Human Language Techlogy Project: Sentimental Analysis task

Xiaoyan Li 627452

September 8, 2023

1 Introduction to the task: Sentimental Analysis

In recent years, there has been an explosive surge in the popularity of community portals across the internet. In addition to content curated by portal editors, user-generated content has become a pivotal component of numerous websites. Users contribute their insights not only through discussions and personal notes in various social web spaces (such as boards and blogs) but also on a large scale by leaving comments and reviews about products and services on various commercial websites. Despite the rapid proliferation of such content, its full potential remains untapped. Information provided by users often lacks organization, and many platforms that facilitate user input refrain from moderating the content added by users.

Sentiment analysis emerges as an endeavor to harness the immense volume of user-generated content effectively. It harnesses computational processing power to systematize the insights derived from user opinions and to subject them to analysis for subsequent utilization. The undertaking of sentiment analysis can be conceptualized as a text classification conundrum. This is because the process entails a series of operations that ultimately culminate in classifying whether a given text conveys a positive or negative sentiment.

Sentiments and opinions can be discerned primarily at three levels: the document level, sentence level, and aspect level. This particular project is specifically aimed at the sentence level. However, while pretrained large language models (LLMs) can offer valuable utility in this task, the challenge lies in effectively transferring the pretrained knowledge in the LLM models to the specific target domain. The gap in data and features distribution between the souce domain used in the pretrained phase for the LLMs and target domains is the major problem. For example, the BERT model is generally pretrained based on BookCorpus and English Wikipedia, which might degrade its performance in a certain specific domain. Consequentely, the need for finetuning LLMs becomes self-evident.

It is worth noting that supervised deep learning tasks are notoriously data-intensive. Fine-tuning all parameters in the LLMs slows down the training process significantly, concurrently making it susceptible to overfitting, particularly in the case of insufficient training data. This project aims at how to address the sentiment analysis task using pretrained LLMs, i.e., BERT, focusing on finetuning the model not just for parameter efficiency, but also for data efficiency.

2 Prior related work

2.1 Bert and prompt tuning

The BERT model [2] consists of a series of stacked Transformers encoders. Since its introduction by researchers at Google in 2018, it has pushed the boundaries of earlier model architectures, such as LSTM and GRU, which were either unidirectional or sequentially bidirectional. BERT considers context from both the past and future simultaneously, a contribution from the so-called "attention mechanism".

Prompt-tuning is an efficient, low-cost way of adapting an large model to target downstream tasks without retraining the model and updating its weights. Prompt-based methods have shown received great attention in few-shot setting, zero-shot setting, and even fully-supervised setting. Current prompt-based methods can be categorized as two branches according to the prompt is whether discrete or continuous.

- Discrete prompts: Discrete prompt is a sequence of words to be inserted into the input text, which helps the PTM to better model the downstream task. Sun [11] constructed an auxiliary sentence by transforming aspect-based sentiment analysis task to a sentence pair classification task, but its model parameters still need to be fine-tuned.
- Continuous prompts: Instead of finding the optimal concrete prompt, another alternative is to directly optimize the prompt in continuous space. The optimized continuous prompt is concatenated with word type embeddings, which is then fed into the pre-trained models. Typically, the model parameters are fixed which means the prompt tokens are also fixed by the model parameters.

This project mainly explored the continuous prompts methods. More specifically, it exprimented two types prompt tuning: the basic prompt tuning [4] and prefix prompting tuning [5]. In [4], sequence classification task is cased as a text generation task, in which the tokens that make up the class label are generated. Prompts are prepended to the input as a series of virtual tokens. While the model parameters are fixed and prompt tokens are also fixed, the embeddings of the prompt tokens are updated independently. Prefix tuning [5] is similar to prompt tuning in [4]. It also attaches a sequence of task-specific parameters to the beginning of the input, or prefix. Only the prefix parameters are optimised and trained, while the pretrained model parameters are kept frozen. The main difference is that the prefix parameters are added in **all** of the model layers, whereas prompt tuning [4] only adds the prompt parameters to the embedding of the model input.

