In-context Learning on Function Classes Unveiled for Transformers

Zhijie Wang [1] **Bo Jiang** [2] **Shuai Li** [2]

Abstract

Transformer-based neural sequence models exhibit a remarkable ability to perform in-context learning. Given some training examples, a pre-trained model can make accurate predictions on an unseen input. This paper studies why transformers can learn different types of function classes in-context. We first show by construction that there exists a family of transformers (with different activation functions) that implement approximate gradient descent on the parameters of neural networks, and we provide an upper bound for the number of heads, hidden dimensions, and layers of the transformer. We also show that a transformer can learn linear functions, the indicator function of a unit ball, and smooth functions in-context by learning neural networks that approximate them. The above instances mainly focus on a transformer pre-trained on single tasks. We also prove that when pre-trained on two tasks: linear regression and classification, a transformer can make accurate predictions on both tasks simultaneously. Our results move beyond linearity in terms of in-context learning instances and provide a comprehensive understanding of why transformers can learn many types of function classes through the bridge of neural networks.

1. Introduction

In-context learning (ICL) is a phenomenon first observed in natural language processing (NLP) problems where large language models (LLM) like GPT-4 can make accurate predictions based on few prompts without any update on model parameters. People's understanding on in-context learning is still limited. How and why neural sequence models can learn in-context remain a black box. The training and inference stages are two main stages for ICL (Dong et al., 2022). During the training stage, existing ICL studies mainly take a pretrained LLM as a backbone, and optionally warmup the model to strengthen and generalize the ICL ability (Min et al., 2021; Wei et al., 2021; Chung et al., 2024; Chen et al., 2022). Towards the inference stage, empirical works study what an LLM or its cornerstone, the transformer, can infer (Garg et al., 2022; Fu et al., 2024), while theoretical works take different measures to explain the mechanism of ICL inference (Von Oswald et al., 2023; Dai et al., 2022; Akyürek et al., 2022; Xie et al., 2021).

The paper attempts to explain why transformer-based predictors can learn different function classes in-context. We interpret in-context learning of a function as learning implicit neural networks that approximate the function. Currently, there are mainly two understandings of in-context learning: one is based on gradient descent (Von Oswald et al., 2023; Dai et al., 2022; Akyürek et al., 2022), and the other views it as Bayesian inference (Xie et al., 2021). We adopt the former perspective to prove that, when trained properly, transformers can perform approximate gradient descent on the parameters of neural networks without any parameter updates or fine-tuning. These neural networks serve as approximators of different functions.

In Section 3, we prove by construction that a family of transformers, with a wide range of activation functions (not necessarily restricted to the commonly used ReLU), can implement a step of approximate gradient descent on the parameters of neural networks. We start by investigating 2-layer neural networks and then generalize our findings to the n-layer neural network setting. An upper bound for the number of heads, hidden dimension, and the number of layers required for the transformer is provided. Among these, the number of layers is presented in a recursive fashion for the n-layer neural network setting. Moreover, we extend our result to multi-step gradient descent and discover an $l\mu = \text{Const.}$ relationship between the transformer layer l and the gradient descent step μ.

In Section 4, we view neural networks as bridges for transformers to learn function classes in-context and provide an analysis of the resources required for a transformer to approximate the same function class through neural networks of different depths and widths. We showcase that

[1]Shanghai Jiao Tong University, Shanghai, China [2]John Hopcroft Center for Computer Science, Shanghai Jiao Tong University, Shanghai, China. Correspondence to: Shuai Li <shuaili8@sjtu.edu.cn>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

for transformers to learn indicator functions in-context, 2-layer neural networks are not sufficient as bridges because they would cause the number of heads of the transformer to be unacceptably large (exponential in the prediction error). In contrast, deeper and narrower neural networks, which achieve the same approximation accuracy, require fewer resources from the transformer. We also present a condition under which deep networks perform better than shallow ones in terms of approximating smooth functions and requiring a smaller transformer size.

In Section 5, we prove that a transformer can perform algorithm selection: when pre-trained on two tasks, linear regression and classification, a transformer can make accurate predictions on both tasks simultaneously. We demonstrate this by constructing an indicator function following Bai et al. 2024 to discriminate between the two tasks and then perform approximate gradient descent on a neural network approximating both tasks.

