FRI Natural language processing course 2024
Project report

University *of Ljubljana*
Faculty *of Computer and Information Science*

Parameter-Efficient Fine-Tuning of Large Language Models

Ondřej Komín, Andrej Sušnik, Eileen Vu

Abstract

In this paper we present different parameter efficient fine tuning methods, i.e. LoRA, Soft prompts tuning, IA3 and Bitfit, and apply them to various NLP tasks. Our evaluation reveals that Soft Prompts tuning, in certain tasks, can surpass full fine-tuning while requiring less training time. Additionally, we observe that LoRA finds a good balance between performance and training time across different tasks.

Keywords

PEFT, LoRA, Prompt tuning, IA3, BitFit, CommonsenseQA, SloSuperGlue, CoNLL-2012, SST5

Advisors: Boshko Koloski, Slavko Žitnik

Introduction

Since the introduction of attention [1], using large language models (LLMs) such as Google's BERT [2] and OpenAI's GPT [3] has become inevitable to various applications across natural language processing (NLP) domains. These models have demonstrated remarkable capabilities in understanding and generating human-like text. However, to achieve optimal performance in specific tasks, fine-tuning these pre-trained models on task-specific data is often necessary.

This research paper focuses on presenting and comparing various parameter-efficient fine tuning (PEFT) methods in the context of optimizing natural language processing tasks. Through empirical experiments, we aim at exploring the effectiveness of different PEFT approaches in achieving desirable trade-offs between model complexity, computational efficiency, and task performance. Similarly to the work done in [4], we begin our research by reviewing and categorizing popular PEFT methods. We continue by presenting and discussing the theoretical foundations of four specific methodologies and applying these to four different datasets that cover various natural language understanding skills. Lastly, the fine-tuned models will be evaluated based on appropriate performance metrics, computational resources required, and ease of adaptation to different tasks. Through this exploration, we aim at facilitating more resource-conscious approaches to model optimization, accelerating progress in the field of NLP and its applications.

Related Work

Fine-tuning LLMs plays a crucial role in adapting these models to domain-specific tasks. As LLMs are pre-trained on vast amounts of text data, they capture linguistic patterns and semantic information. However, for tasks with specific requirements, such as sentiment analysis or named entity recognition, fine-tuning allows these models to tailor their representations to better suit the task at hand. Traditional methods of full fine-tuning involve updating all parameters of the pre-trained LLM. The popular GPT-4 model released in early 2024 contains 1.76 trillion weights [5]. Fine-tuning (and storing) this amount of parameters whenever one wants to apply the model to a specific use case however not only requires significant computational resources but also poses a risk of overfitting, especially in scenarios with limited task-specific data.
In order to avoid these problems, PEFT methods have emerged as a solution to the drawbacks of full fine-tuning. By reducing the number of parameters updated during fine-tuning, PEFT not only mitigates computational costs but also helps alleviate overfitting concerns. As described in [4], PEFT techniques can be divided into five main categories: additive fine-tuning, partial fine-tuning, reparameterized fine-tuning, hybrid fine-tuning and lastly unified fine-tuning. While some of these methods aim at introducing new trainable parameters for use-case-specific fine-tuning, others reduce the number of trainable parameters by transforming the weights into lower dimensions.

In this paper, we focus on presenting and comparing four PEFT methods in the context of different NLP tasks. Specifically, we investigate the performance of the following methodologies: low rank adaptation (LoRA), soft prompt-based fine-tuning, reparameterized fine-tuning and partial fine-tuning. **LoRA** [6] has become very popular in the last years due to its ability to reduce the number of parameters without introducing additional latency, unlike for example adapter methods. This lowers training computational requirements while improving performance for specific NLP tasks. **Soft-prompting** [7] is a machine learning technique that offers subtle guidance to models during training, aiding in learning without the need for explicit labels. This approach is valuable as it allows for more flexible decision-making while still achieving desired outcomes, especially in scenarios where labeled data may be scarce or costly to obtain. **IA3** [8] is a reparameterized fine-tuning technique for pretrained language models. It rescales inner activations with learned vectors, reducing the number of trainable parameters while optimizing model performance for task-specific data. Lastly, we will investigate the partial fine-tuning methods **BitFit** [9]. This method only fine-tunes the bias term of the layers while freezing the rest of the network. This technique, which trains less than 0.1% of the total number of weights, was proven to achieve comparable performance than full fine-tuning.