2.2 Deep metric learning

Conventional deep neural networks utilize a simple linear classifier to make decisions on examples. The feature embeddings learned in this case are linearly separable in the feature space. Nevertheless, these methodologies typically require a large amount of data to achieve successful results.

Deep metric learning(dml) is an approach based directly on a distance metric that aims to establish the similarity or dissimilarity between data points. As a result, objects with the same labels are drawn closer, while objects with different labes are pushed apart. This leads to a more compact representation of data points, and thereby enabling robust performance even in low-data settings.

In this project, we will makes use the combination of the following two distance-based losses:

- Triplet loss: Triplet Loss was first introduced in FaceNet [9] and it has been one of the most popular loss functions for supervised similarity or metric learning ever since. Triplet Loss encourages that dissimilar pairs be distant from any similar pairs by at least a certain margin value.
- DisMax Loss: DisMax loss [6] was proposed to improve the out of distribution detection performance by encouraging clustering in a feature space. It is used in conjunction with crossentropy loss and encourages an input to minimise distance to its ground truth class centre, and maximise distances to the other class centres. The class centres are learnt simultaneously with the feature embedding during training.

In this project, we aim to integrate deep metric learning with prompt tuning to examine the potential enhancement that distance-based losses can bring to prompt tuning methods within a low-data setting.

3 Dataset description and preprocessing

The dataset utilized in this project is derived from the Stanford Sentiment Treebank (SST) corpus, which consists of individual sentences extracted from movie reviews, totalling 11,855 sentences. These sentences within the treebank were partitioned into train (8544), development (1101), and test splits (2210). Additionally, same as [10], we conducted an analysis focusing solely on positive and negative sentences, while ignoring the neutral class. This led to a reduction of approximately 20% of the data, resulting in the three sets containing 6920, 872, and 1821 sentences, respectively. .

One distinctive feature of this dataset is that it has undergone parsing using the Stanford parser. This parsing process involved labelling every individual phrase found within the branches of the parse trees for all the sentences in the corpus, resulting in the labelling of a total of 215,154 unique phrases. This means that it can be used for training models at both the phrase and sentence levels.

However, the project's primary focus is on efficiently tuning a large language model, especially in low-datasetting. Working at the phrase level requires significantly more computational resources, including memory and GPU power, which may not be readily available. Therefore, I decided to focus solely on the sentence level and disregard all parsing tree information. Figure 1 shows what some of the labelled sentences might look like.

sentence	label
a stirring , funny and finally transporting re imagining of beauty and the beast and 1930s horror films	1
apparently reassembled from the cutting room floor of any given daytime soap	0
they presume their audience won't sit still for a sociology lesson	0
this is a visually stunning rumination on love , memory , history and the war between art and commerce	1
jonathan parker 's bartleby should have been the be all end all of the modern office anomie films	1

Figure 1: Examples of SST2 datasets, credits to [1]

4 Architecture

Our goal in the project is to create a model that takes a sentence as input and produces either 1 (indicating the sentence carries a positive sentiment) or a 0 (indicating the sentence carries a negative sentiment). The model is made of the following three components:

1. Tokenization: The first step is to take a sentence as input and tokenize it using a tokenizer. Tokenization involves breaking the sentence into individual tokens (words or subwords) and representing them as tokens that correspond to entries in an embedding table. After tokenization, we have sequences of token IDs for each sentence.
2. Feature Extraction Model: These token IDs are then fed into another secondary model, which extracts meaningful features from the sequences of token IDs.
3. Classifier: The features extracted by the secondary model are then used as inputs to a classifier. This classifier is responsible for determining the sentiment polarity of the sentence. It assigns a label of either 1 (positive sentiment) or 0 (negative sentiment) based on the learned features.