We summarize our contributions as follows:

- We construct explicit weights for transformer attention layers and feed-forward layers to perform approximate gradient descent (GD) on mean squared error loss regarding an n-layer neural network. Our results are also suitable to transformers with various activation functions.
- We theoretically prove that transformers can learn function classes including linear functions, indicator functions of unit ball and smooth functions in-context by learning the neural network that approximates them.
- We show that learning 2-layer neural networks is not sufficient for a transformer to perform in-context learning on different function classes, as it would cause the resource requirements (number of heads and hidden layers) to explode exponentially with the prediction error.
- Instead of focusing on single task in-context learning, we also theoretically explain the algorithm selection phenomenon in in-context learning, in which the transformer is pre-trained on both linear regression and classification tasks yet can make accurate predictions given prompts regarding each task.

1.1. Related Work

In-context learning In-context learning has been studied both empirically and theoretically. Garg et al. 2022 empirically show that transformers can learn linear functions, two-layer ReLU neural networks, and decision trees in context. Min et al. 2022 study what aspects of demonstrations impact the performance of in-context learning. As for the theoretical part, Xie et al. 2021 explains ICL as implicit Bayesian inference despite the difference between pretraining and inference distributions, while Akyürek et al. 2022, Von Oswald et al. 2023, and Dai et al. 2022 all interpret in-context learning as transformers performing gradient descent. These works only focus on linear models or their variants without providing an error bound for multiple gradient descent steps. A more recent work (Bai et al., 2024) also investigates gradient descent on more general functions, like 2-layer neural networks, and demonstrates the model selection ability of transformers. We extend their result on 2-layer neural networks to an n-layer neural network setting and also provide a tighter bound on the number of heads and hidden dimension required for the transformer.

Neural networks and approximation theorems The approximation abilities of neural networks have long been studied. Many results have demonstrated the universal approximation property of neural networks in approximating different function classes (Hornik et al., 1989; Hornik, 1991; Barron, 1993). A more recent work (Siegel & Xu, 2020) improved upon Barron 1993 regarding the approximation rate with general activation functions, and their result is adopted in our work. While these universal approximation theorems focus on neural networks with certain depths, more recent work has begun to explore the expressive power of deep neural networks due to their development and success. Yarotsky 2017 and Liang & Srikant 2016 both show the approximation abilities of deep neural networks. Safran & Shamir 2017 illustrates the width-depth trade-offs of neural networks by proving the inapproximability with 2-layer neural networks and the approximability of 3-layer neural networks in terms of approximating indicator functions.

2. Preliminaries

2.1. Transformers

A Transformer layer contains two sub-layers, the attention layer and the MLP layer. We denote the input sequence to the transformer as $\mathbf{H} = [\mathbf{h}_1, \cdots, \mathbf{h}_N] \in \mathbb{R}^{D\times N}$.

Definition 2.1. (Attention layer) An attention layer with M heads is denoted as $\mathrm{Attn}_{\boldsymbol{\theta}}(\cdot)$ where $\boldsymbol{\theta} = \{\mathbf{V}_m, \mathbf{Q}_m, \mathbf{K}_m\}_{m\in[M]}$. The output of this layer on the input matrix $\mathbf{H}$ is:

$$\mathrm{Attn}_{\boldsymbol{\theta}}(\mathbf{H}) = \mathbf{H} + \tfrac{1}{\mathrm{N}} \textstyle\sum_{\mathrm{m}=1}^{\mathrm{M}} (\mathbf{V}_{\mathrm{m}}\mathbf{H}) \times \sigma((\mathbf{Q}_{\mathrm{m}}\mathbf{H})^\top (\mathbf{K}_{\mathrm{m}}\mathbf{H})),$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function . For each column, we denote $\mathbf{h}_i^+ := [\mathrm{Attn}_{\boldsymbol{\theta}}(\mathbf{H})]_{\mathrm{i}}$ and get:

$$\mathbf{h}_i^+ = \mathbf{h}_i + \tfrac{1}{N} \textstyle\sum_{m=1}^{M} \sum_{j=1}^{N} \sigma\left(\langle \mathbf{Q}_m \mathbf{h}_i, \mathbf{K}_m \mathbf{h}_j \rangle\right) \mathbf{V}_m \mathbf{h}_j.$$

The attention layer is followed by the MLP layer.

