Fine-Tuning Text Pre-Trained Transformers to In-Context Learn Simple Function Classes

Deep Learning Final Project Report, Fall 2023

Team

- William Zhang, EECS Graduate, chengyu_zhang@berkeley.edu
- Yuanbo Chen, EECS Graduate, yuanbo_chen@berkeley.edu
- Eric Tai, EECS Graduate, erictai@berkeley.edu
- Michelle Wang, EECS Undergraduate, michellew@berkeley.edu

Abstract

Our project is inspired by the study titled "What Can Transformers Learn In-Context? A Case Study of Simple Function Classes." We find the practical value limited due to its training approach, which involves starting from scratch. Instead, our focus is on exploring the in-context learning capability of a text pre-trained language model, specifically GPT-2, in the context of a linear regression task. Given the substantial domain shift, we propose two inference approaches. The first adheres to the methodology in the referenced paper, utilizing vectorized xy-pairs as inputs. The second transforms vectorized xy-pairs into a single line of text. Our experimentation reveals that the second approach fails, while the first approach successfully learns the linear regression task. This outcome underscores the presence of valuable knowledge in text pre-trained weights that can be leveraged. We thus further experiment with various fine-tuning techniques, including full fine-tune, soft prompting, and low rank adaptation (LoRA). Through a comprehensive comparison and analysis of results concerning computation and in-context learning errors, we demonstrate that LoRA is most effective in transferring text pre-trained knowledge to in-context linear regression learning with minimal additional computational cost.

1 Introduction

We base our project on *What Can Transformers Learn In-Context? A Case Study of Simple Function Classes* [1], a study that trains standard Transformers (GPT-2 [2]) from scratch to perform in-context learning of simple function classes. While the paper provides a compelling empirical demonstration that transformers can learn algorithms from in-context examples, we deem that there are several shortcomings in its practical value. Our project thus aims to address some of these limitations.

Firstly, the paper trains the model from scratch, *i.e.*, there are no pre-trained weights loaded as the initial state, and the whole model is trained and updated through back-propagation in a meta-learning fashion on task-specific numerical & vectorial data of simple function classes. Such an approach establishes an upper bound for the in-context learning ability/performance of the Transformer model in simple function classes. However, this upper bound might not be a good indicator or easily achievable, as in-context learning happens only at inference time and language models like GPT-2 are typically pre-trained on a variety of text-based inputs instead of simple function classes in vector form. Moreover, in reality, larger models such as GPT-3 [3], GPT-4 [4] are more commonly used, so training the whole model will be costly and impractical. Therefore, we examine the in-context learning performance of the model (GPT-2) directly from text pre-trained weights.

While directly running in-context learning on numerical vectors at inference time using text pre-trained weights is expected to have low performance, as there is a large domain shift, we explore two options to leverage the text pre-trained knowledge: 1) Convert the vectorial input data into text form; 2) Fine-tuning. The first option addresses the traditional in-context learning setting, where large language models are prompted with text described task to generate text form output. Through which we wish to get an intuitive and practical interpretation of Transformer's in-context learning ability. For the second fine-tuning option, we further fine-tune the text pre-trained GPT-2 using techniques such as Soft Prompting [5] and Low-Rank Adaptation (LoRA) [6]. Essentially, we seek to find a middle ground between the pure inference in-context learning and training-from-scratch, with the goal of using limited resources to close the gap between the upper bound (provided by training-from-scratch in the paper) and our fine-tuning approach.

We mainly investigate the in-context learning ability of GPT-2 in linear regression task. We compare the results on the four approaches: only pre-train (vector input), only pre-train (converted text input), fine-tune (vector input), and train-from-scratch (vector input), and show that indeed:

- Text pre-trained GPT-2 can perform in a numerical task, with inference using vector input slightly better than 3-Nearest Neighbors;
- Text-formed input fails to output numerical values, despite some prompt engineering;
- Full fine-tune reaches comparable result as train-from-scratch, but with much less computation overhead and more robustness;
- Fine-tuning with Soft Prompting is less effective in both computation and accuracy than LoRA, with the latter closing in on the performance of full fine-tune or train-from-scratch.

