Continuous Prompt Tuning for Russian: How to Learn Prompts Efficiently with RuGPT3?

Nikita Konodyuk[1,2] and Maria Tikhonova[1,2](✉)

[1] SberDevices, Sberbank, Moscow, Russia
tikhonova.m.iva@sberbank.ru
[2] National Research University Higher School of Economics, Moscow, Russia
nekonodyuk@edu.hse.ru

Abstract. Adaptation to downstream tasks is a crucial part of the pre-trained language model (PLM) life cycle. Fine-tuning, traditionally used for this purpose, is an expensive procedure in terms of computation and memory. Dramatic growth of PLM capacities has led to the emergence of zero- and few-shot methods, which use natural language to describe tasks. Although these methods do not modify the parameters of the model, they rely on manual prompt design, which may be suboptimal. To address this issue, a range of techniques for automatic prompt search have been proposed recently.

In this paper, we present a framework for continuous prompt tuning (CPT) in Russian. We evaluated our framework by adapting RuGPT3 to tasks in the Russian benchmark SuperGLUE. We obtained metrics better or comparable to fine-tuning, while training only an auxiliary model that provides prompt embeddings, so the total number of trained parameters accounts for less than 0.4% of that of RuGPT3. In addition, we conducted experiments comparing different configurations of the framework and explored the lower bound to which we can reduce the number of parameters. Our source code is publicly available at https://github.com/sberbank-ai/ru-prompts.

Keywords: Natural language processing · Language models · Model training · Transformer models · Language model adaptation

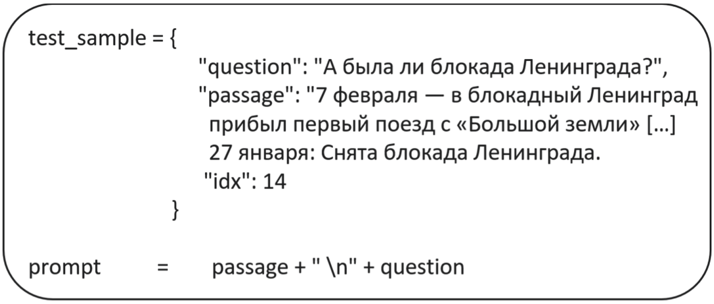
1 Introduction

Language models, in particular, Generative Pre-trained Transformers, have shown prominent abilities for many Natural Language Processing (NLP) tasks. The pre-training fine-tuning paradigm for solving downstream tasks [1], which has been the dominant approach, especially for transformer models, will be limited as long as it requires large labelled training corpora. Moreover, fine-tuning large language models can be computationally expensive and time-consuming.

In [2] the authors present GPT3, an autoregressive language model, which can be applied without any gradient updates or fine-tuning, with tasks and

E. Burnaev et al. (Eds.): AIST 2021, CCIS 1573, pp. 30–40, 2022.
https://doi.org/10.1007/978-3-031-15168-2_3

few-shot demonstrations specified purely via text interaction with the model. The methodology in [3] introduces the concept of **few-shot**: the model receives several training examples and a test prompt in a text format as an input and makes predictions based on them. The setting when the model receives no training examples and makes a prediction based only on the text prompt constructed from a test sample is called zero-shot. To illustrate the idea of a text prompt we present an example for the DaNetQA task[1] (a question answering task for questions with binary answers *yes* or *no* from the Russian benchmark SuperGLUE) in Fig. 1. After the format of the text prompt is defined, we sequentially unite it with each of the answer options (that is, each of the possible labels) and measure the perplexity of the resulting text fragment. The answer options are then sorted by the perplexity scores, and the one with the lowest score is considered to be a prediction. Thus, such an approach does not require any additional training and the answer can be obtained with the use of the original pre-trained model.



```
test_sample = {
                    "question": "А была ли блокада Ленинграда?",
                    "passage": "7 февраля — в блокадный Ленинград
                     прибыл первый поезд с «Большой земли» [...]
                     27 января: Снята блокада Ленинграда.
                    "idx": 14
                }

prompt      =      passage + " \n" + question
```

Fig. 1. A text prompt constructed for a DaNetQA test sample.

