Transformers as Learners, Not Just Models: A Functional Gradient Descent View

Jianren Zhou
TU Munich
`go76cop@mytum.de`

Abstract

This report is for a better understanding of the paper Transformers Implement Functional Gradient Descent to Learn Non-Linear Functions In Context and the related theoretical development of In-Context Learning. Firstly, this report will **illustrate the development of In-Context Learning** and the **connection between Transformer and some algorithms** such as Gradient Descent. Based on the above paper, this article will derive the Proposition 2 in Cheng et al. [2024] and the **error bound** between the finite layer Transformer model and the Bayes Optimal. The proof of the composition of the activations will also be discussed. Some experiments have also been added to prove the theoretical analysis. At last, this report will **analyze the theories and experiments** in the paper. Some experiments will be done again to make sure all of the results are reproducible and faithful to the theoretical setting. The reproducing code can be found in `https://github.com/Anthooooony1105/TF-FGD`.

1 Literature Review

In this section, this paper will introduce the development of the In-Context-Learning and the connections between the Transformer model and the algorithms such as Gradient Descent.

1.1 In-Context-Learning development

Vaswani et al. [2017] introduced the Transformer architecture, which relies on self-attention mechanisms for the first time, achieving state-of-the-art results in machine translation and other sequence modeling tasks. Winata et al. [2021] explained the concept of In-Context-Learning, showing that GPT-3 can condition on examples in the input prompt without gradient updates. Zhao et al. [2021] pointed out that the ICL is extremely sensitive to example selection, order, and format, and introduced contextual calibration. After that, Min et al. [2022] introduced MetaICL, which is a meta-learning method that trains language models to ICL across diverse tasks. Also, Coda-Forno et al. [2023] Connected ICL to meta-learning theory, arguing that Transformers generalize across tasks in a way similar to meta-learners. At the same year, Chan et al. [2022] studied how Transformer model generalize differently when leveraging in-context information vs adapting through weight updates. Also, Olsson et al. [2022] discovered "induction heads" that mechanistically underlie in-context learning. In the next year, Zhou et al. [2024] proposed batch calibration to mitigate label bias and improve reliability in ICL predictions. What's more, Dong et al. [2024] provided the first comprehensive survey that synthesizes definitions, mechanisms, applications, and open challenges of ICL. And Arora et al. [2024] derived theoretical error bounds and scaling laws that describe how ICL performance depends on prompt length, task number, and discretization.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

1.2 The relationship between Transformer and algorithms

Many papers just illustrated how Transformers working but not learning. Motivated by this question, a number of papers came out and can be classified into Bayesian / Kernel Regression Perspective(Choromanski et al. [2022]), Algorithmic / Stability Perspective(Li et al. [2023];Yang et al. [2024]), and Gradient Descent Perspective.

The relationship between the neural network architectures and algorithms has been researched for a long time. At a very early age, H. et al. [1995] studied the Turing completeness of RNNs. Despite the computational power, training RNN was a challenge. Xie et al. [2022] proposed a theoretical framework which viewed the ICL as implicit Bayesian inference. Garg et al. [2023] conducted, which kind of function classes can Transformers learn in ICL. Akyürek et al. [2023] showed for the first time that the Transformer can simulate gradient descent and ridge regression when solving linear regression tasks. After that, von Oswald et al. [2023] demonstrated that Transformers implement in-context learning by performing gradient descent in function space for the first time, which form a base for the research of gradient descent in Reproducing Kernel Hilbert Space. At the same year, Dai et al. [2023] viewed ICL as an implicit finetuning via meta-optimization. Recently, Chen et al. [2025] provides rigorous mathematical guarantees for ICL applied to non-linear elliptic partial differential equations and takes the PDE into consideration for the first time.

2 Proposal for next steps

In this section, this report will illustrate the proposals for the next steps from two perspectives, theoretically and experimentally. At the first subsection, the analysis of the finite-layer Transformer will be discussed, which is an extension of the infinite-layer Transformer. If the layer number converges to infinity, the Transformer's prediction approaches the Bayes optimal. The basic introduction to Bayes Optimal and the proof of the Proposition 2 in paper Cheng et al. [2024] are shown in 2.1.1 and 2.1.2. But in reality, the layer number can not go to infinity. So in subsection 2.1.3 the analysis on the error bound of the finite-depth Transformer and the Bayes optimal is shown. After that, this report will discuss multi-head attention in subsection 2.2. In the future directions in Cheng et al. [2024], it's shown that the power via composition of the attention-head in multi-head Transformer can be analyzed. It's believed the combination can bring Transformer a stronger learning ability. So it's important to investigate the combination of the heads. In this report, some basic experiments have been done as the first-step investigation of the theoretical and practical extension.