Our code is available at `https://github.com/MstXy/in-context-learning`, with all instructions for setup, training and evaluation in the `README.md` file under the repository.

2 Related Works

2.1 In-Context Learning

In-context learning is a specific method of prompt engineering where supporting examples of a novel task of interest are provided to the language model as part of the input prompt. The model can then generate corresponding answers, without any further fine-tuning. With the publication of GPT-3 [3], Transformers have been shown to have powerful in-context learning ability, exceeding the then SOTA performance in many few-shot tasks. [1] poses in-context learning as a well-defined problem of learning function classes at inference time, and empirically investigates Transformer's performance in in-context learning simple function classes through a meta-learning training schema. Yet, the paper trains directly on simple function classes, essentially disregards the "language model" component of in-context learning. [7] uses a text pre-trained GPT-3 instead, and demonstrates the on-the-fly learning ability of GPT-3 in novel and even "unnatural" text tasks. Nevertheless, the blog engage with mostly textual tasks. We thus draw inspirations from it, and uses a text pre-trained GPT-2 on predicting simple function classes, to examine Transformers in-context learning ability on an even more "unnatural" task domain.

2.2 Parameter-Efficient Fine-Tuning

On encounter with a novel task, it is common practice to fine-tune the language model on data of that novel task. A full fine-tune approach typically results in huge performance gains, but as models get larger, it becomes infeasible to train and store the whole parameters of a language model. Parameter-efficient Fine-tuning (PEFT) thus aims to address such issues by only fine-tune a small number of (extra) model parameters. Soft Prompting [5] simplifies [8], instead of including additional trainable context to all the Transformer layers, it only adds to the input prompt. However, optimization on this method requires hyperparameter search and is hard to directly optimize. LoRA [6] thus proposes to use a trainable low rank decomposition to represent the update on pre-trained model weights, freezing pre-trained weights and inserting knowledge rather than context into model layers. Our project aims to compare the different PEFT techniques.

3 Method

3.1 Problem Setup

As an extension research from [1], we follow [1]'s problem setup. Specifically, for each training input prompt, a function f^l is sampled randomly from a certain function class $\mathcal{F}$. Then, $n+1$ x_i's are sampled from a distribution, and we construct the corresponding y_i's with $y_i = f^l(x_i)$. In our experiment, we have $x_i \sim N(0, \mathbf{I}_d)$. In the linear regression task we are addressing in this project, $f^l(x_i) = w^\top x_i$, with $w \sim N(0, \mathbf{I}_d)$.

With the (x_i, y_i) pairs constructed, we form the training input prompt as $P^l = (x_1^l, y_1^l, \ldots, x_n^l, y_n^l, x_{query}^l)$, with $P^l \in \mathbb{R}^{d\times(2n+1)}$. Our Transformer model M then predicts y_{query}^l given all the supporting (x_i^l, y_i^l) pairs, and tries to minimize the expected loss over all the k training input prompt:

$$\min \mathbb{E}_P \left[\frac{1}{k} \sum_{l=1}^{k} \ell \left(y_{query}^l, f^l(x_{query}^l) \right) \right]. \tag{1}$$

In our linear regression task, $\ell(\cdot, \cdot)$ in (1) is the squared error.

Note that for different training input prompt, the function f^l is different. This task is thus a *meta-learning* task. The Transformer model M is *learning to learn* the underlying function f^l, given the supporting (x_i^l, y_i^l) pairs in the prompt. Thus, the task and prompt formulation in test time is exactly the same as the ones in training time.

Notably, the input prompts given to the Transformer model are in vector form, with dimension $d = 10$ set in our experiment. The model thus gets rid of the tokenization process, and replaces the embedding layer default to Transformer model with a linear layer $W_{in} : \mathbb{R}^d \to \mathbb{R}^{embed}$. The output head is also replaced by a linear layer $W_{out} : \mathbb{R}^{embed} \to \mathbb{R}$, mapping the last hidden state of the Transformer backbone to a scalar.