Benchmark	NLP Task
CommonsenseQA [10]	Commonsense Reasoning
CoNLL-2012 [11]	Coreference Resolution
SST5 [12]	Sentiment Analysis
Slovene SuperGLUE [13]	Slovene BoolQ (Boolean Questions)
XSum [14]	Text Summarization

Table 1. Chosen benchmarks for performance evaluation

We will provide an empirical comparison of these three methodologies based on four different NLP tasks[1]. The benchmarks we have chosen for this each represent distinct natural language understanding skills allowing us to provide a comprehensive overview of the advantages and disadvantages of all techniques.

Methods

PEFT methods can be grouped into five main categories. Additive fine-tuning introduces new trainable parameters for task-specific adaptation, including adapter-based fine-tuning, soft prompt-based fine-tuning, and others. Partial fine-tuning reduces the number of fine-tuned parameters by focusing on critical pre-trained parameters, with methods like bias update, pretrained weight masking, and delta weight masking. Reparameterized fine-tuning utilizes low-rank transformation to decrease trainable parameters, through techniques like low-rank decomposition and LoRA derivatives. Hybrid fine-tuning combines multiple PEFT approaches to leverage strengths and mitigate weaknesses, either manually or automatically. Unified fine-tuning provides streamlined frameworks for incorporating diverse fine-tuning methods into cohesive architectures, emphasizing consistency and efficiency across model adaptation without combining multiple methods.

[1]XSum not implemented

In this analysis, we will focus on four of these methods:

LoRA LoRA [6] is a method designed for fine-tuning large models. It operates by fixing the weights of the original model and introducing a trainable low-rank decomposition matrix into the LLM architecture. This modified architecture involves fixing the original weights W_0 while introducing additional trainable weights ΔW, which can be decomposed into matrices BA, where the rank of both B and A is much smaller than that of W_0. Consequently, the resulting weights are formulated as $W0+BA$, significantly reducing the number of parameters that need to be trained.

Empirical results indicate that LoRA performs comparably or even better than other methods, while requiring a comparable or lower number of trainable parameters. Notably, LoRA drastically reduces the number of trainable parameters, such as in the case of fine-tuning the GPT-3 model, where the parameter count was reduced by 10000 times, accompanied by a threefold reduction in GPU memory requirement. Additionally, LoRA exhibits several benefits, including the ability to train specialized models without introducing latency, as seen in adapter methods.

Moreover, it enables the deployment of multiple specialized models simultaneously by reducing memory and computational footprint, achieved through maintaining fixed weights for the base model while having several trained decomposition matrices for each specialized task.

LoRa model is trained with configuration showed in Listing 1.

Listing 1. LoRa parameters

```
lora_config = LoraConfig(
        r=16,
        lora_alpha=32,
        lora_dropout=0.1,
        bias="lora_only",
        task_type="SEQ_CLS"
)
```

Prompt Tuning Prompt tuning is an advanced technique in the field of natural language processing (NLP) that involves fine-tuning the prompts given to pre-trained language models to optimize their performance for specific tasks. Unlike traditional model training, which may involve adjusting vast numbers of parameters, prompt tuning focuses on crafting or adjusting the input prompts in a way that elicits more accurate or relevant responses from the model. This approach leverages the existing capabilities of large language models, allowing for efficient task-specific adaptation with minimal computational resources.

Mathematically, consider a pre-trained language model f_θ with parameters θ. Given an input sequence x and a task-specific prompt p, the output is generated as:

$$y = f_\theta(p,x)$$

where p is designed or tuned to optimize the model's performance on a particular task. The goal is to find an optimal prompt p^* that maximizes the performance metric $\mathscr{M}$ on a validation set D_{val}:

$$p^* = \arg\max_p \mathscr{M}(f_\theta(p,x),y) \quad \text{for} \quad (x,y) \in D_{\text{val}}$$

By carefully designing prompts, practitioners can guide the model to generate desired outputs, improve performance on diverse tasks, and enhance the interpretability and controllability of AI systems.