These few-shot and zero-shot methods have shown promising results on a wide range on Natural Language Processing (NLP) tasks, especially with large generative models. However, such an approach obviously has disadvantages, one of the most important of which is that even a slight change in the prompt format may significantly influence the result. Since manual selection of prompt templates is not optimal, the idea naturally arises: to automatically search prompts. Since recently methods which tune prompt-embedding in discrete and continuous spaces have been actively developing.

Several attempts made in this direction focused on discrete prompt search [4–6] and demonstrated the effectiveness of an automated approach. However, as long as neural language models are inherently continuous, discrete search for the prompts is likely to be sub-optimal. Thus, the next step is to search pseudo-prompts in continuous embedding space. This idea has been explored in several recent works [7–10] and has proved to be quite fruitful.

[1] https://russiansuperglue.com/tasks/task_info/DaNetQA.

In this work we follow the idea of the P-tuning method introduced in [11], where the authors use a bidirectional LSTM to learn continuous prompt embeddings. Namely, we present a framework for ***C***_ontinuous_ ***P***_rompt_ ***T***_uning_ (or simply CPT) for the Russian language. In addition, we carry out a series of experiments on the Russian SuperGLUE benchmark [12][2] comparing CPT with zero-shot and standard fine-tuning. We explore the influence of the number of trainable parameters on the result. The code is publicly available in our GitHub repository.[3]

Thus, the contribution of this work is three-fold: (i) we release the framework for CPT for Russian, which can be easily adapted to various models; (ii) we evaluate CPT for the RuGPT3 model on Russian SuperGLUE and show that it can be regarded as a strong competitor to fine-tuning and zero-shot; (iii) we show that the number of trainable LSTM parameters can be reduced without significant losses in total quality.

This paper is structured as follows: Sect. 2 describes the method implemented in the CPT framework; Sect. 3 presents the evaluation setup and the analysis of the conducted experiments; Sect. 4 is devoted to the analysis of the model behavior and discusses the results; and, finally, Sect. 5 concludes the paper.

2 Method

In this section we introduce CPT, deriving it from few-shot and zero-shot settings as they are introduced in [2]. We consider a classification task and follow the *generative classification* paradigm, where prediction for a prompt is inferred from the first token generated by a model after processing the prompt.

A natural language prompt is the core element of all prompt-based methods. It combines a description of a downstream task with optional examples that should also be given in the format, which should be clearly understood by the model. Manual prompt search is always a matter of trial and error and therefore is substantially hard to formalize. Nevertheless, in the vast majority of cases, the prompt consists of the same set of semantic blocks.

To illustrate this idea, let us consider the following example of a few-shot prompt for a machine translation task:

Translate English to French
sea otter => loutre de mer
plush giraffe => girafe peluche
cheese => ⟨MASK⟩

In this case the prompt takes the following formal format:

[2] https://russiansuperglue.com/.
[3] https://github.com/sberbank-ai/ru-prompts.

Translate English to French
{word_in_english} => {word_in_french}
{word_in_english} => {word_in_french}
{word_in_english} => ⟨MASK⟩

In the example above we have pairs of instance queries (or objects) and targets, where the answer for the last instance is masked, as well as special service elements which we further refer to as task instructions (or simply TI), which help the model understand what is required in the task. In fact, in other applications we also encounter such elements as task context and instance context. They are necessary in such tasks as summarization, i.e. in those where the query depends on additional context.