This project involved an exploration of diverse BERT-based model variants for the tasks of tokenization and feature extraction. Tokenizer parameters remained constant throughout this project, with exclusive attention directed towards parameter tuning in subsequent stages of the model pipeline, particularly within the feature extraction and classification models.

The BERT variants examined in this project encompassed the following models: BERT-base, RoBERTa-base, DeBERTa-base, as well as their larger iterations, specifically BERT-large, RoBERTa-large, and DeBERTa-large.

The features extracted of the BERT model variants are then go through a dropout layer, followed by a binary classifier. As for the classifier, two distinct approaches were experimented:

- The baseline is a linear layer, which which is essentially a fully connected layer of size(768,2) for the BERT-base, RoBERTa-base, DeBERTa-base models, and of size (1024,2) for their large counterparts. The classifier is trained using the usual softmax loss, which is a standard approach for multi-class classification. In this case, it's adapted for binary classification, with two output classes.
- The second approach is inspired by the prototype-based classification. Here the trainable parameters are the class centers, one for each class (positive and negative sentiment). Each class center is represented as a vector of size 768(or 1024 for large version models).

  The training objective is twofold:

  1. Minimize the distance between the features extracted from the BERT variants and the class centre corresponding to their ground truth class. In other words, make the features as similar as possible to the correct class centre.
  2. Maximize the distance between the features and the class centre of the opposite class. This ensures that the features are as dissimilar as possible to the centre of the incorrect class.

  The objective function can be formally written as:

  $$L_{Dis\_Max}(y^k|x) = -log\frac{exp(-\|f_\theta(x) - C_k\|}{\sum_j exp(-\|f_\theta(x) - C_j\|}$$

  where $\|\|$ represents the 2-norm, and C_i represent the class center of the $i-th$ class.

  To further enhance the result, the DisMax loss is combined with the Triplet loss. Let x_a, x_p, x_n be some samples from the dataset. Usually, x_a is called an anchor sample, x_p is called a positive sample if it has the same label as x and x_n is called a negative sample if it has a different label. For some distance on the embedding space d, the loss of a triplet (x_a, x_p, x_n) is:

  $$L_{Triplet} = max(d(x_a, x_p) - d(x_a, x_n) + margin, 0)$$

To minimize this loss, $d(x_a, x_p)$ is pushed to 0, and $d(x_a, x_n)$ is pushed to be greater than $d(x_a, x_p) + margin$.

The total loss is:

$$L = L_{Dis\_Max} + L_{Triplet}$$

This dual loss strategy can be effective for improving the performance of the model, as it combines the benefits of better class separation and feature space organization The DisMax loss helps in refining the decision boundary for classification, while the Triplet loss aids in creating more discriminative feature representations.

Tuning approaches To transfer the knowledge of the large pre-trained language models to our specific sentimental analysis task, it is necessary to tune the model to be adapted to our specific data domain.

we employ two prompt-based tuning methods: the basic prompt tuning, which only inserts the prefix parameters into the input layer, and the prefix tuning, which inserts the prompt parameters into both the input layer and all hidden layers. The architecture for model prompt tuning and prefix tuning are shown in Figure 2 and Figure 3. In the basic prompt tuning, the prefix parameters are essentially added to the input of the model to guide its behaviour and adaptation to our specific task. This method is less parameter-intensive, as it focuses on modifying the input layer. In contrast, prefix tuning is a more comprehensive approach. Here, the prompt parameters are not only inserted into the input layer but also into all hidden layers of the pre-trained model. This means the prompts influence the model's behaviour at various stages of its architecture and thus involve a greater number of parameter tuning compared to basic prompt tuning.

Both of these methods are aimed at transferring knowledge from the large pre-trained language model to our specific task while allowing for task-specific adaptations. They differ in the extent to which they modify the model's architecture and parameters.