BitFit Bias-terms Fine-tuning (BitFit) [9] is a parameter-efficient fine-tuning technique for pretrained language models that focuses on updating only the bias terms of the model's weights. This approach aims to reduce the computational and memory resources required for fine-tuning while maintaining performance on downstream tasks.

In BitFit, instead of updating all parameters $\mathbf{W}$ and $\mathbf{b}$, we update only the bias $\mathbf{b}$. The weights $\mathbf{W}$ remain fixed. During fine-tuning, the gradients are computed with respect to $\mathbf{b}$ only, and the updates are applied as follows:

$$\mathbf{b} \leftarrow \mathbf{b} - \eta \frac{\partial \mathscr{L}}{\partial \mathbf{b}}$$

where η is the learning rate and $\mathscr{L}$ is the loss function.

BitFit has three key properties: it can match the results of a fully fine-tuned model, it enables tasks to arrive in a stream without requiring simultaneous access to all datasets, and it fine-tunes only a small portion of the model's parameters. Specifically, BitFit trains less than 0.1% of the total number of parameters, yet it achieves transfer learning performance comparable to, and sometimes better than, fine-tuning the entire network.

Infused Adapter by Inhibiting and Amplifying Inner Activations (IA3) The IA3 approach rescales inner activations with learned vectors. These learned vectors are injected in the attention and feedforward modules in a typical transformer-based architecture. These learned vectors are the only trainable parameters during fine-tuning, and thus the original weights remain frozen. Dealing with learned vectors (as opposed to learned low-rank updates to a weight matrix like LoRA) keeps the number of trainable parameters much smaller.

Similar to LoRA, IA3 offers several advantages: it efficiently reduces the number of trainable parameters, with IA3 models typically having only about 0.01% trainable parameters for base models like T0, compared to over 0.1% for LoRA. Additionally, IA3 maintains frozen pre-trained weights, allowing for the creation of multiple lightweight and portable models for various tasks. Despite its parameter efficiency, models fine-tuned using IA3 demonstrate performance comparable to fully fine-tuned models, without introducing any inference latency.

Training Pipeline

The training pipeline consists of five main steps, each applicable for the successful fine-tuning of the models for specific tasks.

Step 1 Initially, we imports necessary libraries and set up the environment. Using custom dataset handlers, we load the datasets based on the specified task.

Step 2 After loading the dataset, the we apply pre-processing functions tailored to the BERT tokenizer. This step involves tokenizing and encoding the text data, which is essential for input to the BERT model. The pre-processed data is split into train, validation, and test sets, and formatted as PyTorch tensors for compatibility with the BERT model.

Step 3 Next, we configure the training arguments, including parameters such as output directory, evaluation strategy, learning rate, batch size, and number of epochs. It also initializes the BERT model for the task and defines the trainers for the different methods. Each trainer is associated with its specific model path and training configurations.

Listing 2. Training Arguments

```
args = TrainingArguments(
    output_dir=model_path,
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=64,
    per_device_eval_batch_size=64,
    auto_find_batch_size=True,
    num_train_epochs=20,
    weight_decay=0.01,
)
```

Step 4 With the trainers defined, we execute the training process for all PEFT methods. This step involves fine-tuning the BERT model on the data using the specified training arguments and data.

Step 5 Finally, we evaluate the trained models using the evaluation datasets. Performance metrics such as accuracy, precision, F1-score, and recall are computed for the different methods. The results are compared to assess the effectiveness of each fine-tuning approach on specific tast.

Environment and reproducibility

We ran the experiments on the Arnes cluster, the compute node that we used included AMD EPYC™ 9124 CPU, 256GB of RAM and NVIDIA H100 GPU. The cluster uses SLURM middleware for it's job managment and we used sbatch command to run our jobs. To have reproducible environment we used containerization with Apptainer. Apptainer is a containerization solution mainly used in high performance computing. We used Python version 3.12 and we have gathered all the requirements into requirements.txt file and specified the versions of the packages. The source code is available in the repository[2]. Results can be reproduced by logging into the supercomputer cluster of your choice, cloning the repository and executing following commands.