Thus, the generic few-shot prompt format can be formalized as follows:

⟨TI⟩ ⟨task context⟩ ⟨TI⟩
⟨TI⟩ ⟨instance context⟩ ⟨TI⟩ ⟨instance query⟩ ⟨TI⟩ ⟨instance target⟩
⟨TI⟩ ⟨instance context⟩ ⟨TI⟩ ⟨instance query⟩ ⟨TI⟩ ⟨instance target⟩
⟨TI⟩ ⟨instance context⟩ ⟨TI⟩ ⟨instance query⟩ ⟨TI⟩ ⟨MASK⟩

Since the zero-shot approach differs from the few-shot only in terms of the number of provided examples, the zero-shot prompt format in essence is just a truncation of the few-shot prompt format:

⟨TI⟩ ⟨instance context⟩ ⟨TI⟩ ⟨instance query⟩ ⟨TI⟩ ⟨MASK⟩

For example:

Известно, что Москва была основана в 1147 году на Москве-реке.
Вопрос: Была ли Москва основана в 12 веке? Ответ: ⟨MASK⟩

Note that everything except the task instruction is usually derived from the dataset fields and thus is not subject to change. Task instructions, on the contrary, are defined by the task itself and even minor changes lead to significant performance deviations. Moreover, for different models different TI may be optimal. Although hard to discover manually, they can be trained by gradient descent, as proposed in [8,9,11] in different variations.

We train task instructions by gradient descent, so the prompt takes the following form:

⟨learned instructions⟩ Москва была основана в 1147 году на Москве-реке.
⟨learned instructions⟩ Была ли Москва основана в 12 веке?
⟨learned instructions⟩ ⟨MASK⟩

We follow the original methodology proposed in [11] and produce trainable embeddings with an auxiliary BiLSTM-based model, which we also refer to as *prompt provider*. Its architecture is as follows: a sequence of trainable vectors is passed through BiLSTM and then through two-layer MLP with ReLU activation. The dimension of the output sequence of vectors is equal to the embedding dimension of backbone model, and their number is equal to the total number of ⟨SP⟩ tokens in *prompt format*. These embeddings are inserted to the corresponding positions in the input of backbone and trained via backpropagation.

3 Experiments

In this section, we describe experiments conducted on the Russian SuperGLUE benchmark. First, we give a brief description of the tasks on which we evaluated CPT, then we describe the model used in the experiments, after that we specify details about the baseline methods and, finally, present the results and their analysis. We conclude the section with an additional series of experiments with a different number of trainable parameters.

3.1 Data

All the experiments were conducted using the Russian general language understanding evaluation benchmark – RussianGLUE. It was collected and organized analogically to the SuperGLUE methodology [13]. Russian SuperGLUE comprises 9 tasks divided into 5 groups:

- **Textual Entailment & Natural Language Inference (NLI)**: TERRa, RCB, LiDiRus;
- **Common Sense**: RUSSE, PARus;
- **World Knowledge**: DaNetQA;
- **Machine Reading**: MuSeRC, RuCoS;
- **Reasoning**: RWSD.

Below a brief description of each task is given, and aggregated information is presented in Table 1.

TERRA Textual Entailment Recognition for Russian is aimed at capturing textual entailment in a binary classification form. Given two text fragments (premise and hypothesis), the task is to determine whether the meaning of the hypothesis is entailed from the premise. The dataset was sampled from the Taiga corpus [14].

RCB The Russian Commitment Bank is a 3-way classification task aimed at recognizing textual entailment (NLI). Analogically to TERRA, each example in RCB consists of premise and hypothesis. However, in this task a premise can be a short paragraph, not necessarily one phrase.

LiDiRus (also referred to as a diagnostic dataset) is an expert-constructed evaluation dataset for recognizing textual entailment tasks on paired sentences. It is a direct translation from the English SuperGLUE diagnostic dataset, originally introduced in [15]. It consists of 1104 sentence pairs which are used as a test set for testing models' capacity to solve NLI task. In addition, the diagnostic dataset has a rich annotation of various linguistic phenomena, partly inserted artificially to explore the possible biases and errors on a task that can be considered truly universal for all languages. The annotation includes 33 features which can be devided into 4 categories: *Predicate-Argument Structure, Logic, Lexical-Semantics, and Knowledge*. Such annotation makes it possible to analyze the model behaviour with respect to linguistic features.