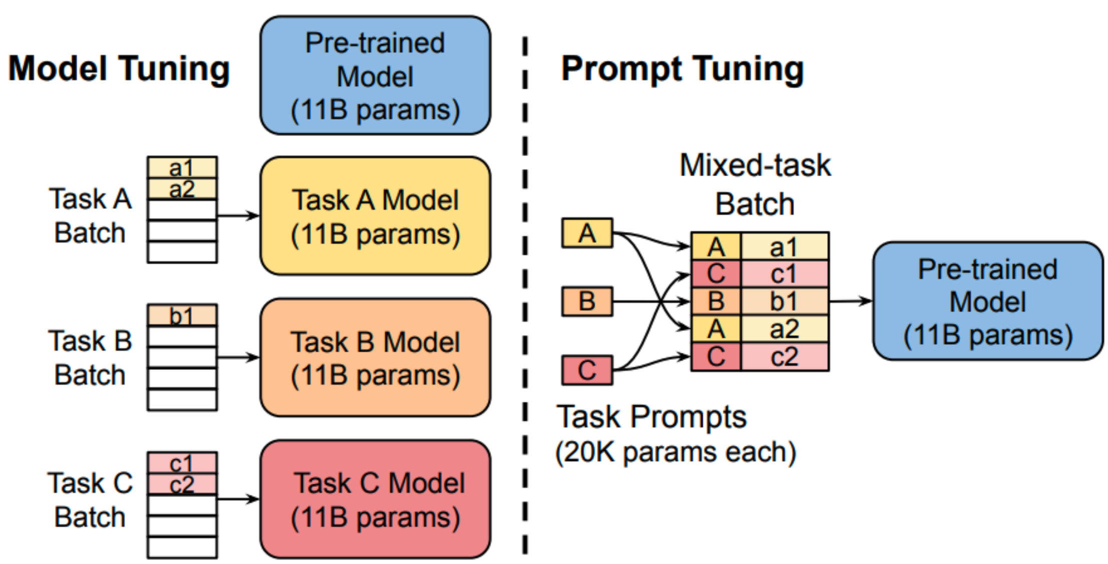

Figure 2: Architecture of prompt tuning, credit to [4]