Definition 2.2. (MLP layer) An MLP layer with hidden dimension D' is denoted as $\mathrm{MLP}_{\boldsymbol{\theta}}(\cdot)$ where $\boldsymbol{\theta} =$

$(\mathbf{W}_1, \mathbf{W}_2) \in \mathbb{R}^{D' \times D} \times \mathbb{R}^{D \times D'}$. The output of this layer on input $\mathbf{H}$ is

$$\mathrm{MLP}_{\boldsymbol{\theta}}(\mathbf{H}) = \mathbf{H} + \mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{H})$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function. For each column:

$$[\mathrm{MLP}_{\boldsymbol{\theta}}(\mathbf{H})]_{\mathrm{i}} = \mathbf{h}_{\mathrm{i}} + \mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{h}_{\mathrm{i}}).$$

Note that the activation function σ in the attention layer and MLP layer applies to each element of the matrix (vector). Next we give a definition on the class of activation functions we focus on.

Definition 2.3. (General decay condition) We call an activation function σ satisfies the general decay condition if $\sigma \in W^{m,\infty}(\mathbb{R})$ is non-zero and there exists a $\nu \in \{\sum_{i=1}^{n} \beta_i\sigma(\omega_i \cdot x + b_i) : \omega_i, b_i, \beta_i \in \mathbb{R}\}$ which satisfies

$$|\nu^{(k)}(t)| \leq C_p(1 + |t|)^{-p}$$

for $0 \leq k \leq m$ and some $p > 1$.

Here $W^{m,\infty}$ denotes the Sobolev space, which is the subset of functions f in $L^{\infty}(\mathbb{R})$ such that f and its weak derivatives up to order m have a finite L^{∞} norm. Most commonly used activation functions, such as Sigmoidal, Arctan, ReLU, Softplus etc., do satisfy the general decay condition as stated in Siegel & Xu 2020.

2.2. Neural Networks

We formulate the mathematical representation of n-layer neural networks as follows:

Definition 2.4. (n-layer neural networks) We denote the output of an n-layer neural network on the input $x \in \mathbb{R}^d$ as

$$\mathrm{pred}_{\mathrm{n}}(\mathbf{w}, \mathbf{x}) = \mathbf{W}^{(\mathrm{n})}(\mathrm{r}(\mathbf{W}^{(\mathrm{n}-1)}(\mathrm{r}(\cdots \mathrm{r}(\mathbf{W}^{(1)}\mathbf{x}))))),$$

where r is an activation function (not necessarily the same as the σ activation in the transformer architecture) and $\mathbf{w} = (\mathbf{W}^{(1)}, \cdots, \mathbf{W}^{(n)})$, $\mathbf{W}^{(i)} \in \mathbb{R}^{K_i \times K_{i-1}}$ for $i = 1, \cdots, n$ with $K_n = 1, K_0 = d$. We denote the k-th row vector of the matrices $\mathbf{W}^{(i)} (i \in [n-1])$ as $\mathbf{v}_{i,k}$, and the k-th element in the vector $\mathbf{W}^{(n)}$ as u_k.

In the n-layer neural network setting above, we omit the bias terms and let the output be a number instead of a vector for simplicity. Also, the activation function r act on each element of the input vector.

2.3. In-context Learning

Here we introduce our in-context learning (ICL) setting. A complete ICL process contains two stages: pre-training and inference.

In the pre-training stage, a transformer is trained on meta-data generated from n different tasks, where each data point $(\mathbf{x}, y)$ is sampled from a distribution $\mathrm{P_i}$, where $i = 1, \cdots, n$.

In the inference stage, the prompts are sampled from a distribution P'_{k} corresponding to task k. Here P'_{k} and the $\mathrm{P_k}$ in pre-training can be different. We denote the prompts as $\mathcal{D} = (\mathbf{x}_i, y_i)_{i \in [N]}$, and a novel input $\mathbf{x}_{N+1}$ is sampled from $\mathrm{P}_{\mathbf{x}}$. So each input is of the form $(\mathcal{D}, \mathbf{x}_{N+1})$. Here $\mathbf{x}_i \in \mathbb{R}^d$.