2.1 Finite Transformer's error bound

Proposition 2 is based on the infinite-layer Transformer in the Cheng et al. [2024]. But the Transformer structure will not be built with infinite layers in reality. So in 2.1 the analysis of the finite Transformer and the error bound between it and the Bayes optimal will be included. In 2.1, some lemmas(What is Bayes Optimal, Why infinite layer Transformer's prediction approaches the Bayes optimal) will be given in 2.1.1 and 2.1.2 respectively. And in 2.1.3, some proof will be given for the error bound.

2.1.1 Bayes Optimal

In Proposition 2, it was proved that as the $l \to \infty$, the Transformer's prediction for $y^{(n+1)}$ approaches the Bayes Optimal. The definition of the Bayes Optimal will be illustrated as follows. From the multi-variable Gaussian distribution, we can obtain the conditional distribution as follows:

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\right) \tag{2.1}$$

From the above distribution, we can write down the conditional distribution of $Y_2|(Y_1 = y_1)$ can be written as

$$Y_2|\left(Y_1 = y_1\right) \sim N\left(\mu_2 + \Sigma_{21}\Sigma_{11}^{-1}\left(y_1 - \mu_1\right), \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}\right) \tag{2.2}$$

For the above situation, the problem can be written as:

$$\begin{bmatrix} \hat{Y} \\ y^{(n+1)} \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \hat{K} & v \\ v^T & K_{(n+1)(n+1)} \end{bmatrix}\right), \tag{2.3}$$
$$where\ \hat{K} = K_{n\times n}$$

And the conditional distribution can be written as:

$$y^{(n+1)}|(Y_1 = \hat{Y}) \sim N\left(V^T \hat{K}^{-1}(Y-0), K_{(n+1)(n+1)} V^T \hat{K}^{-1} V\right) \tag{2.4}$$

For the $y^{(n+1)}$, the Bayes optimal can be written as:

$$V^T \hat{K}^{-1} \hat{Y} = \Sigma_{j,k=1}^{n} K\left(x^{(n+1)}, x^{(j)}\right) \hat{K}_{jk}^{-1} y^{(k)} \tag{2.5}$$

2.1.2 Proposition 2 with infinite layers

Considering the construction in Proposition 2, the notations can be written as:

$$r_l = \delta \quad for \quad all \quad l; \quad 0 < \delta < ||\hat{k}||_2, for \quad l = 0...k;$$
$$y_l^{(i)} := [Z_l]_{(d+1),i}, let \quad \hat{Y}_l := [y_l^1, ..., y_l^{(n+1)}]$$

With these settings and the definition from chapter 2.3 in Cheng et al. [2024], the ICL problem can be written as:

$$\begin{aligned} Z_{l+1} &= Z_l + V_l Z_l M \hat{h}\left(Q_l Z_l, K_l Z_l\right) \\ &= Z_l + \begin{bmatrix} 0 & 0 \\ 0 & -\delta \end{bmatrix} \begin{bmatrix} X \\ \hat{Y}_l \end{bmatrix} K(X, X) \\ &= Z_l + \begin{bmatrix} 0 \\ -\delta \hat{Y}_l \end{bmatrix}^T \begin{bmatrix} \hat{K}_{(1:n,1:n)} & V \\ V^T & \hat{K}_{(n+1,n+1)} \end{bmatrix} \\ &= Z_l + \begin{bmatrix} 0 & 0 \\ \left(-\delta \hat{Y}_l \hat{K}_{(1:n,1:n)}\right) & \left(-\delta \hat{Y}_l \hat{K}_{(n+1,n+1)}\right) \end{bmatrix} \end{aligned} \tag{2.6}$$

For the last row in (6) , by recursion it can also be written as:

$$\hat{Y}_l = \left(I - \delta \hat{K}\right) \hat{Y}_l = \left(I - \delta \hat{K}\right)^l \hat{Y} \tag{2.7}$$

For the last element $y_{l+1}^{(n+1)}$, by recursion it can be written as:

$$\begin{aligned} y_{l+1}^{(n+1)} &= y_l^{(n+1)} - \delta V^T \hat{Y}_l \\ &= y_{l-1}^{n} - \delta V^T Y_{l-1} - \delta V^T \hat{Y}_l \\ &= y_0^{(n+1)} - \delta V^T \hat{Y}_0 - ... - \delta V^T \hat{Y}_l \\ &= -\delta V^T \Sigma_{k=0}^{l} \hat{Y}_k = -\delta V^T \Sigma_{k=0}^{l} \left(I - \delta \hat{K}\right)^k \hat{Y} \end{aligned} \tag{2.8}$$

With the setting $\delta = \frac{1}{n}$ and Neumann series equation, which has been discussed in Appendix A. It can be proved that the $y_{l+1}^{(n+1)}$ happens to be the Bayes optimal:

$$\begin{aligned} y_{l+1}^{(n+1)} &= -\delta V^T \sum_{k=0}^{l} \left(I - \delta \hat{K}\right)^k \hat{Y} \\ &= -V^T \left(\underbrace{\frac{1}{n} \sum_{k=0}^{l} \left(I - \frac{1}{n} \hat{K}\right)^k}_{\text{Neumann series}} \right) \hat{Y} \\ &= -V^T K^{-1} \hat{Y} \end{aligned} \tag{2.9}$$

The result in (9) happens to be the Bayes optimal in (5). It can be proved that the Transformer's prediction is the Bayes optimal when the layer number comes to infinite.