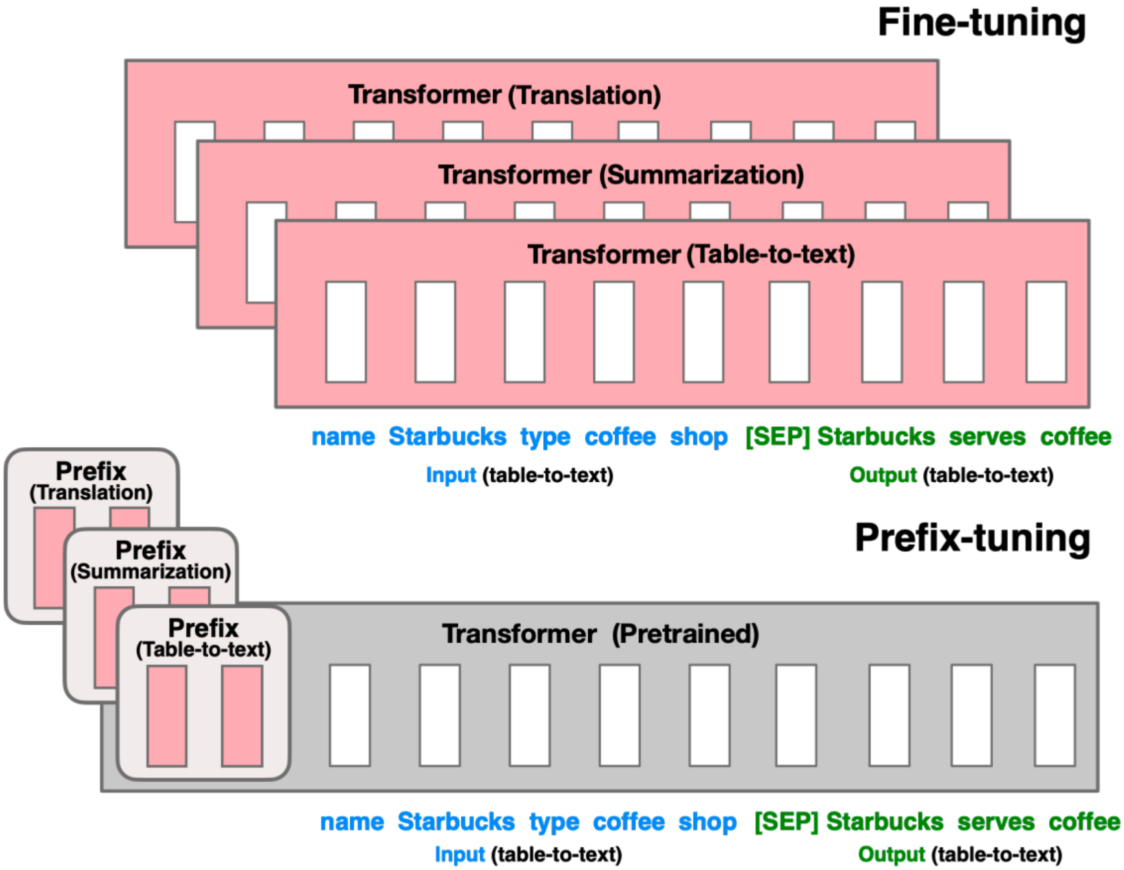

Figure 3: Architecture of prefix tuning, credit to[5]

5 Experimental setup

The project is implemented with Python language, and with the help of the Prompt Engineering for Transformers(PEFT) library, and the Pytorch deep metric learning library. PEFT [7] contains parameter-efficient finetuning methods for training large pre-trained models, while Pytorch metric learning [8] allows the easiest way to use deep metric learning in our application.

5.1 Datasets and Metrics

The model's performance is evaluated using the SST2 dataset. Due to limitations in computational resources, a total of 4000 samples were drawn from the total training set(6920 samples), with 2000 samples from class 0 (negative) and 2000 samples from class 1 (positive), ensuring balanced classes. The complete development set, consisting of 868 samples, was used for fine-tuning hyperparameters. This ensures that our model's hyperparameters are tuned effectively. The entire test set, comprising 1821 samples, was used to evaluate the model's performance on unseen data, providing a comprehensive assessment of how well the model generalizes.

To explore model performance in a low-data setting, 200 samples were randomly selected from the training dataset for training. This allows us to assess how well our model can perform when training data is limited. Similarly, 200 samples were selected from the development set for validation in the low-data setting experiment. As previously, the evaluation was conducted using the complete test set to have a comprehensive assessment of the generalization capability of the model. Furthermore, the results were measured by averaging the outcomes obtained from sampling with three different seeds to ensure result reliability and account for randomness in data sampling and model initialization.

Given that the task is fundamentally a classification task (sentiment analysis), accuracy was chosen as the metric criterion for assessing the models. Accuracy measures the proportion of correctly classified samples, providing a straightforward measure of model performance.

This evaluation approach allows us to thoroughly assess the model's performance under various conditions, including standard and low-data settings.

5.2 Hyperparameters

During the training phase, we use the AdamW optimizer and a linear learning rate scheduler, as suggested by the Hugging Face default setup. The hyperparameters subject to tuning include the number of epochs, batch size, learning rate, and prefix length. In the Tables 1 to 3, we report hyperparameters used to train the models documented in the following result section.

methods	**learning rate**	**num_epochs**	**batch size**	**num_virtual tokens**
	sufficient data setting	w/o deep metric learning		
fine-tuning	1e-5/1e-5	11/17	16/32	—
prompt tuning	1e-2/6e-3	11/38	16/16	7/7
prefix tuning	5e-3/1e-2	13/32	32/16	10/10
	low-data setting	w/o deep metric learning		
fine-tuning	1e-5/1e-6	13/12	8/8	—
prompt tuning	4e-2/4e-2	61/69	8/8	7/7
prefix tuning	1e-2/1e-2	5/24	8/8	10/10

Table 1: Hyperparameter settings for BERT-base-uncased model

5.3 Results

benchmarks: The benchmarks for comparison in our experiment are presented in Table 4. In these benchmarks, models were trained and evaluated using identical datasets.