RUSSE is a binary classification task that involves word sense disambiguation. Given a pair of sentences containing the same ambiguous word, the goal of the model is to recognize if the word is used in the same meaning. The dataset was constructed from RUSSE[4].

PARus is a binary classification task aimed at identifying the most plausible alternative out of two for a given premise. The correct alternatives is the dataset are randomized so that the expected performance of random guessing yields 50% accuracy score.

DaNetQA is a Russian question-answering dataset for questions with binary answers (*yes* or *no*) which follows the BoolQ design. Each example consists of a triplet of question, passage, and answer.

MuSeRC is a machine reading comprehension (MRC) task. Each sample consists of a text paragraph, multi-hop questions based on the paragraph, and possible answers for each question. The goal of the task is to choose all correct answers for each question.

Table 1. Russian SuperGLUE task description. Train/Val/Test stand for example amount (sentence pairs or texts); MCC stands for Matthews Correlation Coefficient; EM - Exact Match.

Task	Task type	Task metric	Train	Val	Test
TERRa	NLI	Accuracy	2616	307	3198
RCB	NLI	Avg. F1/Accuracy	438	220	438
LiDiRus	NLI & Diagnostics	MCC	0	0	1104
RUSSE	Common sense	Accuracy	19845	8508	18892
PARus	Common sense	Accuracy	400	100	500
DaNetQA	World knowledge	Accuracy	1749	821	805
MuSeRC	Machine reading	F1/EM	500	100	322
RuCoS	Machine reading	F1/EM	72193	7577	7257
RWSD	Reasoning	Accuracy	606	204	154

RuCoS is an MRC task that involves commonsense reasoning and world knowledge. The dataset is a counterpart of ReCoRD[5] for English.

RWSD The Russian Winograd Schema task is devoted to coreference resolution in a binary classification form. The corpus was created as a manually validated translation of the Winograd Schema Challenge[6].

[4] https://russe.nlpub.org/downloads/.

[5] https://sheng-z.github.io/ReCoRD-explorer/.

[6] https://cs.nyu.edu/faculty/davise/papers/WinogradSchemas/WS.html.

3.2 Model

We investigate the effectiveness of CPT and conduct all the experiments using a RuGPT3[7] model, a Russian adaptation of the autoregressive language model GPT3 [2], which the authors claimed as having strong in-context learning abilities and which has shown impressive results in zero- and few-shot settings in many NLP tasks. Namely, we run the experiments on *RuGPT3-Large* which is publicly available in the Hugging Face Python library[8].

For a prompt provider we used an LSTM with a hidden dimension of 256 and input dimension of 16. Thus, given that the number of parameters of RuGPT3-Large equals 760M and the number of trainable parameters of the prompt provider was 3M, the fraction of total trainable parameters accounts only for 0.4% of model size. Additionally, in Sect. 3.5 we experiment with different numbers of parameters in the prompt provider.

We cast each of the tasks described in Sect. 3.1 to a binary or ternary classification problem. Training CPT on a task involves formatting its samples in line with the corresponding prompt format from Table 2, where ⟨target⟩ takes the values of "да" and "нет" (which are also called *label verbalizers*), and additionally "возможно" in the case of ternary classification. The symbol ⟨SP⟩ represents a trainable token (soft prompt). Following the original methodology from [11] we used sequences of 3 soft prompt tokens. However, it should be noted that the number of soft prompt tokens can be varied and the optimal choice of the number of ⟨SP⟩ in each sequence for each task is another area for the research. The embeddings of trainable tokens are provided by the prompt provider. We train the prompt provider to output such embeddings, which maximize the probability of the right label verbalizer for each sample. As a criterion we use cross entropy among the logits corresponding to the label verbalizers.

Table 2. Prompt formats used for training on Russian SuperGLUE tasks. The symbol ⟨SP⟩ represents a trainable token (soft prompt).