2.1.3 Proposition 2 with finite layers

The above proof has illustrated the Transformer's prediction for $y^{(n+1)}$ approaches the Bayes optimal. But in reality, the Transformer architecture can not have infinite layers. So in this subsection, the finite situation will be discussed, aiming to find the error bound when the Transformer only has finite depth. The derivation of the error bound of the finite layer Transformer will be illustrated as follows:

Proposition 2.1. *Fix a stepsize $\delta > 0$. Consider the depth-L predictor defined by the recursion that the Transformer implements when its attention nonlinearity matches the kernel K. The error bound between the L-layer Transformer and the Bayes Optimal satisfies the following inequality:*

$$|y^* - \hat{y}_l| \leq \frac{||v||_2||\hat{Y}||_2}{\lambda_{min}}(\frac{\kappa - 1}{\kappa + 1})^L \tag{2.10}$$
$$where \quad the \quad \lambda_{min} = min(eig(\hat{K}))$$

Proof. In the above content 2.1.2, it has been shown that the Transformer model can be written like 2.8. If the layer number doesn't come to infinity, it can be written in this way:

$$y_{l+1}^{(n+1)} = -\delta V^T \Sigma_{k=0}^{l} (I - \delta \hat{K})^k \hat{Y} \tag{2.11}$$

So the difference between the L-layer Transformer and the Bayes Optimal can be calculated in this way:

$$\begin{aligned} y^* - \hat{y}_L &= V^T[\hat{K}^{-1} - \delta \Sigma_{k=0}^{L-1}(I - \delta\hat{K})^k]\hat{Y} \\ &= V^T[\delta\Sigma_{k=0}^{\infty}(I - \delta\hat{K})^k - \delta\Sigma_{k=0}^{L-1}(I - \delta\hat{K})^k]\hat{Y} \\ &= V^T[\delta\Sigma_{k=L}^{\infty}(I - \delta\hat{K})^k]\hat{Y} \\ &= V^T[\delta(I - \delta\hat{K})^L\Sigma_{k=0}^{\infty}(I - \delta\hat{K})^k]\hat{Y} \\ &= V^T(I - \delta\hat{K})^L[\delta\Sigma_{k=0}^{\infty}(I - \delta\hat{K})^k\hat{Y}] \\ &= V^T(I - \delta\hat{K})^L\hat{K}^{-1}\hat{Y} \end{aligned} \tag{2.12}$$

To obtain the norm bound, use the Cauchy-Schwarz inequality of the operator norm

$$|y^* - \hat{y}_l| \leq ||v||_2||(I - \delta\hat{K})^L||_2||\hat{K}^{-1}||_2||\hat{Y}||_2 \tag{2.13}$$

Because $\hat{K}$ is symmetric positive semidefinite matrix, the $||\hat{K}^{-1}||_2 = \frac{1}{\lambda_{min}}$, $||(I - \delta\hat{K})^L||_2 = ||(I - \delta\hat{K})||_2^L$ and $||(I - \delta\hat{K})||_2 = max_i|1 - \delta\lambda_i| = (1 - \delta\lambda_{min})^L$. Using the minimax step $\delta^* = \frac{2}{\lambda_{max}+\lambda_{min}} = \frac{2}{\lambda_{min}(1+\kappa)}$. The inequality can be written as:

$$|y^* - \hat{y}_l| \leq \frac{||v||_2||\hat{Y}||_2}{\lambda_{min}}(\frac{\kappa - 1}{\kappa + 1})^L \tag{2.14}$$

□

2.1.4 Experiments on Proposition 2.1

This subsection aims to prove Proposition 2.1 with some experiments to illustrate that the above formula 2.14 is the error bound. The hypothesis to be proven is inequality 2.14. Some settings will be shown as follows.

In the experiment, the data is generated. The training data points $X = \{x^{(i)}\}_{i=1^n}$ are drawn from standard Gaussian vectors. The query data point x^{n+1} is drawn independently. And the labels $\hat{Y}$ are also drawn from a Gaussian. The RBF kernel $K(x, y) = exp(-||x - y||^2/(2\delta^2))$ is used in this experiment, which guarantees a PSD Gram matrix. And the Bayes Optimal can be written as $y^* = v^T\hat{K}^{-1}\hat{Y}$ and the finite layer Transformer is $\delta v^T\Sigma_{k=0}^{L-1}(I - \delta\hat{K})^k\hat{Y}$. And two step sizes will be tested:$\delta_{optimal} = \frac{2}{\lambda_{min}+\lambda_{max}}$ and $\delta_{safe} = \frac{1}{\lambda_{max}}$. Both of these steps will be tested, and the converging speed will also be compared.