Tables 5 to 7 demonstrated the results we obtained from the base models, namely, BERT base, RoBERTa-base, DeBERta-base, using different tuning approaches. Please note that only the number of parameters for tuning the BERT base model was reported, while the number of parameters for the linear classifier was not taken into account.

[1] This model does not support past key values which are required for prefix tuning
[2] This model does not support past key values which are required for prefix tuning
[3] This model does not support past key values which are required for prefix tuning
[4] This model does not support past key values which are required for prefix tuning

methods	**learning rate**	**num_epochs**	**batch size**	**num_virtual tokens**
	sufficient data setting	w/o deep metric learning		
fine-tuning	1e-5/1e-5	15/22	32/16	—
prompt tuning	5e-2/2e-2	27/46	32/32	7/7
prefix tuning	1e-2/1e-2	10/32	16/16	10/10
	low-data setting	w/o deep metric learning		
fine-tuning	1e-5/1e-5	16/32	16/10	—
prompt tuning	5e-2/2e-2	88/92	8/8	7/7
prefix tuning	1.5e-2/1e-2	29/24	8/32	10/10

Table 2: Hyperparameter settings for RoBERTa-base model

methods	**learning rate**	**num_epochs**	**batch size**	**num_virtual tokens**
	sufficient data setting	w/o deep metric learning		
fine-tuning	1e-5/1e-5	4/6	32/32	—
prompt tuning	5.2e-2/2.2e-2	21/17	32/32	7/7
prefix tuning [1]	—	—	—	—
	low-data setting	w/o deep metric learning		
fine-tuning	1e-5/1e-5	50/77	32/16	—
prompt tuning	5e-2/2e-2	8/50	8/8	7/7
prefix tuning[2]	—	—	—	—

Table 3: Hyperparameter settings for DeBERTa-base model

We can see that when working with an abundance of data for training and validation, fine-tuning yields the best results. However, it comes at the cost of a significant number of trainable parameters. Prompt Tuning achieves competitive accuracy while utilizing a minimal number of trainable parameters. It appears to be an efficient method for achieving good results with fewer modifications to the model. Prefix tuning also performs well in terms of accuracy but requires more trainable parameters compared to prompt tuning.

However, in the low-data setting, fine-tuning remains effective but with a noticeable decrease in accuracy due to overfitting. Prompt tuning continues to perform well and efficiently, and prefix tuning shows competitive results as well. *It is worth noticing that deep metric learning can effectively enhance the result in this case, when used in conjunction with prompt tuning.* As the model size increases, this effect becomes more pronounced, which is further evident when testing with larger models. Table 8 presents the result for the RoBERTa-large model and DeBERTa-large model.

Notably, we observe that prompt tuning for DeBERTa-large achieved an impressive accuracy of 92.31% with only approximately 7,000 trainable parameters, representing a minimal proportion (0.0018%) of the total parameters, despite having just 200 training samples. Remarkably, it even surpasses the performance of fine-tuning all parameters of a BERT-base model trained with the entire available

models	BERT base	BERT large	RoBERTa base	RoBERTa large
accuracy	91.8	93.1	93.4	94.9

Table 4: Benchmarks for our results[3]

methods	accuracy w/o deep metric learning	#trainable parameters(% proportions)
	sufficient data setting	
fine-tuning	90.94%/91.05%	109M (100%)
prompt tuning	89.40%/89.51%	5K (0.0049%)
prefix tuning	90.28% /91.05%	184k (0.16%)
	low data setting	
fine-tuning	85.67%/85.75%	109M(100%)
prompt tuning	**85.23**%/78.80%	5K (0.0049%)
prefix tuning	86.44%/84.90%	184k (0.16%)

Table 5: Results for BERT-based-uncased model

dataset.