More specifically, we denote the input to the transformer as

$$\mathbf{H} = \begin{bmatrix} \mathbf{x}_1, & \mathbf{x}_2, & \cdots, & \mathbf{x}_N, & \mathbf{x}_{N+1} \\ y_1, & y_2, & \cdots, & y_N, & 0 \\ \mathbf{p}_1, & \mathbf{p}_2, & \cdots, & \mathbf{p}_N, & \mathbf{p}_{N+1} \end{bmatrix} \in \mathbb{R}^{D \times (N+1)},$$

where $\{\mathbf{p}_i\}_{i \in [N+1]}$ are fixed vectors of the form

$$\mathbf{p}_i = \begin{bmatrix} \mathbf{0}_{D-d-3} \\ 1 \\ 1\{i < N+1\} \end{bmatrix} \in \mathbb{R}^{D-d-1}.$$

A transformer takes the prompt input $\mathbf{H}$ and makes a prediction on the label corresponding to $\mathbf{x}_{N+1}$. The prediction $\hat{y}_{N+1}$ is stored in the output matrix $\hat{\mathbf{H}}$ in the position next to y_N. We say in-context learning succeeds if $\hat{y}_{N+1}$ and y_N is close enough, or ϵ-close, under the metric corresponding to task k.

2.3.1. Algorithm-Selection Mechanism

Most theoretical works focus on explaining the in-context learning ability of transformers pre-trained on a single task, such as linear regression. However, in real life an LLM is pre-trained on a mixture of dataset generated from different tasks. Bai et al. 2024 trained a transformer on both regression and linear classification tasks data and showed that given prompts on linear regression or classification, the prediction ability of the transformer approaches the baseline algorithm on both tasks, as if the transformer automatically chose the best algorithm for each task. This phenomenon is called algorithm-selection. Algorithm-selection naturally fits into our definition of in-context learning by choosing $n = 2$ and 1 represents linear regression, 2 represents classification.

3. Gradient Descent on Multi-layer Neural Networks

In this section, we aim to understand why transformers can learn multi-layer neural networks in-context through the lens of gradient descent. We start by analyzing 2-layer neural networks and then generalize our findings to networks with n layers. To apply gradient descent to these neural networks, we consider the following optimization problem on the loss

function:

$$\min_{\mathbf{w}\in\mathcal{W}} L_N(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N} l(\mathrm{pred}(\mathbf{x}_\mathrm{i};\mathbf{w}),\mathrm{y}_\mathrm{i}).$$

Now we present necessary assumptions to begin our analysis. Following Barron 1993, We define $\mathcal{B}^s$ to be the space of functions $f:\mathbb{R}^d\to\mathbb{R}$ with bounded Barron norm:

$$\|f\|_{\mathcal{B}^s} = \int_{\mathbb{R}^d}(1+|\boldsymbol{\omega}|)^s|\hat{f}(\boldsymbol{\omega})|d\boldsymbol{\omega}.$$

where $\hat{f}$ denotes the Fourier transform of f.

Assumption 3.1. Both the activation function r in neural networks and the loss function l has finite Barron norm.

As is discussed in Barron 1993, $\mathcal{B}^s$ is closed to multiplication, linear combination and translation. Thus, sigmoidal functions, functions with derivatives of sufficiently high order and Boolean functions are all in $\mathcal{B}^s$, so this should include most common activation functions and loss functions in practice. We also point out that Bai et al. 2024 used the assumption that r, l are both C^4 smooth, which is a stricter assumption since C^4 functions do have bounded Barron norm.

During the process of gradient descent, it is likely that the neural networks parameter $\mathbf{w}$ goes out of its domain $\mathcal{W}$, so we need the following assumption to project $\mathbf{w}$ onto $\mathcal{W}$.

Assumption 3.2. $\mathcal{W}$ as the domain of $\mathbf{w}$ is compact and there exists some MLP layer parameter such that the MLP layer projects $\mathbf{w}$ into $\mathcal{W}$.

This assumption states that the MLP layer truncates $\mathbf{w}$ element-wise according to $\mathcal{W}$. Such an MLP layer always exists in the sense that we can choose proper $\mathbf{W}_1, \mathbf{W}_2$ to fulfill the truncation task.

3.1. Gradient Descent on 2-layer Neural Networks

A 2-layer neural network can be written as

$$\mathrm{pred}_2(\mathbf{w},\mathbf{x}) = \mathbf{W}^{(2)}(\mathrm{r}(\mathbf{W}^{(1)}\mathbf{x})),$$

where $\mathbf{x}\in\mathbb{R}^d, \mathbf{W}^{(1)}\in\mathbb{R}^{K_1\times d}, \mathbf{W}^{(2)}\in\mathbb{R}^{1\times K_1}$.

We want to show that transformers can implement gradient descent on 2-layer neural networks in-context without any parameter update.