After the experiment, the result can be found in Figure 1. In Figure 1, the blue solid line means the actual error when using the optimal step $\delta^* = 2/(\lambda_{max} + \lambda_{min})$. The Green solid line stands for the when using the safe step $\delta = 1/\lambda_{max}$. The orange dashed line means the theoretical bound for the optimal step size, and the red dashed line means the theoretical error for the safe step size.

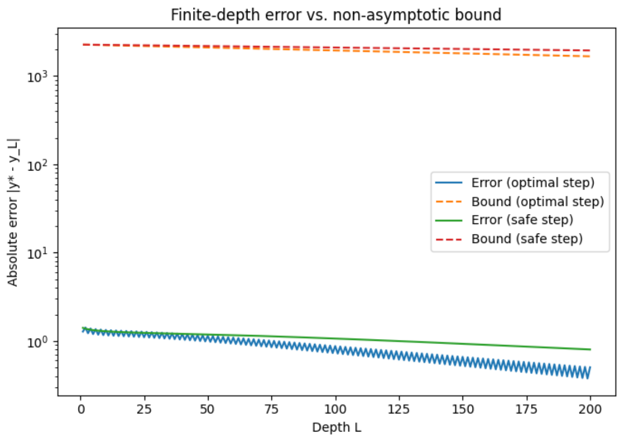


Figure 1: Finite TF vs Bayes Optimal (two step sizes)

It can be seen clearly in the graph that the bounds(dashed lines) are above the errors(solid lines). So this experiment can come to the conclusion that the inequality 2.14 stands. What's more, the blue curve(optimal step) decays more steeply than the green curve(safe step), which means the optimal step converges faster than the safe step. And all of the errors are decreasing with depth L, which matches the theory that the deeper Transformers get closer to the Bayes predictor.

2.2 Composition kernels

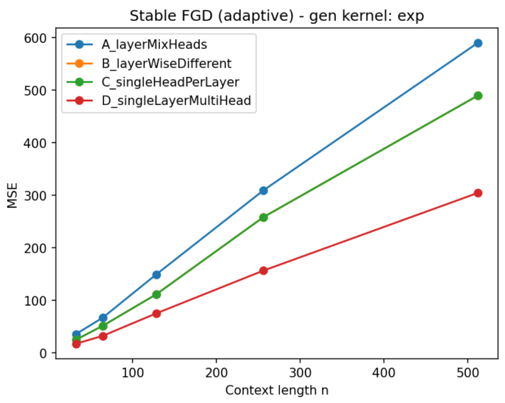
At the future directions in Cheng et al. [2024], the composition of the kernels are mentioned to investigate. So, in this report, some basic research will be done to illustrate that multi-head attention corresponds to a composite kernel, while stacking layers realizes iterative Landweber updates. Besides, the identical heads add no expressivity, and for large context length n, the composition multi-head kernels strictly outperform the stacking layers by having a better expressivity. In subsection 2.2.1, this report will do some experiments on kernel composition. It will be shown that the ability to learn by compositing the kernels is much more important than the stacking of the layers. In subsection 2.2.2, the report will illustrate that the ability will not change with the identical heads in the same layer.

2.2.1 Kernel composition experiments

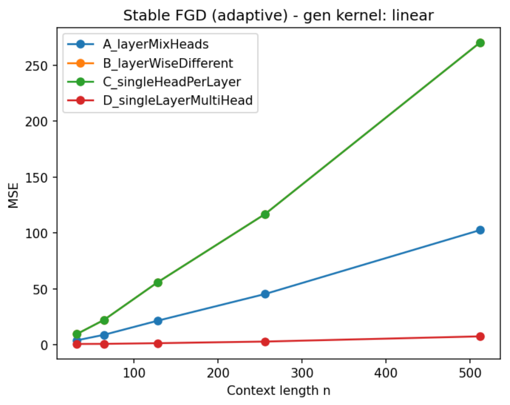
What are the most important factors influencing the Transformer architecture, the layer number, or the composition of the kernels? This is a good question to think about. In this subsection, the experiment is designed to test which is significant. The report put forward a hypothesis *Within-layer mixtures of heterogeneous kernels learn target functions more effectively than deeper stacks of homogeneous kernels* and design the experiment to test it.

In the experiment, the experimental task is generated from a kernelized Gaussian process episode. The unit-normalized random vectors are sampled in $X \in R^{d*(n+1)}$ with $d = 8$. The outputs y are drawn from a zero-mean Gaussian process with the covariance is defined by a ground-truth kernel, chosen from the set $\{linear, ReLU, exp, softmax\}$. And the context size n varied across $\{4, 8, 12, 16\}$. What's more, the predictor is based on the functional gradient descent(FGD) formula, where each layer applies a kernel map to the context data and performs a residual update on predictions with learning rate η. In multi-head settings, the kernel matrix produced by different heads is averaged within a layer to construct the effective kernel used for the update. The final prediction for the query point is obtained by applying the learned coefficients against the averaged kernel between the query and the context set.