6 Limitations and directions for improvement

In this project, we have focused on the challenges posed by the low-data setting and explored how prompt tuning can be enhanced by deep metric learning to achieve results comparable to scenarios with sufficient data. However, it should be noted that even though we have made decent progress in improving the performance, the linear classifier layer, serving as the final layer of classification, still encompasses a considerable number of parameters. In cases where we delve further into data scarcity, such as few-shot learning scenarios, the linear classifier layer could potentially become a bottleneck.

To tackle this issue, we propose considering parameter-free classification techniques. More specifically, we suggest a two-step training approach:

- In the first step, our focus is solely on training the model to extract features using deep metric learning methods or contrastive learning methods. At the end of this step, the learned features should exhibit proximity for examples of the same label.
- In the second step, we can leverage tools like sklearn's NearestCentroid or k-nearest neighbours (knn) to classify unseen examples.

This two-step process holds promise for addressing challenges related to data scarcity and potentially reducing the parameter dependency of the linear classifier layer.

methods	accuracy w/o deep metric learning	#trainable parameters(% proportions)
	sufficient data setting	
fine-tuning	93.62%/93.63%	124M (100%)
prompt tuning	93.36%/93.35%	5K (0.0043%)
prefix tuning	93.30% /93.62%	184k (0.15%)
	low data setting	
fine-tuning	86.10%/85.72%	124M(100%)
prompt tuning	**89.84**%/85.83%	5K (0.0043%)
prefix tuning	88.41%/87.86%	184k (0.15%)

Table 6: Results for RoBERTa-base model

methods	accuracy w/o deep metric learning	#trainable parameters(% proportions)
	sufficient data setting	
fine-tuning	92.64%/93.08%	138M (100%)
prompt tuning	93.08%/92.86%	5K (0.0039%)
prefix tuning[3]	—	—
	low data setting	
fine-tuning	86.16%/86.10%	138M(100%)
prompt tuning	**90.11**%/86.66%	5K (0.0039%)
prefix tuning[4]	—	—

Table 7: Results for DeBERTa-base model

methods	accuracy w/o deep metric learning	#trainable parameters(% proportions)
	RoBERTa-large	
fine-tuning	90.94%/90.77%	355M (100%)
prompt tuning	92.48%/91.04%	10K (0.0029%)
	DeBERTa-large	
fine-tuning	86.82%/86.77%	405M(100%)
prompt tuning	**92.31**%/90.00%	7K (0.0018%)

Table 8: Results for RoBERTa-large and DeBERTa-large model in the low-data setting

References

[1] Jay Alammar. A Visual Guide to Using BERT for the First Time. `https://jalammar.github.io/a-visual-guide-to-using-bert-for-the-first-time`, 2018.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[3] Yuxin Jiang. Use BiLSTM_attention, BERT, ALBERT, RoBERTa, XLNet model to classify the SST-2 data set based on pytorch. `https://github.com/YJiangcm/SST-2-sentiment-analysis/tree/master`, 2020.

[4] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[5] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, August 2021. Association for Computational Linguistics.

[6] David Macêdo, C. Zanchettin, and Teresa B Ludermir. Distinction maximization loss: Efficiently improving out-of-distribution detection and uncertainty estimation by replacing the loss and calibrating. 2022.

[7] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, and Sayak Paul. Peft: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`, 2022.

[8] Kevin Musgrave, Serge J. Belongie, and Ser-Nam Lim. Pytorch metric learning. *ArXiv*, abs/2008.09164, 2020.

[9] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[10] 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.

[11] Chi Sun, Luyao Huang, and Xipeng Qiu. Utilizing BERT for aspect-based sentiment analysis via constructing auxiliary sentence. 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 380–385, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.