3.2 Train from Scratch

The paper [1] trains the Transformer model from scratch with vast training input prompts generated in the aforementioned manner, which we deem as impractical and non-intuitive. But we start our experiment replicating the results in the paper. The training process is simple: the input layer, Transformer backbone, and the output layer is trained all together. We utilize a similar curriculum training procedure in [1]. Because it trains from scratch to convergence solely on generated input prompts, such an approach provides an upper bound of Transformer's in-context learning ability.

3.3 Text Pre-train Only

Having attained the upper bound, we seek to investigate: what is the common case performance of text pre-trained Transformer's performance in in-context learning? We formulate two approaches, one stick to the vector input in [1], the other gets creative and converts generated numerical vectors to text, aiming to alleviate the problem of domain shift from text generation to mathematical tasks (*i.e.*, linear regression).

For the first approach, we load the pre-trained weights directly from HuggingFace with the following one-liner:

```
from transformers import GPT2Model
GPT2Model.from_pretrained("gpt2")
```

Note that here GPT2Model is the Transformer backbone, with no input nor output layers. Because our input is in numerical and vectorial form, and output is in scalar form, we can not load the text pre-trained weights for the input embedding layer and output head. To achieve inference on text pre-trained model, we freeze the Transformer backbone, and train to fit our randomly initialized input and output linear layers W_{in} & W_{out}, which could be trained very quickly.

For the second approach, we first convert the sampled prompt $P^l = (x_1^l, y_1^l, \ldots, x_n^l, y_n^l, x_{query}^l)$ to a single string s^l in the following way:

$$s^l = \text{“}\mathbf{x:}\ x_{1,(1)}^l \ldots x_{1,(d)}^l\ \mathbf{y:}\ y_1^l\mathbf{,} \ldots\mathbf{,}\ \mathbf{x:}\ x_{n,(1)}^l \ldots x_{n,(d)}^l\ \mathbf{y:}\ y_n^l\mathbf{,}\ \mathbf{x:}\ x_{query,(1)}^l \ldots x_{query,(d)}^l\ \mathbf{y:}\ \text{”} \tag{2}$$

All the $x^l_{i,(j)}$,$y^l_{i,(j)}$'s are scalar entries of the corresponding vectors rounded to 2 decimal places. The goal is to let the model fill in the value y_{query} for the last $\mathbf{y}$. We pass the batched strings through a pre-trained tokenizer and then a pre-trained GPT-2 model with language model head. Lastly, the first of the autoregressive outputs is decoded by the tokenizer, taken as y_{query}.

3.4 Fine-tuning

Admittedly, there is still a large domain shift from the pre-trained task and data to our task, despite our attempts, and this could be seen in Section 4. A natural question then rises: is there a way to fine-tune the text pre-trained model to take advantage of its pre-trained knowledge and adapt to the new task? We explore three fine-tuning options, namely: full fine-tune, soft prompting, and low rank adaptation.

3.4.1 Full Fine-tune

This is the most straight forward approach, where after text pre-trained weights are loaded for our Transformer backbone, we unfreeze it and train it all together with the randomly initialized linear input and output layers W_{in} & W_{out}. The training schema is the same as that of train-from-scratch, with a more loose hyperparameter setting. And the result is comparable with train-from-scratch. Still, full fine-tune is generally slower, we thus explore other designed fine-tuning techniques of large language models.

3.4.2 Soft Prompting

Soft prompting [5] freezes the model weights, and updates a parameterized prompt. Formally, we prepend each input prompt $P^l = (x^l_1, y^l_1, \dots, x^l_n, y^l_n, x^l_{query}) \in \mathbb{R}^{d\times(2n+1)}$ with soft prompt $P_e \in \mathbb{R}^{d\times e}$, where e is the soft prompt length. The text pre-trained Transformer backbone is then frozen at training time and only the learnable P_e is updated. For the two linear layers W_{in} & W_{out}, we directly load the fitted weights in the text pre-trained approach. There are choices here of whether to freeze the linear layers, and experiments shows slight performance increase with negligible computation add-on if we also update the linear layers. We thus train the linear layers also.