Listing 3. Commands for running the training and evaluation

```
cd src
sbatch run.sh
```

Sbatch command will put job in the SLURM queue. After the job is accepted Apptainer container with the environment will be built. When the environment is built, the main script will be executed inside the containerized environment, firstly all the models will be trained and the evaluated.

Results

CommonsenseQA The CommonsenseQA dataset is a benchmark designed to evaluate AI systems on commonsense reasoning. It consists of multiple-choice questions, each with five answer options and one correct answer. The questions cover various aspects of commonsense knowledge, such as physical properties, social behaviors, causal relationships, and temporal and spatial reasoning.

The CommonsenseQA dataset consists of multiple-choice questions, each with five answer options (labeled A to E), one of which is correct. The correct answer is annotated for each question. The dataset is designed to challenge AI models to develop a deeper understanding of the world, similar to human commonsense reasoning. It is used to evaluate natural language understanding models, develop and benchmark new AI algorithms, and study AI's limitations and capabilities in understanding everyday scenarios. The CommonsenseQA dataset presents challenges such as handling ambiguity, context-dependence, and requiring complex reasoning.

We finetuned the mdeberta-small model. We can see that when using LoRa PEFT method we can achieve almost the same performance than with full finetuning while requiring much less training time. The BitFit method does not give a good result for this dataset.

Metric	FFT	LoRA	BitFit
Batch Size	64	128	128
Training Time [sec]	5798	4523	4539
RAM Usage [MB]	1998	2002	2005
# Parameters [$\times 10^6$]	14	0.03	0.033
Accuracy	0.66	0.65	0.42
Precision	0.66	0.65	0.42
F1-Score	0.66	0.65	0.42
Recall	0.66	0.65	0.43

Table 2. Performance of Full Finetuning vs PEFT methods on CommonsenseQA

[2]https://github.com/UL-FRI-NLP-2023-2024/ul-fri-nlp-course-project-naturallylazypeople

CoNLL-2012 We will continue our analysis with the task of coreference resolution. Coreference resolution is the task of identifying when different expressions in a text refer to the same entity. The CoNLL-2012 shared task [3] is a benchmark dataset and competition for coreference resolution which provides annotated data where entities are linked across sentences, helping models learn to recognize these relationships. We will fine-tune an mDeBERTa model of type DebertaForTokenClassification, i.e. a model with a token classification head suitable for coreference resolution tasks.

Pre-processing for coreference resolution is quite different from pre-processing in the context of text classification. While text classification assigns a single label to an entire text (e.g. A or B), coreference resolution assigns labels to individual tokens to indicate their membership in coreference chains, which group tokens referring to the same entity. During tokenization, it is crucial to align these labels with the tokenized output. Furthermore, padding ensures consistent length for all token sequences. Tokens without corresponding coreference labels are padded with a special token to maintain alignment and indicate they do not belong to any coreference chain. This method accurately prepares the dataset for training a model to resolve coreferences in text. Compared to Full Fine-Tuning, the PEFT methods dramatically reduce training time and memory usage. Despite their efficiency, PEFT methods achieve comparable, and in some cases superior results. Among the PEFT methods evaluated, Soft Prompts performed the best overall. It achieved the highest accuracy and recall, indicating it can identify correct answers more effectively and consistently. It even outperforms full-fine-tuning. Additionally, Soft Prompts had the second lowest RAM usage and a minimal number of trainable parameters, underscoring its efficiency.