We note that

$$\nabla_{\mathbf{w}} L_N(\mathbf{w}) = \tfrac{1}{N}\textstyle\sum_{i=1}^{N}\partial_1 l(\mathrm{pred}(\mathbf{x}_\mathrm{i};\mathbf{w}),\mathrm{y}_\mathrm{i})\cdot\nabla_{\mathbf{w}}\mathrm{pred}(\mathbf{x}_\mathrm{i};\mathbf{w}),$$

where $\partial_1 l$ is the partial derivative of l with respect to the first component. Furthermore,

$$\nabla_{\mathbf{w}}\mathrm{pred}(\mathbf{x}_\mathrm{i};\mathbf{w}) = \begin{bmatrix} u_1\cdot r'(\langle\mathbf{v}_1,\mathbf{x}_i\rangle)\cdot\mathbf{x}_i \\ r(\langle\mathbf{v}_1,\mathbf{x}_i\rangle) \\ \vdots \\ u_K\cdot r'(\langle\mathbf{v}_K,\mathbf{x}_i\rangle)\cdot\mathbf{x}_i \\ r(\langle\mathbf{v}_k,\mathbf{x}_i\rangle)\end{bmatrix}\in\mathbb{R}^{\dim(\mathbf{w})}.$$

We show below that a 2-layer transformer can compute one step of approximate gradient descent, following the intuition of Siegel & Xu 2020: The first self attention sub-layer computes and stores approximate $\mathrm{pred}(\mathbf{x}_\mathrm{i};\mathbf{w})$ in hidden space, and the MLP sub-layer computes and stores $\partial_1 l$, while the second attention sub-layer computes and stores $\mathbf{w}-\eta\nabla L_N(\mathbf{w})$, and the last MLP sub-layer maps this result of one step gradient descent to the domain of $\mathbf{w}$. In order for the attention layer and MLP layer to compute $\nabla_{\mathbf{w}} L_N(\mathbf{w})$, we need the following neural network approximation lemma to approximately compute $\mathrm{pred}(\mathbf{x}_\mathrm{i};\mathbf{w}), \partial_1\mathrm{l}$ and $r'(t)$. We denote $\Sigma_d^n(\sigma) = \{\sum_{i=1}^n \beta_i\sigma(\boldsymbol{\omega}_i\cdot\mathbf{x}+b_i) : \omega_i\in\mathbb{R}^d, b_i,\beta_i\in\mathbb{R}\}$ in the following lemma.

Lemma 3.3 (NN approximation)**.** *Let $\Omega\subset\mathbb{R}^d$ be a bounded domain. If the activation function $\sigma\in W^{m,\infty}(\mathbb{R})$ is non-zero and there exists a $\nu\in\Sigma_1^{n_0}(\sigma)$ which satisfies the polynomial decay condition*

$$|\nu^{(k)}(t)|\le C_p(1+|t|)^{-p}$$

for $0\le k\le m$ and some $p>1$, we have

$$\inf_{f_n\in\Sigma_d^n(\sigma)}\|f-f_n\|_{H^m(\Omega)} \le |\Omega|^{\frac{1}{2}}\sqrt{n_0}C(p,m,dim(\Omega),\sigma)n^{-\frac{1}{2}}\|f\|_{\mathcal{B}^{m+1}}$$

for any $f\in\mathcal{B}^{m+1}$.

Here $H^m(\Omega) = W^{m,2}(\Omega)$ is a Sobolev space, which is the subset of functions f in $L^2(\Omega)$ such that f and its weak derivatives up to order m have finite L^2 norm . In this lemma we assume the activation function satisfies the general decay condition in Definition 2.3. Note that when $m=0$, the Sobolev space H^0 is in effect the L^2 space, and we only consider $m=0$ in the following discussion. Setting the right hand side to $\mathcal{O}(\epsilon)$ yields $n=\epsilon^{-2}$, and we can interpret the lemma into: let $n=\epsilon^{-2}$, for any $f\in\mathcal{B}^1$, there exists an $f_n\in\Sigma_d^n(\sigma)$ such that

$$\|f-f_n\|_{L^2(\Omega)}\le\mathcal{O}(\epsilon).$$

Now we are ready to provide our theorem on in-context learning of 2-layer neural networks.