3.4.3 Low Rank Adaptation

Low Rank Adaptation (LoRA) [6] method injects small trainable rank decomposition matrices into each layers of our Transformer backbone. Let x be the input to any weighted matrices in the attention module (W_q, W_k, W_v, W_o) in any Transformer layer (LayerNorm and MLP weights are not updated in LoRA), then let $W_{(\cdot)} \in \mathbb{R}^{embed\times embed}$ be any of the four text pre-trained weighted matrices, the output is now:

$$h = W_{(\cdot)}x + \Delta Wx = W_{(\cdot)}x + BAx, \tag{3}$$

where the weight update ΔW is low-rank decomposed as $\Delta W = BA$, and $B \in \mathbb{R}^{embed\times r}, A \in \mathbb{R}^{r\times embed}$, the rank $r \ll embed$. We inject such low-rank adaptation in every layer of our Transformer backbone, and experiment with different r. Similarly to soft prompting, we load the fitted weights for W_{in} & W_{out} and tune them. The LoRA approach yields the best parameter efficient fine-tuning result in our experiments.

4 Experiments

We conduct our experiment on one NVIDIA Titan RTX 24GB GPU. The Transformer model is selected to be the default GPT-2 [2] model available on HuggingFace [1], with 12 layers of Transformer layer, 12 attention heads each Transformer layer, and embedding dimension *embed* = 768. The batch size of training input prompt is set to 64. Due to computation and time limitation, we set the dimension of the prompts d to 10, and we train/fine-tune for 15000 iterations (compared to $d = 20$, and 50000 training iterations in [1]). We use Adam optimizer, unless otherwise specified, the learning rate is set to 0.001. Same as [1], the curriculum training scheme starts with a lower dimension d and less in-context examples n in each prompt. We start with $d = 5, n = 10$ and end with $d = 10, n = 20$. d, n increase every 2000 iterations.

[1]`https://huggingface.co/docs/transformers/model_doc/gpt2`.

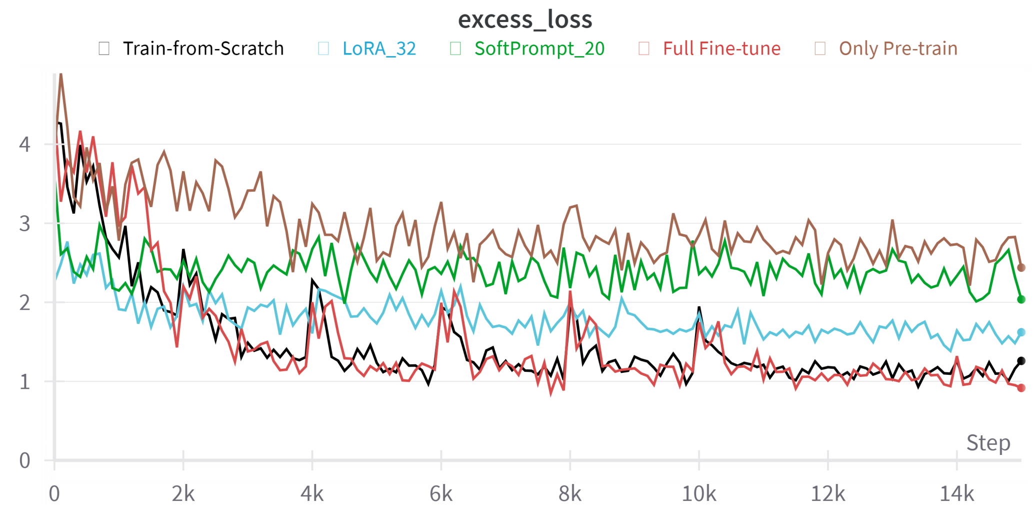

Figure 1: Loss graph for our different approaches. We here present excess loss for a more intuitive demonstration due to the curriculum training scheme. Excess loss is the training loss divided by the baseline loss given by majority vote. † wandb link: `https://api.wandb.ai/links/mstxy/xypdoh2p`.

Only pre-train (text input). This approach fails to learn the task and outputs the most common word (*e.g.* "The" and "\n") instead of numbers. Prompts that include more context are also experimented, such as changing `"y:"` to `"the number of y is:"`, but with no sign of improvement. We attribute the failure to GPT-2's "silly" way of number tokenization[2]. Essentially, numbers are divided into chunks and tokenized into unique tokens rather than being respected its decimal system. Thus, text pre-trained GPT-2 has horrendous math ability. With the failure, we do not include text input approach in the following figures or discussions.