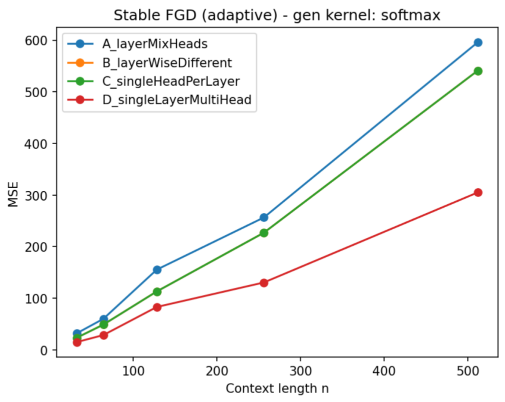
In the output, for model configurations are compared. Config A (LayerMixHeads) uses four layers, each with four heterogeneous heads-linear, ReLU, exponential, and softmax-like kernels are averaged within the layer. Config B (LayerWiseDifferent) has four layers, each containing four identical heads, but the kernel type changes across layers in the order linear→ReLU→exp→softmax-like. Config C (SingleHeadPerLayer) also has four layers but with a single head in each, again using the same ordered sequence of kernels. Finally, Config D (SingleLayerMultiHead) consists of one layer containing four heads, each a different kernel, whose outputs are averaged to form the effective kernel for that single layer. And the results are listed in Fig. 2



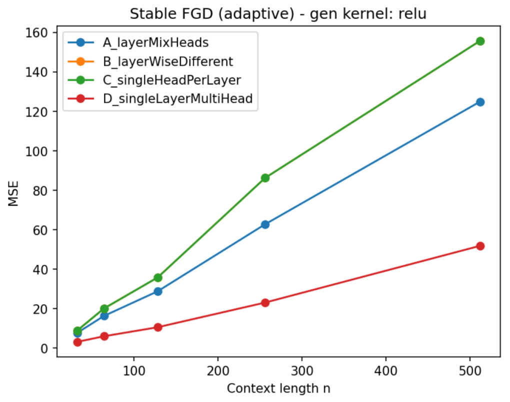
(a) fig:a



(b) fig:b



(c) fig:c



(d) fig:d

Figure 2: Comparison of MSE across different generation kernels (exp, linear, softmax, relu).

From Fig. 2, it's obvious that the single-layer multi-head line(red line D) performs best. That's to say, the learning ability of the Transformer model mostly depends on the composition of the activations, which enable the model to learn different kinds of non-linearity. Compared with the stacking of the layers(Green line C), the multi-head attention with combined activations has a stronger learning ability than the activation-stacked model. Layer stacking corresponds to repeated iterative updates, but unless the data distribution requires multi-step refinement, the expressivity unlocked by parallel activation composition is far more decisive for performance. This experiment can prove the hypothesis above.

Although four configurations are drawn in the graph, the $B_{layerWiseDifferent}$ line disappears because it's covered by the $C_{singleHeadPerLayer}$. That's to say the line B and C are the same in Fig. 2, which will be proved in 2.2.2.

2.2.2 Identical heads experiments

From Fig. 2, it's easy to find that the line B is covered by line C. To avoid the randomness, this report also extend the context length n to 1024 and obtain the result in Fig. 3.

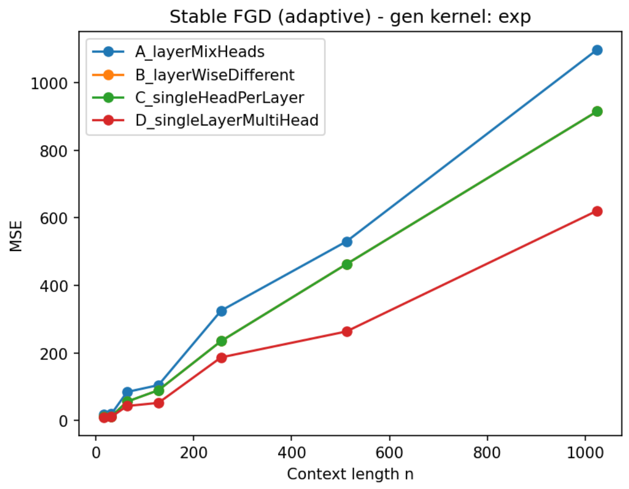


Figure 3: Extended context length experiment

In Fig. 3, it's obvious that the yellow line is also covered by the green line. That's to say, if only using the identical heads in the multi-head attention. The output is the same as the single-head one. That's easy to prove. The $B_{LayerWiseDifferent}$ line can be written in 2.15

$$f_{l+1} = f_l + \Sigma_{s=1}^4 r'_{l,s} \Sigma_i (y^{(i)} - f_l(x^{(i)}))K_l(\cdot, x^{(i)}) \tag{2.15}$$