Metric	FFT	LoRA	Soft Prompts	IA3	BitFit
Batch Size	64	128	128	128	128
Training Time [sec]	2129	1803	1881	1799	1788
RAM Usage [MB]	2291	2302	2341	2341	2342
# Parameters [$\times 10^6$]	183.86	0.64	0.05	0.06	0.10
Accuracy	0.76	0.78	0.79	0.78	0.77
Precision	0.64	0.61	0.63	0.61	0.61
F1-Score	0.69	0.68	0.70	0.68	0.68
Recall	0.76	0.78	0.79	0.78	0.77

Table 3. Performance of Full Finetuning vs PEFT methods on CoNLL-2012 for mDeBERTa-base

If we compare the above results for mDeBERTa-base with the below results for mDeBERTa-small, we notice that the training time and RAM usage are significantly smaller for the mDeBERTa-small model. However, despite the smaller model size and reduced resource requirements, the performance in terms of accuracy, precision, recall, and F1-score remains similar between the two models. This suggests that the mDeBERTa-small model offers a more resource-efficient alternative without compromising on performance compared

[3]https://huggingface.co/datasets/conll2012_ontonotesv5

to the larger mDeBERTa-base model.

Metric	FFT	LoRA	Soft Prompts	IA3	BitFit
Batch Size	64	128	128	128	128
Training Time [sec]	1114	971	1060	951	958
RAM Usage [MB]	1926	1928	2024	2100	2094
# Parameters [$\times 10^6$]	141.34	0.34	0.05	0.05	0.05
Accuracy	0.77	0.77	0.78	0.77	0.63
Precision	0.63	0.61	0.62	0.61	0.40
F1-Score	0.69	0.68	0.69	0.68	0.49
Recall	0.77	0.77	0.78	0.77	0.63

Table 4. Performance of Full Finetuning vs PEFT methods on CoNLL-2012 for mDeBERTa-small

Slovene SuperGLUE While most of the research in the field of NLP has been conducted for the English language, we also aim at applying the PEFT methods to multilingual text reasoning. The benchmark used is the Slovene SuperGLUE [13], which is the Slovene version of the English SuperGLUE. The benchmark consists of eight tasks encompassing general NLP tasks. Let us begin with the task BoolQ: a dataset with questions and binary answers [4]. Each instance consists of three components: a question, a passage, and an answer, with the possibility of including the page title as supplementary context. This configuration mirrors the structure of typical natural language inference tasks focused on text pairs.

We employed an mDeBERTa model, see Table 5, as well as a BERTić [15] model, see Table 6, for fine-tuning, which is a transformer language model specifically designed for Bosnian, Croatian, Montenegrin, and Serbian. Note that for BERTić, we had resources on the shared cluster for training only LoRA and FFT.

Metric	FFT	LoRA	Soft Prompts	BitFit
Batch Size	64	128	128	128
Training Time [sec]	5282	4478	4567	4456
RAM Usage [MB]	2337	2338	2338	2337
# Parameters [$\times 10^6$]	278.6	0.62	0.16	0.1
Accuracy	0.7	0.63	0.62	0.65
Precision	0.74	0.78	0.62	0.62
F1-Score	0.77	0.72	0.76	0.76
Recall	0.81	0.77	1.0	1.0

Table 5. Performance of Full Finetuning vs PEFT methods on Slovene SuperGLUE for mDeBERTa-base

For mDeBERTa, the LoRA technique stands out with significantly lower RAM usage and a remarkable reduction in the number of parameters compared to full fine-tuning, while maintaining competitive performance. On the other hand, Soft Prompts and BitFit show promising results in terms of accuracy and precision, although they require slightly more training time and memory resources. For BERTić, the LoRA method also demonstrates a substantial reduction in RAM usage and parameter count, with a notable improvement in precision compared to full fine-tuning. These findings suggest that LoRA could be a viable option for enhancing model efficiency without compromising task performance across different languages and models.

[4]https://huggingface.co/datasets/google/boolq

Metric	FFT	LoRA
Batch Size	64	128
Training Time [sec]	3967	3277
RAM Usage [MB]	761	458
# Parameters [$\times 10^6$]	110.6	1.2
Accuracy	0.74	0.65
Precision	0.78	0.71
F1-Score	0.79	0.73
Recall	0.81	0.75

Table 6. Performance of Full Finetuning vs LoRA on Slovene SuperGLUE for BERTić

SST-5 dataset (also known as Stanford Sentiment Treebank with 5 labels) was created to test model's capacity for sentiment classification. It contains 11855 sentences from movie reviews labeled by human judges. Label of the review can be one of the five classes - negative, somewhat negative, neutral, somewhat positive and positive. Therefore the dataset is sometimes also referred as SST fine-graned.

