_k, 0)\}$ that include randomly chosen sentences from same and different classes. The third component, which can be either 1 or 0, refers to similarity (1 stands for "similar and 0 for "not similar"). We repeat this process for each sentence for N iterations. To create the final contrastive fine-tuning dataset, we concatenate the positive and negative triplets across all class labels. In the case of a multi-class classification problem, we generate a triple for each class label for each sentence s_i. Assuming we have C classes, we would have one triple consisting of randomly selected sentences with the same label, and C - 1 triples containing randomly selected sentences with different labels.

By using this contrastive fine-tuning approach, we can effectively expand the size of the training data within few-shot scenarios. Assuming we have a small number (M) of labeled examples for a binary classification task, the size of the ST fine-tuning set T can be derived from the number of sentence pairs, which is equal to M*2*N. This may also include duplicate sentence pairs. The number of unique sentence pairs would be $M(M-1)/2$. This expanded set is larger than the original set M, providing additional instances for training and improving the learning process.

SETFIT changes the functionality of ST from sentence embedding to topic embedding as the data is generated based on the class labels. The idea behind this algorithm is to make sentences that have the same label more similar and sentences that have different labels less similar during the fine-tuning of the Sentence Transformer. This would make the sentence embeddings of one class linearly separable from the sentence embeddings of the other class in the embedding space.

2.2 Sentence Transformer

The main component of SETFIT is a Sentence Transformer. Sentence Transformers are large neural networks designed to optimize and generate high-quality sentence embeddings by relying on pre-trained Transformer-based models such as BERT and RoBERTA. They are fine-tuned on various downstream tasks to learn high-quality sentence representations. Sentence embeddings are vector representations that capture the semantic and contextual information of a sentence. They encode the meaning of a sentence in a fixed-dimensional vector and allow various natural language processing tasks such as text classification, semantic search, and clustering to be performed based on the sentence embeddings.

Let's take as an example a BERT based-Sentence Transformer SBERT. Unlike BERT, SBERT is fine-tuned on sentence pairs using a siamese architecture. We can think of this as having two identical BERTs in parallel that share the exact same network weights. Therefore during backpropagation we optimize the weights of one BERT. Suppose we have a set of sentence pairs where each pair is assigned one of the labels: 1-similar and 0-not similar. Given an sentence pair (sentence 1 and 2), in forward propagation after tokenizing the sentences, we feed sentence A into siamese BERT 1 (left) and sentence B into siamese BERT 2 (right) to embed fixed-sized token representations. Then in the pooling layer, mean aggregation, which was proved to have the best performance compared to max or CLS aggregation, is used to generate u and v which are the corresponding sentence embeddings for the sentence 1 and 2. We then concatenate the embeddings as follows: $(u, v, \|u - v\|)$, multiply by a trainable weight matrix $W \in \mathbb{R}^{3N \times K}$, where N is the sentence embedding dimension, and K is the number of labels. This means we are adding a softmax layer as follows: $\text{Softmax}(W^T(u, v, \|u-v\|))$ We optimize cross-entropy loss to update the model's weights. The architecture of SBERT is shown in Fig. 4.

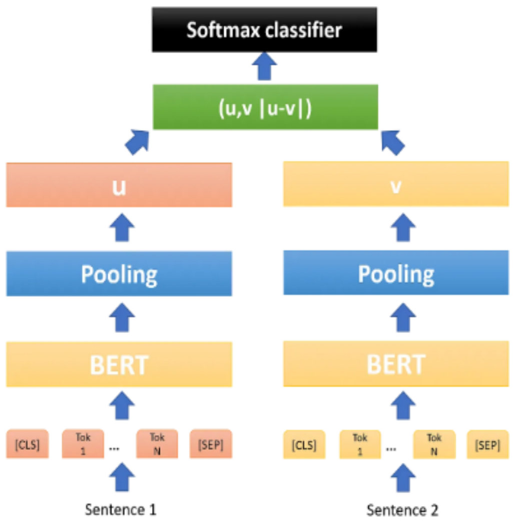


Fig. 4: Architecture of SBERT.

This approach results in pooled sentence embeddings for similar sentences (label 1) becoming more similar, and embeddings for dissimilar sentences (label 0) becoming less similar.