Only pre-train (vector). Figure 1 shows the baseline performance provided by using only pre-trained weights. While the training loss is high, inferencing directly from text pre-trained weights does successfully capture the numerical relations in the input prompt. As can be seen from Figure 2, it can outperform 3-Nearest Neighbors and Averaging in linear regression task, though no where near Least Squares. Note that Least Squares represents the optimal solution for simple linear regression task, and 3-NN and averaging are consistent but sub-optimal estimators. For more details, please refer to Appendix A.3 in [1].

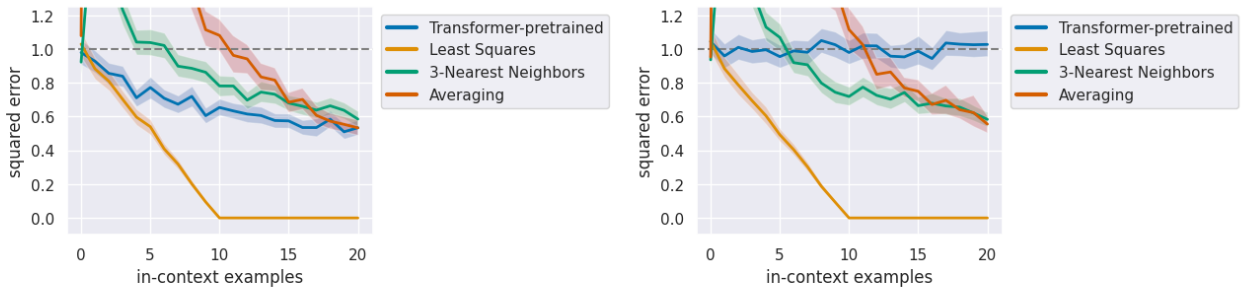

Figure 2: Test squared error on linear regression task. Left: only pre-train. Right: control experiment.

To further justify the impact of text pre-trained weights and the trivialness of learned W_{in} & W_{out}, we follow [9] and conduct a control experiment, where we randomly initialize and freeze the Transformer backbone and only train W_{in} & W_{out}. The model fails to converge, as the right figure of Figure 2 shows. We thus prove it is not W_{in} or W_{out} that is learning the task; instead, text pre-trained Transformer backbone is significant and the linear layers W_{in} & W_{out} are leveraging its knowledge.

[2]See `https://www.beren.io/2023-02-04-Integer-tokenization-is-insane/`.

Train-from-scratch vs. Full fine-tune. From Figure 1, we can see that full fine-tune converges to a similar loss as train-from-scratch. From iteration 0-2k, the pre-trained model seems to be stuck in local minima, outputing a much higher loss than train-from-scratch. This could be explained by the huge domain shift from the pre-trained text generation task to the simple numerical function fitting task. But one finding is that, training from scratch is more sensitive to the settings of hyperparameter, as a traditional learning rate of 0.001 will lead to divergence and only after tuning it down to 0.0001 will result in convergence. On the contrary, fine-tuning on text pre-trained weights is more tolerant to the learning rate, converging under both conditions. This tells that there are indeed valuable information hidden in the text pre-trained weights.

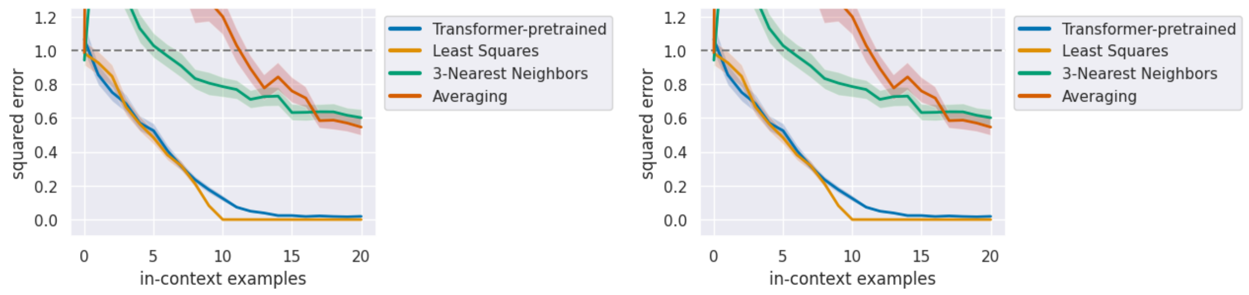

Figure 3: Test squared error on linear regression task. Left: train-from-scratch. Right: full fine-tune.