Before tokenizing sentences for the model we do basic preprocessing which involves removal of special characters, conversion to lower case and removal of stopwords.

For this dataset we employed Deberta model for classification. Results of the finetuning can be seen in Table 7. Lora seems to be performing the best out of all peft methods, regarding accuracy and recall. Other metrics are slightly better or comparable with the other peft methods.

Metric	FFT	LoRA	IA3	BitFit
Batch Size	64	128	128	128
Training Time [sec]	663	542	548	531
RAM Usage [MB]	1979	2105	2105	2105
# Parameters [$\times 10^6$]	184.43	0.61	0.07	0.10
Accuracy	0.52	0.42	0.25	0.25
Precision	0.52	0.22	0.2	0.06
F1-Score	0.51	0.29	0.2	0.06
Recall	0.52	0.42	0.25	0.25

Table 7. Performance of Full Finetuning vs PEFT methods on SST5 for mDEBERTA-base

Discussion

Discussion and issues In our analysis, we successfully implemented four different PEFT methods and applied them to four different datasets. Unexpectedly, the implementation of the PEFT methods were rather straightforward while the import as well as pre-processing of the datasets posed the real difficulty. All but the BitFit method are implemented through the library `peft`. For BitFit, we had to manually specify, that only the bias parameters of the model are to be trained.

One issue with implementing the pre-processing of the CoNLL-2012 dataset was to handle the special token representing an "empty" label for a token. In the beginning we labelled an empty class with the value of -100 for the delimiters, padding and tokens without corresponding coreference, this caused an assertion failure in the CUDA code which proved to be very difficult to debug. In the end we replaced -100 by introducing an additional class in the classifier and assigning this class to all the special tokens. We also had problems with soft prompt trainer with CommonsenseQA and SST5 datasets, here the issue was kernel length mismatch we did not manage to fix this issue so we did not evaluate soft prompt trainer on this two datasets. The last issue that we faced was the resource availability on the cluster, as we do not own any GPUs we had to test everything on the cluster and that made debugging much harder since we had to first wait to acquire resources and only then we could see the results, which made our development cycle quite slow.

Future directions As an improvement for the future we would do more thorough hyperparameter search for even more robust results. Another thing could be to resolve issues with soft prompt finetuning for some of the datasets/models.

Conclusion

In conclusion, our investigation into PEFT methods across various NLP tasks reveals significant advantages in terms of resource efficiency without compromising task performance. Specifically, the LoRA method consistently demonstrates remarkable reductions in RAM usage and parameter count compared to full fine-tuning, while maintaining competitive or even superior performance across different models and languages. Soft Prompts also show promise, particularly in achieving high accuracy and recall, indicating effective identification of correct answers with minimal resource requirements. These findings underscore the potential of PEFT methods in enhancing model efficiency and scalability, paving the way for more sustainable and accessible NLP solutions.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[3] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training, 2018.

[4] Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment, 2023.

[5] OpenAI. Gpt-4 technical report, 2024.

[6] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[7] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021.

[8] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning, 2022.

[9] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models, 2022.

[10] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge, 2019.

[11] Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *Joint conference on EMNLP and CoNLL-shared task*, pages 1–40, 2012.

[12] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[13] Aleš Žagar and Marko Robnik-Šikonja. Slovene SuperGLUE benchmark: Translation and evaluation. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 2058–2065, Marseille, France, June 2022. European Language Resources Association.

[14] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! Topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018.

[15] Nikola Ljubešić and Davor Lauc. BERTić - the transformer language model for Bosnian, Croatian, Montenegrin and Serbian. In *Proceedings of the 8th Workshop on Balto-Slavic Natural Language Processing*, pages 37–42, Kiyv, Ukraine, April 2021. Association for Computational Linguistics.