2.3 Classification head

In the second stage, the fine-tuned sentence transformer (ST) processes the original labeled training data denoted as $\{s_i\}$, resulting in a singular sentence representation for each training instance: $\text{Emb}_{s_i} = \text{ST}(s_i)$, where the function ST() signifies the fine-tuned ST. These representations, accompanied by their respective class labels, comprise the training set for the classification head: $D_{\text{CH}} = \{(\text{Emb}_{s_i}, l_i)\}$.A logistic regression model is utilized as the text classification head in case of a binary classification problem.

2.4 Prediction

During the inference phase, the fine-tuned ST encodes an unseen input sentence s_i and produces a sentence representation. Subsequently, the classification head, which was trained in the earlier phase, generates the class prediction for the input sentence based on its sentence representation. Formally, this process can be expressed as: $x_{\text{pred}_i} = \text{CH}(\text{ST}(s_i))$, where CH denotes the prediction function of the classification head.

2.5 Advantages of SETFIT

1. **State-of-the-art results:** SETFIT achieves state-of-the-art results in few-shot setup which makes its performance comparable with other SOTA tech-

niques like standard fine tuning, in-context learning, parameter-efficient fine tuning and pattern exploiting training.

2. **Faster to train:** SETFIT is orders of magnitude faster to train or perform inference compared to other SOTA techniques.
3. **No need of prompt engineering:** Pattern exploiting training and prominent PEFT methods require, as part of their training, the input of manually generated prompts, yielding varying outcomes depending on the level of manual prompt-engineering. On the contrary, SETFIT does not require any prompt design.
4. **Run zero-shot classification:** Using SETFIT we can also run zero-shot classification (when there is no labeled data available) by producing synthetic data for example: 'this sentence is negative' has the label 0 and 'this sentence is positive' has the label 1. This has been proven to result in a very good performance too.

2.6 Hard negative/positive sampling

Introduction My contribution to the original paper of SETFIT is as follows:

1. I propose a new technique for sampling the data in the data generation process which is used for fine-tuning the Sentence Transformer which I called hard negative/positive sampling. I demonstrate the effectivness of this approach and compare the performance with random sampling.
2. I make the code and data used in my work publicly available.

Data generation for ST fine-tuning using hard negative/positive sampling Formally, let us consider again the case of a binary classification problem. Give a small collection of M labeled instances denoted as $D = \{(s_i, l_i)\}$, where s_i represents sentences and l_i represents their corresponding class labels. For each sentence s_i and each iteration i= we sample "positive" s_j such that $l_i == l_j$ and cos_sim(s_i, s_j) is minimal. where cos_sim is a function which computes the cosine similarity of two vectors. Similarly, sample a "negative" s_k where $l_i \mathrel{!=} l_k$ and cos_sim(s_i, s_k) is maximal. This produces two triples $\{(s_i, s_j, 1), (s_i, s_k, 0)\}$ that includes sampled sentences from same and different classes. The third component which can be either 1 or 0 is related to the similarity (1 stands for "similar and 0 for "not similar"). To avoid sampling the same sentence pairs, I remove the already sampled s_j and s_k for the sentence s_i from the previous iteration. Like random sampling, we repeat this process for each sentence and for N iterations. To construct the final contrastive fine-tuning dataset, we concatenate the positive and negative triplets across all class labels. Similarly, in case of a multi-class classification problem, for each sentence s_i we generate a triple for each class label. Assuming we have C classes, we would have one triple which consists of randomly sampled sentences with same label and C - 1 triples which include randomly chosen sentences with different labels. The goal of this method is to choose sentences that are not similar at all but have the same label and therefore make them more similar during fine tuning of ST. Similarly we choose sentences that are very similar but have different labels and as a result make them less similar during fine tuning of ST. We sample the data in such a way that we can make some update or improvement of the ST during fine-tuning. This method works better than random sampling especially for less number of iterations (N) and greater number of training examples (M).

Advantages and limitations The benefit of employing hard negative/positive sampling is that it achieves comparable or superior performance compared to random sampling. However, it requires additional time to generate the samples due to the computation of sentence similarity against every sentence from the positive and negative classes. This process is essential for identifying the lowest or highest similarity value. Additionally, the proposed approach is limited by using the cosine similarity as loss function. Current implementation of SETFIT uses also by default cosine similiarity but it can be fine-tuned using other loss functions too. Furthermore, hard negative/positive sampling is implemented only for binary classification use-cases.