Theorem 1 (ICGD on 2-layer NNs)**.** *Under Assumption 3.1 and Assumption 3.2, there exists a family of 2-layer transformers (with activation functions satisfying the general decay condition in Definition 2.3) such that for any input*

data $(\mathcal{D}, \mathbf{x}_{N+1})$ *and any* $\mathbf{w}$*, a transformer performs approximate gradient descent on the neural networks parameter* $\mathbf{w}$*:*

$$\mathbf{w}_\eta^+ = \mathrm{Proj}_{\mathcal{W}}\left(\mathbf{w} - \eta(\nabla \mathrm{L_N}(\mathbf{w}) + \epsilon(\mathbf{w}))\right),\ \|\epsilon(\mathbf{w})\|_2 \le \eta\epsilon.$$

Furthermore, the upper bound for number of heads and hidden dimension for the transformer is:

$$\max_{l\in[2]} M^{(l)} \le \mathcal{O}(\epsilon^{-2}), \max_{l\in[2]} D^{(l)} \le \mathcal{O}(\epsilon^{-2}).$$

Remark 3.4. We note that in the theorem when we say a transformer performs approximate gradient descent on the neural networks parameter, we mean that the transformer maps each column of the input matrix $\mathbf{H}$: $\mathbf{h}_i = [\mathbf{x}_i; y_i'; \mathbf{w}; \mathbf{0}; 1; t_i]$ to the output vector $\mathbf{h}_i' = [\mathbf{x}_i; y_i'; \mathbf{w}_\eta^+; \mathbf{0}; 1; t_i]$, where only the parameter $\mathbf{w}$ is updated according to gradient descent and the other parts remain unchanged.

This theorem improves upon the result in Bai et al. 2024 in two ways: first we relieve the requirement for the transformer activation function to be ReLU and allow a wide range of activation functions, second we achieve a tighter upper bound on the hidden dimension and number of heads compared with their $\mathcal{O}(\epsilon^{-2}\log(1/\epsilon))$ result.

3.2. Gradient Descent on n-layer Neural Networks

Now we tackle the n-layer neural networks setting. As in Definition 2.4, n-layer neural networks can be formulated as

$$\mathrm{pred}_n(\mathbf{w}, \mathbf{x}) = \mathbf{W}^{(n)}(r(\mathbf{W}^{(n-1)}(r(\cdots r(\mathbf{W}^{(1)}\mathbf{x}))))).$$

We theoretically prove that transformers can implement gradient descent on n-layer neural networks in context. Additionally, we provide an upper bound for the number of heads and the hidden dimension of transformers. We also present a recurrence relation for the number of transformer layers.

Theorem 2 (ICGD on n-layer NNs)**.** *Under Assumption 3.1 and Assumption 3.2, there exists a family of* a_n*-layer transformers (with activation functions satisfying the general decay condition in Definition 2.3) such that for any input data* $(\mathcal{D}, \mathbf{x}_{N+1})$ *and any* $\mathbf{w}$*, a transformer performs approximate gradient descent on the* n*-layer neural networks parameter* $\mathbf{w}$*:*

$$\mathbf{w}_\eta^+ = \mathrm{Proj}_{\mathcal{W}}\left(\mathbf{w} - \eta(\nabla \mathrm{L_N}(\mathbf{w}) + \epsilon(\mathbf{w}))\right),\ \|\epsilon(\mathbf{w})\|_2 \le \eta\epsilon,$$

where a_n *satisfies* $\mathcal{O}(a_n) = \mathcal{O}(n) + \mathcal{O}(a_{n-1})$*. Furthermore, the upper bound for number of heads and hidden dimension for the transformer is :*

$$\max_{l\in[a_n]} M^{(l)} \le \mathcal{O}(nK^2\epsilon^{-2}), \max_{l\in[a_n]} D^{(l)} \le \mathcal{O}(nK^2\epsilon^{-2}),$$

where K *denotes the maximum width of the neural networks:* $K = \max\{K_0, K_1, \cdots, K_n\}$.

Remark 3.5. We note that a_n, the number of transformer layers required in the above theorem is of order $\mathcal{O}(n^2)$, which is a decent growth rate considering the fast growth of neural networks neurons as its depth increases.

Remark 3.6. The result in Section 3.1 is of course a special case of Theorem 2, but We provide Theorem 1 separately for two reasons: first it is the most simple and empirically verified case(Garg et al., 2022), second it is the initial condition for recursion of the general case in terms of transformer layers.