Both methods converge to the same optimal solution, as Figure 3 shows: the left figure is identical to the right. A well trained Transformers successfully recover the optimal solution given by Least Squares.

Nevertheless, if we look at the computation and parameter information in Table 1, full fine-tune takes the same computation as train-from-scratch, which is suboptimal. Therefore, we need fine-tuning techniques to better extract the pre-trained information as well as save computation.

Soft prompting. For soft prompting, note that we load fitted linear layer weights for W_{in} & W_{out}, so the loss is lower than other methods at first iterations. Still, soft prompting yields uninteresting results, giving performance only slightly better than direct inference, see Figure 4.

To investigate, we varies the length of soft prompt e with $10, 20, 100$. We expect that as the length of soft prompt increases, the model should be able to gain a more accurate instruction from the prompt and thus tune its behavior accordingly, following [5]. However, experiments show diminishing returns, as the performance improves only slightly from 10 to 20, but degrades slightly from 20 to 100, showing small and non-monotonic performance change in trainable parameters. To save space, we are not providing the loss graphs here.

Additionally, while soft prompt only increases the trainable parameters by 1% despite its length e, the GPU memory greatly increases with a larger e. We attribute such effect to the space complexity of Transformers: $\mathcal{O}(N, \max(N, embed))$, with $N = 2n + e$, increases linearly as the length of soft prompt e increases. Moreover, as we use a naive implementation of soft prompt, its impact on memory consumption is expected to be higher than well-optimized HuggingFace's GPT-2.

Overall, we conclude that soft prompting does not fit well in the context of large domain shift (or complete out-of-domain fine-tune), as the intention of prompt is to help instruct the extraction of *known* information from the pre-trained model.

Low rank adaptation. Contrary to the poor added performance of soft prompting, low rank adaptation (LoRA) boosts the fine-tuning performance to a great level, with only negligible added trainable parameter size. Similar to soft prompting, we experimented with different intermediate rank r. A larger r does give better performance, but only sub-one percent improvement, so we choose 32 as the optimal approach, anticipating a more robust model in other conditions. Note here an increase in r does not lead to significant GPU memory increase, this is due to LoRA [6]'s VRAM optimization.

Figure 4 shows a much larger improvement compared to soft prompting. This fits the intuition that directly learning from evidence is more effective than receiving instruction from others on what to do. One could argue here that LoRA outperforms soft prompting due to the increased number of parameters. Indeed, when

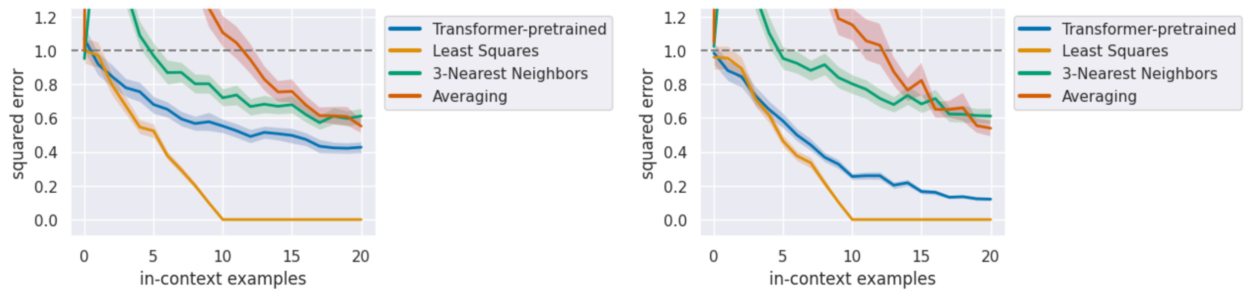

Figure 4: Test squared error on linear regression task. Left: soft prompting. Right: low rank adaptation.

experimenting on $r = 1$ (0.05 M parameters, the lowest we could get compared to soft prompting's 0.02 M), the model deteriorates to a similar loss as soft prompting. Still, through the above experiments, we show that LoRA has a much higher expected performance and is suitable for out-of-domain fine-tune; while soft prompting's fine-tuning performance is capped, invariant to its length e.