Visual example A simple example is shown in Fig. 5. For illustration purpose we are using eucledian distance as distance metric and two principal components after reducing the dimensions of the embeddings using TSNE. For a given sentence s_i, in the first iteration we sample s_j from the positive class since euc_dist(s_i, s_j) is maximal and s_k from the negative class since euc_dist(s_i, s_k) is minimal. euc_dist is a function that represents the eucledian distance. Then we remove s_j and s_k from the sampling set. In the second iteration we follow the same approach.

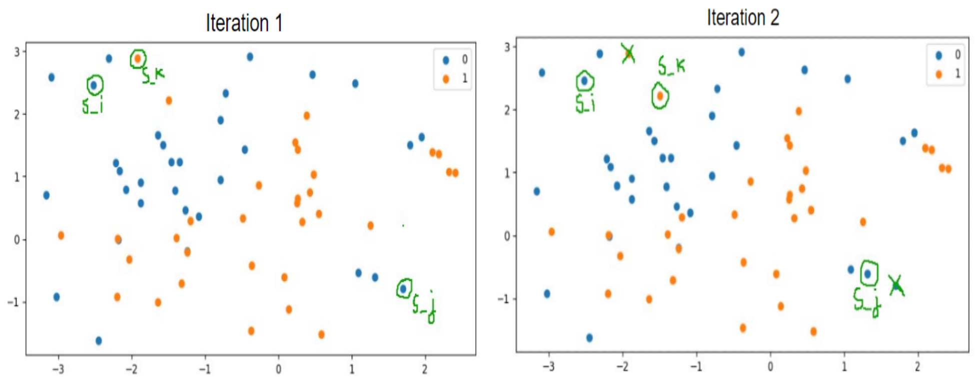


Fig. 5: Example of hard negative/positive sampling for a given sentence s_i using 2 iterations.

2.7 Experiments

Data I conduct experiments on 3 text classification datasets: SST2, SentEval-CR and Ade-corpus-v2. These datasets used are available on the Hugging Face Hub under the SETFIT organisation. I split the datasets into training and test datasets.

SETFIT Model As underlying Sentence Transformer of SETFIT I chose all-mpnet-base-v2 since it outperforms all the other models on average across several benchmarks and tasks. This ST model was trained on a large and diverse dataset of over 1 billion training pairs and it maps sentences to a 768 dimensional dense vector space.

Baselines I compare SETFIT's performance against the following models or approaches:

- Standard fine tuning of BERT on few-shot training data. We optimize all model's parameters using batches from a few training examples.
- Standard Fine Tuning of BERT using the entire training dataset. We optimize all model's parameters using batches from the entire dataset.
- SETFIT without fine tuning the ST. Instead of fine tuning the pre-trained sentence embeddings generated by the ST model, we train the classification head based on the pre-trained embeddings.

I also compare the performance of random sampling against hard negative/positive sampling.

Experimental Setup Since the performance of SETFIT and standard fine-tuning may be sensitive to the choice of few-shot training data, like the original paper I use 5 different random training splits for each dataset and sample size. Similar to the original paper, I use 2 different sample sizes of training data: M=18 and M=50 whereas the original paper uses M=8 and M=64. I use accuracy to evaluate the performance of each method on test dataset since the data is balanced and report the mean like the original paper. I fine-tune SETFIT using cosine-similarity loss, a learning rate of 0.00002, a batch size of 16 and epoch of 1. I also use the same learning, batch size and number of epochs for standard fine tuning. I set the same values for the hyperparameters across all methods and do not perform any hyperparameter tuning.

2.8 Results

Sentence Embeddings vs Topic Embeddings Let us with a simple example compare the pre-trained against the fine-tuned ST sentence embeddings as shown in Fig. 6. I plot two principal components after reducing the dimensions of the embeddings using TSNE for illustration purposes. Upper plots include the pre-trained embeddings of the training dataset (on the left) and fine-tuned embeddings (on the right) and similarly in the plots below we can see the embeddings from the test dataset. After fine tuning the pre-trained embeddings, the data becomes linearly separable. That's why training a simple classification head like Logistic Regression leads to good results. We call the fine-tuned sentence representation topic embedding since the fine-tuning data is generated based on the topics or the labels of the sentences.