Task	**Prompt Format**
DaNetQA	⟨SP⟩⟨SP⟩⟨SP⟩ ⟨passage⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨question⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨target⟩
TERRa	⟨SP⟩⟨SP⟩⟨SP⟩ ⟨premise⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨hypothesis⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨target⟩
LiDiRus	⟨SP⟩⟨SP⟩⟨SP⟩ ⟨sentence1⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨sentence2⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨target⟩
MuSeRC	⟨SP⟩⟨SP⟩⟨SP⟩ ⟨paragraph⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨question⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨answer⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨target⟩
PARus	⟨SP⟩⟨SP⟩⟨SP⟩ ⟨premise⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨choice2⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨choice1⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨target⟩
RCB	⟨SP⟩⟨SP⟩⟨SP⟩ ⟨premise⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨hypothesis⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨target⟩
RUSSE	⟨SP⟩⟨SP⟩⟨SP⟩ ⟨word⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨sentence1⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨sentence2⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨target⟩
RWSD	⟨SP⟩⟨SP⟩⟨SP⟩ ⟨text⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨span1_text⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨span2_text⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨target⟩
RuCoS	⟨SP⟩⟨SP⟩⟨SP⟩ ⟨passage⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨statement⟩ ⟨SP⟩⟨SP⟩⟨SP⟩ ⟨target⟩

[7] https://github.com/sberbank-ai/ru-gpts.

[8] https://huggingface.co/sberbank-ai/rugpt3large_based_on_gpt2.

3.3 Baselines

In order to evaluate the proposed framework we conducted a series of experiments with our RuGPT3-Large model, comparing CPT with standard fine-tuning and zero-shot approaches.

We fine-tuned RuGPT3-Large for every task using *jiant-russian* (version 2.0) library[9] (a library released by the creators of the benchmark, which is aimed at fine-tuning various models on Russian SuperGLUE) with standard parameter configuration.

For the zero-shot method we used its modification based on the model perplexity. Namely, for each test sample we calculate the perplexity of the corresponding prompts united with one of the possible targets using formula 1. We then choose the best target, as the one with the lowest perplexity score.

$$PPL(t) = \exp\left(-\frac{1}{|t|}\sum_{i=0}^{|t|}\log_{p_\theta}(x_i|x_{<i})\right) \tag{1}$$

where t is an input text (in our case, a text prompt concatenated with one of possible targets), $|t|$ is the length of the text in tokens, and $log_{p_\theta}(x_i|x_{<i})$ is the log-likelihood of the i-th token in t conditioned on the preceding ones.

3.4 Results

The results of the experiments are presented in Table 3. It gives an exact representation of CPT performance compared with zero-shot and fine-tuning approaches. It can be seen that CPT outperforms each of zero-shot and fine-tuning on most of the tasks. Namely, it outperforms zero-shot on LiDiRus, MuSeRC, TERRa, RUSSE, RWSD, and DaNetQA; and it shows better results than fine-tuning on MuSeRC, TERRa, RWSD, DaNetQA and RuCoS. Thus, CPT allows to achieve reasonable model performance without either human assistance in prompt search or computational resources sufficient for fine-tuning.

The poor performance of CPT on RCB can be explained by the small size of the training corpora (only 438 training samples), which is insufficient for learning good prompts. As for RuCoS, the most plausible explanation is that the chosen generative approach is not optimal for such a complicated type of task. In the future we plan to use CPT with contrastive classification for this task (see Sect. 4 for a more in-depth description of the approach), which will hopefully yield better results.

3.5 Experiments with Different Number of Trainable Parameters

In addition, we explored how the number of trainable parameters in an LSTM influence the model quality. Our goal was to minimize the number of trainable

[9] https://github.com/RussianNLP/RussianSuperGLUE/tree/master/jiant-russian-v2.

Table 3. Results of RuGPT3-Large evaluation on Russian SuperGLUE in different settings. We score the tasks in line with the metrics specified in Table 1. The scores for all tasks are then averaged to get the total score. For the tasks with multiple metrics, the metrics are averaged.