Method	Specs	Trainable Params $\mathcal{T}$	Total Params $\mathcal{P}$	$\mathcal{T}/\mathcal{P}$	GPU Memory
Only Pre-train	—	0.01 M	124.45 M	0.01%	1991 MB
Train-from-Scratch Full Fine-tune	—	124.45 M	124.45 M	100.0%	3198 MB
Soft Prompt	10	0.02 M	124.46 M (+0.01 M)	0.01%	3021 MB
	†20	0.02 M	124.46 M (+0.01 M)	0.01%	3018 MB
	100	0.02 M	124.46 M (+0.01 M)	0.01%	8720 MB
LoRA	1	0.05 M	124.49 M (+0.04 M)	0.04%	2675 MB
	8	0.30 M	124.74 M (+0.28 M)	0.24%	2672 MB
	†32	1.19 M	125.63 M (+1.18 M)	0.95%	2688 MB
	128	4.73 M	129.17 M (+4.72 M)	3.66%	2712 MB

Table 1: Parameter information and GPU memory usage for each method. Specs for Soft Prompt is the length of soft prompt e; Specs for LoRA is the intermediate rank r. $^\dagger\cdot$ represents the spec with best accuracy. GPU Memory is retrieved during the first episode of curriculum training.

5 Discussion & Conclusion

Why Transformers work for In-Context Learning? Though not explored in our project, we would like to discuss Transformer's in-context learning ability. An analogy could be drawn from the paper *Matching Networks for One Shot Learning* [10]. In the paper, the prediction y_{query} on x_{query} is generated from:

$$y_{query} = \sum_{i=1}^{n} a(x_i, x_{query}) y_i, \tag{4}$$

where $a(\cdot,\cdot)$ is an attention mechanism, and $\{x_i, y_i\}_{i=1}^{n}$ are the support examples of each class. This is very similar to a single forward pass of the Transformers model, where all the supporting (x_i, y_i) pairs are fed into the Transformers model all at once and the attention between tokens are computed. In a task as simple as linear regression without intercept, we could compute y_{query} as:

$$y_{query} = \langle \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2}, x_{query} \rangle = \frac{\sum_{i=1}^{n} \langle x_i, x_{query} \rangle \cdot y_i}{\sum_{i=1}^{n} x_i^2}, \tag{5}$$

which is analogous to (4), where we could think of each y_i as an individual class. With the built-in global attention structure and its pre-trained weights, Transformer could effectively extract related features and aggregate them for a new query, thus working well in a meta-learning/few-shot learning task, as our work shows.

With the above analysis, we thus think it is a promising direction to replace the Transformers model with LSTMs [11] and compare the in-context learning results. Will LSTMs totally fail due to a very different inductive bias? Or could it succeed in certain in-context learning tasks? Moreover, adapting and applying LLM fine-tuning techniques into LSTMs might shed some light on understanding both LSTMs and LLMs.

In conclusion, our work investigates the in-context learning ability of text pre-trained Transformers. Contrary to training from scratch, which provides an upper bound to Transformers in-context learning ability, we address a more realistic scenario, directly inferencing from pre-trained weights, with both text format input and numerical vector format input, and shows that a baseline performance could be met without any update to the Transformer backbone. We further fine-tune the model with a variety of methods to demonstrate an achieve-able performance, using full fine-tune, soft prompting and low rank adaptation. A comparison of their efficacy has been shown by the extensive experiments. The fine-tuning approaches offer intuitive thoughts on how well a text pre-trained Transformer could behave in out-of-domain in-context learning setup with limited modifications.

Moreover, the comparison between different fine-tuning techniques has significance not only in the scope of learning to do in-context learning using Transformers, but also extends to any fine-tuning tasks of Transformers.