And the line $C_{singleHeadPerLayer}$ can be written as 2.16

$$f_{l+1} = f_l + \bar{r}'_i \Sigma_i (y^{(i)} - f_l(x^{(i)}))K_l(\cdot, x^{(i)}) \tag{2.16}$$

From 2.15 and 2.16, the difference lies in :

$$\bar{r}'_i = \Sigma_{s=1}^4 r'_{l,s} \tag{2.17}$$

The equation 2.17 illustrates the reason why these two curves are the same. 2.16 is an extension of the 2.15 with the step $\bar{r}'_i$. So essentially, using the same head in one multi-head attention block is an update to the step size. It doesn't increase the learning ability or the iterative steps. So line B and line C are the same in Fig. 2.

3 Critical thoughts of the results

After reading the paper Cheng et al. [2024], some critical thoughts arise. They cover the reality of the hypothesises and the future directions for the improvement. Besides, some of the analysis on the experiments will also be discussed. And in the last part, this report will reproduce some of the experiments to find out whether the experiments are reproducible or not.

3.1 The application of theory in reality

First of all, the practical application ability of the theories in Cheng et al. [2024] will be analysed. In the above paper, the attention activation functions correspond with the kernels from which the data are generated. But in reality, specially in Computer Vision and Natural Language Processing, the data points may not be generated from special kernels, and finding the corresponding activation function might be a great challenge. So this report thinks there would be researched more on how to find the kernel or the distribution in the real dataset. On the other aspect, it would be better if the hypothesis can be loosened a bit in practice. It would be better if relaxing the requirements for kernel function matching would be researched. For example, by examining the performance degradation when considering approximate rather than exact matching with functions. Introducing data distribution assumptions that are closer to real-world tasks, such as high-dimensional sparse vectors and long-tail distributions, are also worth researching.

For the experiments, it's significant to obtain the conclusion that Transformer implements Functional Gradient Descent and prove that the Transformer model converges to Bayes Optimal when the attention activation function $\tilde{h}$ corresponds with the data generating kernels. And the ReLU and Softmax Transformer has been proved in the above paper. However, there would be a big difference between real-life data and the theoretically generated data. So it would also be a little hard to use the conclusions in the paper directly. Moreover, the Transformer in the paper is subject to many restrictions and needs to be analyzed under specific parameter settings. If the proof could be carried out with fewer constraints, the results would be more general and effective.

3.2 Experiments reproducibility test

With respect to the reproducibility presented in the paper, the following tests were carried out in this report. However there is no code attached in the paper. The experiments can only be completed with the new code. And the results will be discussed in the following paragraphs. The code is stored in the GitHub `https://github.com/Anthoooooony1105/TF-FGD`. And the results are shown in the following figures.