Approach	Total score	LiDiRus	RCB	PARus	MuSeRC	TERRa	RUSSE	RWSD	DaNetQA	RuCoS
Zero-shot	51.4	12.8	30.4/42.2	63.0	72.7/52.2	52.5	57.1	62.3	57.0	64.0/63.5
Fine-tuning	50.5	23.1	41.7/48.4	58.4	72.9/33.3	65.4	64.7	63.6	60.4	21.0/20.2
CPT	48.2	14.0	17.6/35.8	47.2	74.2/38.3	67.9	62.8	66.9	60.7	32.0/31.4

parameters and, therefore, to optimize and speed up CPT training. For this purpose we conducted a series of experiments on 4 Russian SuperGLUE tasks (DaNetQA, PARus, RCB, and TERRa). These tasks were chosen as long as they are considered to be most popular among all the benchmark tasks and due to the size of their training corpora.

In the experiments we trained CPT clones, each with a different dimension of LSTM hidden states varying from 1 to 1536. Results are presented in Table 4 and a general picture is given in Fig. 2. It can be seen that the number of the hidden dimensions and, therefore, the number of the trainable parameters can be significantly reduced without noticeable decrease in total quality. Moreover, an excessive number of trainable parameters may seemingly lead to overfitting and a non-optimal score. For instance, while for TERRa and DaNetQA the accuracy keeps increasing, on PARus it reaches the optimal value on `hidden_dim=16` and then decreases. We connect this behaviour with the number of training samples, which is significantly greater for DaNetQA and TERRa, than for PARus and RCB.

If we calculate the average of the scores for the 4 tasks considered, we see that the result for `hidden_dim=64` is only 7,5% worse than for the maximal size of `hidden_dim=1536` while it requires 368 times fewer trainable parameters. Thus, it can be concluded that the number of trainable parameters can be significantly reduced without much loss in quality.

Table 4. CPT results with different LSTM hidden dimensions. "Hidden dim" stands for the number of LSTM hidden dimensions and "Params" for the number of trainable parameters. The average score is calculated as the mean score of 4 tasks. For RCB the two metrics are first averaged.

`hidden_dim`	Params	DaNetQA	PARus	RCB	TERRa	Average
1	3.2K	50.3	48.8	17.6/35.6	50.3	44.0
4	8.5K	53.4	47.0	20.7/32.9	50.3	44.4
16	37.5K	52.0	53.0	17.6/35.8	53.8	46.4
64	274K	54.7	51.2	17.5/35.6	54.8	46.8
256	3.2M	56.9	47.4	17.6/35.8	53.2	46.1
1024	45.7M	58.0	48.4	17.6/35.8	60.4	48.4
1536	101M	60.7	47.2	17.6/35.8	67.9	50.6

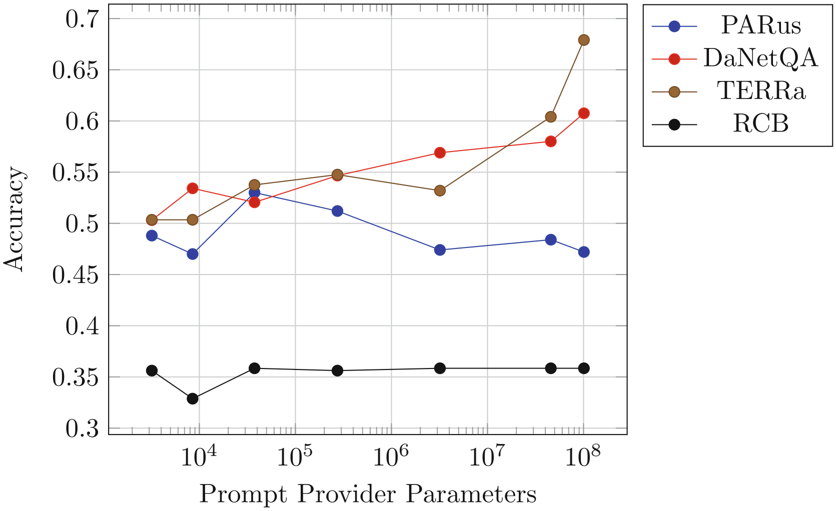


Fig. 2. Accuracy score for CPT with different LSTM hidden dimensions.