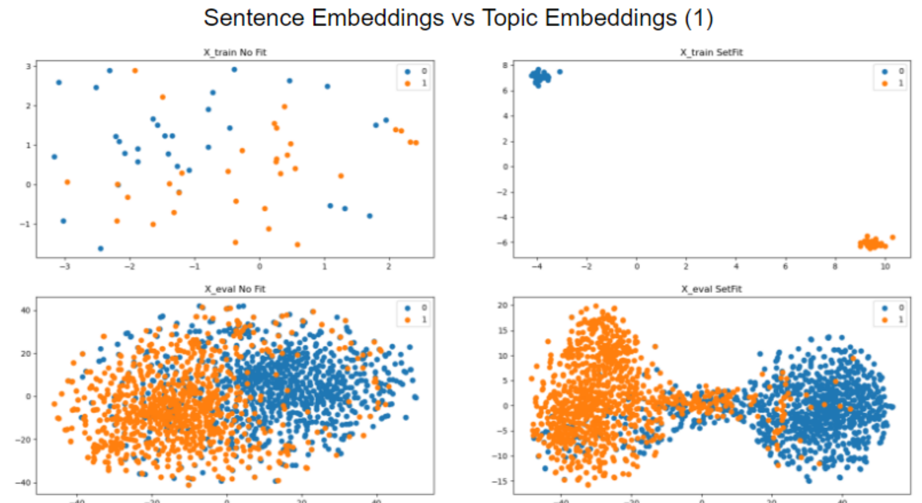


Fig. 6: Topic embeddings help the SETFIT model achieve higher performance on a text classification problem.

But that might not always be the case. On Sent-Eval CR data since there is too much overlapping between these classes, fine-tuning does not make the data linearly separable.

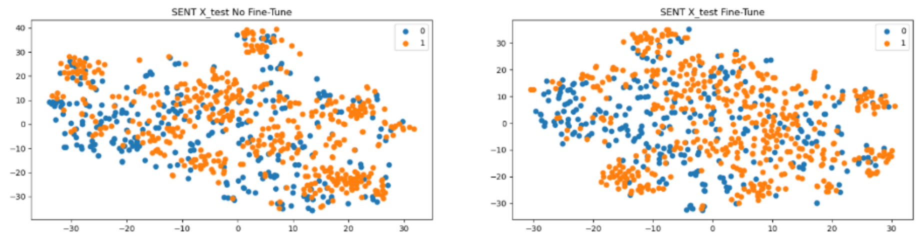


Fig. 7: On Sent-Eval CR dataset even after fine tuning there is overlapping between the classes.

Experimental results and comparison with original paper I find that SETFIT significantly outperforms the standard fine-tuning baseline for M = 18 by an average of 22.6 points(in the original paper by 19.3 points). However, as the number of training samples increases to M = 50, the gap decreases to 13.3 points (in the original paper by 5.6 points). Similarly, standard fine-tuning on entire dataset outperforms SETFIT on average by 5.7 points. For M=18, SETFIT with random sampling has similar performance compared to SETFIT with hard negative sampling. As the size of few-shot training data increases, hard negative/positive sampling outperforms random sampling on average by 1.1 points. SETFIT with fine tuning clearly outperforms SETFIT without fine tuning for M=18 as well as for M=50.

Method/Dataset	SST-2	SentEval-CR	Ade-corpus-v2	Average
M=18*				
Standard Fine-Tuning	50.1	38	49.8	46
SETFIT-random sampling(Fit)	89.1	37.7	79.1	68.6
SETFIT-hard negative sampling	89	37.6	78.8	68.4
SETFIT(No Fit)	83.2	38.3	71.4	64.3
M=50*				
Standard Fine-Tuning	70.3	50.2	62	57.2
SETFIT-random sampling(Fit)	90.3	37.7	83.5	70.5
SETFIT-hard negative sampling	93.2	37.7	84.4	71.6
SETFIT(No Fit)	85.6	38	84.8	69.5
M=Full**				
Standard Fine-Tuning	92.1	48.2	88.5	76.2

Table 8. SETFIT performance score compared to the baselines across 3 test datasets for three training set sizes. *Number of training samples per class. **Entire dataset

2.9 Conclusion

This work presents SETFIT, an innovative method for few-shot text classification. My research reveals that SETFIT achieves state-of-the-art results and comparable performance to similar approaches like standard fine tuning despite being smaller and as a result avoiding high computation cost. Moreover, SETFIT avoids the issues of manually designing high quality prompts. I also propose a new sampling method which achieves comparable or better performance compared to random sampling.

2.10 References

1. L. Tunstall et al, ”Efficient Few-Shot Learning Without Prompts”, Arxiv: 2209.11055, 2022
2. Tom B. Brown et al, ”Language Models are Few-Shot Learners”, Arxiv: 2005.14165, 2020