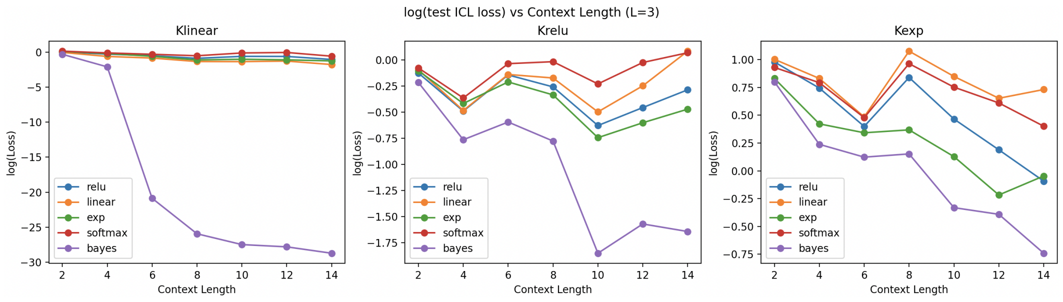


Figure 4: Experiments reproducing 3.2.1

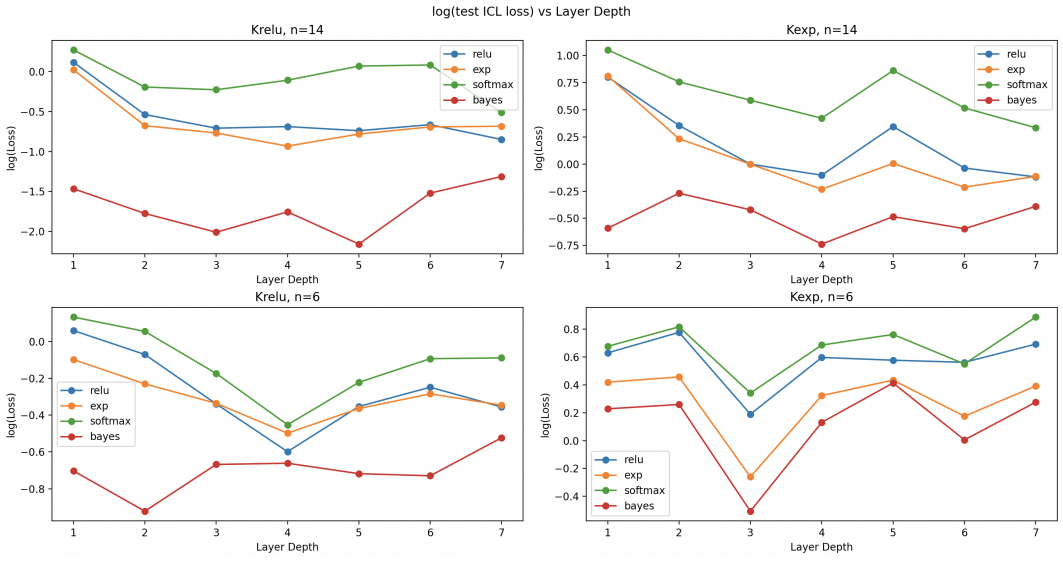


Figure 5: Experiments reproducing 3.2.2

For the experiments in section 3.3 in Cheng et al. [2024], the first figure illustrate how the choice the activation function $\tilde{h}$ affects the Transformer's performance under different data-generating kernels. In Fig. 4, the result has been reproduced. From the left and right figures, it can be found that the linear activation in the left figure and the exponential activation in right figures perform well, which corresponds with the result in Proposition 2. But there is a small error when the experiment comes to ReLU activation in the middle figure. This phenomenon might be caused by the settings of the experiment. The Proposition 2 can still be proved by this group of experiments.

This report also tests in-context loss against the number of layers, under different numbers of demonstrations. In Fig. 5, the right two exponential activation figures can prove the proposition 2 well, in which the exponential activation Transformer behaves well when matched with the exponential kernel-generated data. There is still a small error with the ReLU activation Transformer in the left two figures. But when the Layer Depth is around 4, the proposition holds and the error is not very high when the Layer Depth is in other case. Perhaps it's due to the settings in the code. But both of the Fig. 4 and Fig. 5 can prove the proposition 2 in Cheng et al. [2024].

For the multi-head attention situation in section 3.4 of Cheng et al. [2024], the experiment has also been designed to test the proposition 3 and the result have been displayed in Fig. 6. In the left figure, the lines are almost the same. All models(linear, exp, and 2-head linear+exp) perform similarly. It indicates that a single linear head is sufficient when the data is generated by a linear kernel. The multi-head setting provides no additional benefit in this case. But the learning ability is not decreased compared with other two kinds of activations. The figure in the middle can prove that the 2-head Transformer improves the learning ability because the green line(2-head, linear + exp) is the closest to the red line(Bayes Optimal). And the figure in the right explains the learning ability of mixed heads is better than the other single heads when the layer depth is not very high. As the layer depth increasing, the curve is also getting close to Bayes Optimal. So these results confirm Proposition 3: a multi-head Transformer can combine different activations to emulate composite kernels, thereby outperforming single-head Transformers in more complex data distributions.

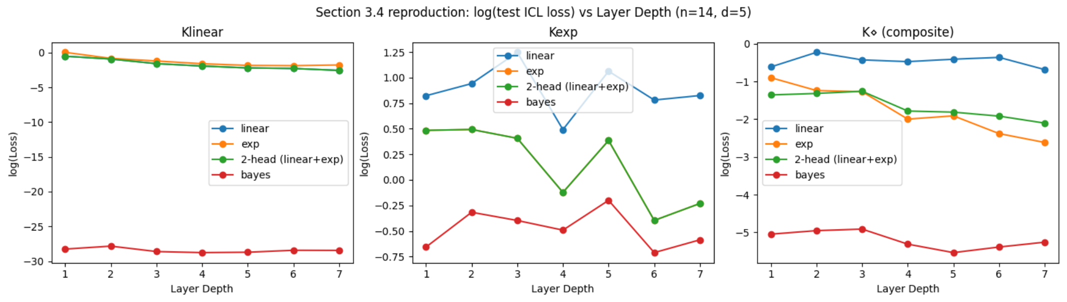


Figure 6: Experiments reproducing 3.4

For the experiments in chapter 4 of the paper Cheng et al. [2024], this report thinks they are consistent with the theoretical framework. By designing experiments under controlled settings, the paper shows that empirical findings align well with the stationary points established in the analysis. The experimental design is clear and the related problems in the experiments are pointed out. Although the experiments are conducted under idealized settings, they demonstrate that the optimization dynamics identified in the theoretical results are not only mathematically plausible but also observable in controlled training environments. These findings have broader implications: they explain why large language models are capable of rapid task adaptation without parameter updates, and they suggest practical directions for applying in-context learning in real-world domains such as natural language processing, scientific computing, and personalized AI systems.