4 Discussion

Despite the fact that the overall RuGPT3 performance with CPT is comparable with fine-tuning and zero-shot approaches, it shows quite a poor score on several tasks (namely, RCB, PARus, and RuCoS). This may be explained in several ways. For example, we suppose that the low score on RCB and PARus can be accounted for by the small size of the training corpora (438 and 400 training samples respectively). Such a modest dataset size is probably not enough for learning good continuous prompts.

As for RuCoS, its low score can be explained by the fact that a generative approach is not optimal for such a complicated task. Thus, in order to overcome this limitation we are planning to use CPT for contrastive classification. Compared with generative classification, this approach will utilize the relationships of multiple versions of each text, for example multiple answers or multiple prompts, thus being able to handle multiple-choice tasks in a more natural manner. Another use case of contrastive classification will probably be multiclass classification where it is hard to choose suitable label verbalizers.

Another area for future research could become imposing additional restrictions on prompt provider to increase interpretability of the output prompt. Although they are practically efficient, they currently remain non-interpretable.

5 Conclusion

In this paper we propose a framework for continuous prompt tuning for the Russian language. We use an RuGPT3 model and evaluate it on the Russian

SuperGLUE benchmark. In the experiments our method shows results competitive with zero-shot and fine-tuning and even outperforms them on most of the tasks. In addition, we explore the influence of the number of trainable LSTM parameters and find out that it can be significantly reduced without any losses in quality.

In the future we are planning to implement a contrastive classification approach for CPT and apply the framework to other models, such as the RuT5 and RuBERT models.

Acknowledgements. We would like to thank Sarah Caitlin Bennett for her help with editing the paper and advice on the text structure.

References

1. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: Bert: pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
2. Brown, T.B., et al.: Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020)
3. Radford, A., Jeffrey, W., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners. OpenAI Blog **1**(8), 9 (2019)
4. Shin, T., Razeghi, Y., Logan IV, R.L., Wallace, E., Singh, S.: Autoprompt: eliciting knowledge from language models with automatically generated prompts. arXiv preprint arXiv:2010.15980 (2020)
5. Reynolds, L., McDonell, K.: Prompt programming for large language models: beyond the few-shot paradigm. In: Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems, pp. 1–7 (2021)
6. Gao, T., Fisch, A., Chen, D.: Making pre-trained language models better few-shot learners. arXiv preprint arXiv:2012.15723 (2020)
7. Li, X.L., Liang, P.: Prefix-tuning: optimizing continuous prompts for generation. arXiv preprint arXiv:2101.00190 (2021)
8. Lester, B., Al-Rfou, R., Constant, N.: The power of scale for parameter-efficient prompt tuning. arXiv preprint arXiv:2104.08691 (2021)
9. Hambardzumyan, K., Khachatrian, H., May, J.: Warp: word-level adversarial reprogramming. arXiv preprint arXiv:2101.00121 (2021)
10. Liu, X., Ji, K., Fu, Y., Du, Z., Yang, Z., Tang, J.: P-tuning v2: prompt tuning can be comparable to fine-tuning universally across scales and tasks. arXiv preprint arXiv:2110.07602 (2021)
11. Liu, X., et al.: GPT understands, too. arXiv preprint arXiv:2103.10385 (2021)
12. Shavrina, T., et al.: Russiansuperglue: a Russian language understanding evaluation benchmark. arXiv preprint arXiv:2010.15925 (2020)
13. Wang, A., et al.: Superglue: a stickier benchmark for general-purpose language understanding systems. arXiv preprint arXiv:1905.00537 (2019)
14. Shavrina, T., Shapovalova, O.: To the methodology of corpus construction for machine learning: "taiga" syntax tree corpus and parser. In: Proceedings of "CORPORA-2017" International Conference, pp. 78–84 (2017)
15. Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., Bowman, S.R.: Glue: a multitask benchmark and analysis platform for natural language understanding. arXiv preprint arXiv:1804.07461 (2018)