Last but not least, there are many possibilities to further develop our research, apart from the aforementioned comparison between Transformers and LSTMs. On one hand, because fine-tuning techniques such as soft prompting and LoRA are largely designed to target a variety of down-stream tasks, in-context learning tasks that incorporates a mixture of sub-tasks could be utilized. This also fits the more practical in-context learning setup, where LLMs are prompted with different tasks at inference, under the same pre-trained backbone, and only potentially different heads. On the other hand, in-context tasks based on text that are beyond numerical analysis should also be investigated more deeply, which is slightly touched upon in our project. This could involve tasks such as logical reasoning and deduction, but they require a systematic task generation schema to be devised, as textual tasks are less bounded in prompt length or structure.

References

[1] S. Garg, D. Tsipras, P. S. Liang, and G. Valiant, "What can transformers learn in-context? a case study of simple function classes," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 30 583–30 598.

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[4] OpenAI, "Gpt-4 technical report," 2023.

[5] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, Eds. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 3045–3059.

[6] E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *International Conference on Learning Representations*, 2022.

[7] F. Rong, "Extrapolating to unnatural language processing with gpt-3's in-context learning: The good, the bad, and the mysterious," 2021. [Online]. Available: https://ai.stanford.edu/blog/in-context-learning/

[8] X. L. Li and P. 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)*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Association for Computational Linguistics, Aug. 2021, pp. 4582–4597.

[9] B. Z. Li, M. Nye, and J. Andreas, "Implicit representations of meaning in neural language models," 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)*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Online: Association for Computational Linguistics, Aug. 2021, pp. 1813–1827.

[10] O. Vinyals, C. Blundell, T. Lillicrap, k. kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, nov 1997.

Self Review

Main Goal. Our project is inspired by the study titled "What Can Transformers Learn In-Context? A Case Study of Simple Function Classes." We aim to address its lacking of practical value in its training approach, which involves starting from scratch. Thus, our goal is to explore the in-context learning capability of a text pre-trained language model, specifically GPT-2, in the context of a linear regression task.

Claims. We claim that, despite the substantial domain shift in task, there are valuable knowledge in text pre-trained weights that can be leveraged, through both direct inference and fine-tuning.

Experiments. To support our claim, we propose different ways to extract useful information: direct inference from text pre-trained weights using either vector input or converted text input, full fine-tuning, parameter efficient fine-tuning including soft prompting and low rank adaptation. These approaches are experimented on linear regression in-context learning, with varied hyperparameters.

Evaluation Protocol. Our evaluation is based on computation (trainable parameter size, total parameter size, and GPU memory usage) and in-context learning accuracy (the squared error on test data, and comparison to 3-Nearest Neighbors & least squares & majority vote on the same data).

Data & Task. As our project works with linear regression task, the data are sampled through Gaussian distribution. For details on data and task, see 3.1.

Experiment Effectiveness. The experiments show both failure and success. And in the analysis and comparison, the reason behind failure and success are analyzed, explaining why some techniques do not work and the others work with both theoretical investigation and intuitive interpretation.

Limitations. We do include our limitations, such as the decrease of dimension and examples due to our limitation of time and computation resources. We also include future directions to extend our work.

Paper Strength. The paper shows a very systematic exploration and logical progression from one method to the other. Each method is well explained, and corresponding experiments are evaluated and analyzed reasonably and comprehensively. Additionally, the discussion at the end offers some interesting insight into the subject matter.

Paper Weakness. It is relatively long and extensive, might be unfriendly to outside viewers. The bullet points at the end of the introduction section might not summarize our contribution and finding so well. And for experiments it only investigates simple linear regression.

Suggestions. Could experiment with more in-context learning tasks, as the original paper [1] does. And make the introduction more concise and informative.

Related Works. They are all discussed in Section 2.

Reproducibility. We are confident that all experiments, results, are reproducible

Interpretability. All plots are supported with detailed caption, and each plot is detailedly discussed in the main body. Methodology are also clearly stated in a whole section.

Writing. The writing is clear, though sometimes informal, sometimes repetitive.

TODOs. Improve the introduction section, and proofread the overall writing.