References

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models, 2023. URL `https://arxiv.org/abs/2211.15661`.

Aryaman Arora, Dan Jurafsky, Christopher Potts, and Noah D. Goodman. Bayesian scaling laws for in-context learning, 2024. URL `https://arxiv.org/abs/2410.16531`.

Stephanie C. Y. Chan, Ishita Dasgupta, Junkyung Kim, Dharshan Kumaran, Andrew K. Lampinen, and Felix Hill. Transformers generalize differently from information stored in context vs in weights, 2022. URL `https://arxiv.org/abs/2210.05675`.

FirstAuthor Chen, SecondAuthor CoauthorA, and ThirdAuthor CoauthorB. Provable in-context learning for elliptic pdes. *Journal or Conference Name*, xx(yy):pp–qq, 2025. doi: 10.xxxx/xxxxx. URL `https://...`

Xiang Cheng, Yuxin Chen, and Suvrit Sra. Transformers implement functional gradient descent to learn non-linear functions in context, 2024. URL `https://arxiv.org/abs/2312.06528`.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2022. URL `https://arxiv.org/abs/2009.14794`.

Julian Coda-Forno, Marcel Binz, Zeynep Akata, Matthew Botvinick, Jane X. Wang, and Eric Schulz. Meta-in-context learning in large language models, 2023. URL `https://arxiv.org/abs/2305.12907`.

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers, 2023. URL `https://arxiv.org/abs/2212.10559`.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. A survey on in-context learning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1107–1128, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.64. URL `https://aclanthology.org/2024.emnlp-main.64/`.

Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes, 2023. URL `https://arxiv.org/abs/2208.01066`.

T. H., Siegelmann, , , D. E., and Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995.

Yingcong Li, M. Emrullah Ildiz, Dimitris Papailiopoulos, and Samet Oymak. Transformers as algorithms: Generalization and stability in in-context learning, 2023. URL `https://arxiv.org/abs/2301.07067`.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. Metaicl: Learning to learn in context, 2022. URL `https://arxiv.org/abs/2110.15943`.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads, 2022. URL `https://arxiv.org/abs/2209.11895`.

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

Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent, 2023. URL `https://arxiv.org/abs/2212.07677`.

Genta Indra Winata, Andrea Madotto, Zhaojiang Lin, Rosanne Liu, Jason Yosinski, and Pascale Fung. Language models are few-shot multilingual learners, 2021. URL `https://arxiv.org/abs/2109.07684`.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference, 2022. URL `https://arxiv.org/abs/2111.02080`.

Tong Yang, Yu Huang, Yingbin Liang, and Yuejie Chi. In-context learning with representations: Contextual generalization of trained transformers, 2024. URL `https://arxiv.org/abs/2408.10147`.

Tony Z. Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models, 2021. URL `https://arxiv.org/abs/2102.09690`.

Han Zhou, Xingchen Wan, Lev Proleev, Diana Mincu, Jilin Chen, Katherine Heller, and Subhrajit Roy. Batch calibration: Rethinking calibration for in-context learning and prompt engineering, 2024. URL `https://arxiv.org/abs/2309.17249`.

A Technical Appendices and Supplementary Material

Neumann Series The Neumann series is the operator analogue of the geometric series. For a linear operator or matrix A, if its spectral radius satisfies $\rho(A) < 1$, then the inverse of $(I - A)$ can be expanded as

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k.$$

This result is directly analogous to the scalar case: for $|x| < 1$, we have

$$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k.$$

The Neumann series generalizes this idea from scalars to operators. The condition $\rho(A) < 1$ (or equivalently $\|A\| < 1$ in some submultiplicative matrix norm) guarantees convergence, since

$$\lim_{n\to\infty} A^n = 0,$$

which ensures that the infinite sum is well-defined.

Proof (sketch). Consider the partial sum

$$S_m = \sum_{k=0}^{m} A^k.$$

Multiplying by $(I - A)$, we obtain

$$(I - A)S_m = I - A^{m+1}.$$

If $\rho(A) < 1$, then $A^{m+1} \to 0$ as $m \to \infty$. Hence

$$\lim_{m\to\infty} (I - A)S_m = I,$$

which implies

$$(I - A)\Big(\sum_{k=0}^{\infty} A^k\Big) = I,$$

and thus $(I - A)^{-1} = \sum_{k=0}^{\infty} A^k$.

The Neumann series is widely used in analysis and numerical linear algebra. It provides a way to approximate the inverse of $(I - A)$ by truncating the series:

$$(I - A)^{-1} \approx \sum_{k=0}^{m} A^k,$$

with error controlled by $\|A^{m+1}\|$. In practice, this expansion is employed in iterative methods, perturbation theory, and preconditioning, where computing an exact inverse is costly but a series approximation is efficient and interpretable.