Trade-off of width and depth of neural networks People are interested in the approximation capabilities of neural networks, and many studies have discussed the how width and depth mutually affect the approximation ability of neural networks. Thus it's worthwhile discussing how the trade-off between width and depth of neural networks can affect the error of the approximate gradient descent step. If we control the error in the approximate gradient descent to be $\eta\epsilon$, then in order for the transformer to learn two different neural networks with the same magnitude of number of heads, we require nK^2 to be a constant. Thus controlling the width of the network is more efficient than controlling the depth of the neural networks in terms of maintaining the same approximation error, indicating that deep and narrow neural networks may perform better than shallow and wide neural networks in terms of saving computational resource.

The a_n layer transformer in Theorem 2 can be stacked l times to perform l steps of gradient descent. Thus as a corollary of Theorem 2, we provide the result for multi-step approximate gradient descent. We say a function f is strongly convex with modulus m if

$$mI \preceq \nabla^2 f.$$

Now we have the following corollary.

Corollary 3.7 (multi-step ICGD on n-layer NNs)**.** *For* $l \ge 1$*, suppose* $\nabla L_N(\mathbf{w})$ *is* M*-Lipschitz on* $\mathcal{W}$ *and* $L_N(\mathbf{w})$ *is strongly convex with modulus* m*. Then under Assumption 3.1 and Assumption 3.2, the* (la_n)*-layer transformers in Theorem 2 approximates gradient descent and the output satisfies:*

$$\|\hat{\mathbf{w}}^l - \mathbf{w}_{\mathrm{True}}\|_2 \le (M^{-1}(1+\eta M)^l + (1-\eta\frac{2Mm}{M+m})^{\frac{l}{2}})\epsilon,$$

where $\hat{\mathbf{w}}^l$ *is the output of the transformer,* $\mathbf{w}_{\mathrm{True}}$ *is the true value of the neural networks parameter.*

In the above corollary, we denote $f(\eta) = M^{-1}(1+\eta M)^l + (1-\eta\frac{2Mm}{M+m})^{\frac{l}{2}}$ and solve the optimization problem over η. Letting $f'(\eta) = 0$ yields

$$(1+\eta M)^{l-1} = \frac{Mm}{M+m}(1-\eta\frac{2Mm}{M+m})^{\frac{l}{2}-1}.$$

Notice that η is relatively small, so we use Taylor's expansion up to order one and get:

$$[(l-1)M + (\frac{Mm}{M+m})^2(l-2)]\eta = \frac{Mm}{M+m} - 1.$$

For l large enough, $l \approx l-1 \approx l-2$. This shows when $l\eta = \text{Const.}$ we achieve a minimum upper bound for $\|\hat{\mathbf{w}}^l - \mathbf{w}_{\text{True}}\|_2$. Although this is mathematically interesting, we should remind readers that the gradient descent step η is learned by the transformer during the pre-training process, not manually set by users. The result may indicate that given an l layer transformer, it tends to learn gradient descent step of order $\mathcal{O}(\frac{1}{l})$. Moreover, it seems that the first term in $f(\eta)$ will go to infinity if $l \to \infty$. But if we choose $\eta = \frac{1}{l}$ according to the above analysis, then the term $(1+\eta M)^l \to e^M$ as $l \to \infty$, which is a constant, showing that the error will not explode as l increases.

4. Transformer Learn Function Classes In-context

Since neural networks are universal approximators, and transformers can learn neural networks in context, a natural question is whether transformers can learn function classes that neural networks can approximate in context. Theorem 2 bridges the gap between transformers and neural networks, and previous work on approximation theorems of neural networks has bridged the gap between neural networks and function classes, so it is natural to consider the whole path of transformers learning function classes in-context. But how much resource does it cost for a transformer (number of layers of the transformer and number of heads for each attention layer, or equivalently number of parameter matrices) to approximate a neural network (as an approximator)? We consider three types of function classes: indicator functions of L_2 balls corresponding to classification problems, linear functions corresponding to linear regression problems and smooth functions. In all these instances, the transformer is pre-trained on different distributions corresponding to a single task, and the prompts are generated from a distribution corresponding to the task in pre-training. For example, we set the task as linear regression, and $y = \langle \mathbf{w}, \mathbf{x} \rangle$. In the pre-training part $\mathbf{w}$ is generated from different distributions, and in the post-training part a new $\mathbf{w}_{\text{prompt}}$ is generated from a distribution to form the prompts.

4.1. Indicator Functions of L_2 Balls

$f(\mathbf{x}) = \mathbf{1}(\|A\mathbf{x} + \mathbf{b}\| \leq r)$ is the indicator function of unit ball. This indication function naturally induces a classification problem.

Safran & Shamir 2017 proves the inapproximability of 2-layer neural networks and the approximability of 3-layer neural networks. We present them as lemmas below.

Lemma 4.1. *The following holds for some positive univeral constants c_1, c_2, c_3, c_4, and any network employing an activation functioin satisfying Assumptions 1 and 2 in Eldan & Shamir 2016: For any $d > c_1$, and any non-singular matrix $A \in \mathbb{R}^{d\times d}, \mathbf{b} \in \mathbb{R}^d$ and $r \in (0, \infty)$, there exists a continuous probability distribution γ on $\mathbb{R}^d$, such that for any function g computed by a 2-layer network of width at most $c_2 \exp(c_4 d)$, and for the function $f(\mathbf{x}) = \mathbf{1}(\|A\mathbf{x}+\mathbf{b}\| \leq r)$, we have*

$$\int_{\mathbb{R}^d} (f(\mathbf{x}) - g(\mathbf{x}))^2 \cdot \gamma(\mathbf{x}) d\mathbf{x} \geq \frac{c_2}{d^4}.$$

Lemma 4.2. *Given $\delta > 0$, for any activation function σ satisfying Assumption 1 in Eldan & Shamir 2016 and any continuous probability distribution μ on $\mathbb{R}^d$, there exists a constant c_σ dependent only on σ, and a function g expressible by a 3-layer network of width at most $\max\{8c_\sigma d^2/\sigma, c_\sigma\sqrt{1/2\delta}\}$, such that the following holds:*

$$\int_{\mathbb{R}^d} (g(\mathbf{x}) - \mathbf{1}(\|\mathbf{x}\|_2 \leq 1))^2 \mu(\mathbf{x}) d\mathbf{x} \leq \delta,$$

where c_σ is a constant depending solely on σ.

These two lemmas reveal that the indicator of the L^2 ball can be better approximated by a 3-layer neural network with width $\mathcal{O}(d^2)$ than a 2-layer neural network requiring width at least exponential in the input dimension.

For a transformer to learn the indicator of unit ball in-context, we show below that it learns a three layer neural network that approximates the target function.

Theorem 3. *For any given $\epsilon > 0$, let $f(\mathbf{x}) = \mathbf{1}(\|A\mathbf{x} + \mathbf{b}\| \leq r)$ be the indicator of unit ball. There exists a cL-layer transformer with*

$$\max_{l\in[cL]} M^{(l)} \leq \mathcal{O}(\frac{1}{\delta\epsilon^2}), \max_{l\in[cL]} D^{(l)} \leq \mathcal{O}(\frac{1}{\delta\epsilon^2}),$$

where c is a constant, such that it performs approximate gradient descent on a 3-layer, $\mathcal{O}(\delta^{-1/2})$-wide neural network which δ-approximates $f(\mathbf{x})$: the l-th layer's output is $\mathbf{h}_i^{(cl)} = [\mathbf{x}_i; y_i'; \hat{\mathbf{w}}^l; \mathbf{0}; 1; t_i]$ for $i \in [N+1]$, and

$$\hat{\mathbf{w}}^{(l)} = \text{Proj}_{\mathcal{W}}(\hat{\mathbf{w}}^{(l-1)} - \eta(\nabla \text{L}_\text{N}(\hat{\mathbf{w}}^{(l-1)}) + \epsilon(\hat{\mathbf{w}}^{(l-1)}))),$$

where $\|\epsilon(\mathbf{w})\|_2 \leq \epsilon$. Moreover, the prediction of the transformer $\hat{y}_{N+1}$ satisfies

$$|\hat{y}_{N+1} - y_{N+1}| \leq \mathcal{O}(\epsilon + \delta).$$

Remark 4.3. Both the neural networks in Lemma 4.1 and Lemma 4.2 can δ-approximate the target function $f(\mathbf{x})$, but if we substitute the 3-layer neural network in Theorem 3 with the 2-layer neural network in Lemma 4.1, then the transformer needs to learn an at least $\mathcal{O}(\exp(\delta^{-1/4}))$-wide

neural network, which will cause the upper bound for the number of heads and hidden dimension of the transformer to be $\mathcal{O}(\exp(\delta^{-1/4})\epsilon^{-2})$.

The observation underscores the potential for an exponential increase in the size of transformers when a 2-layer neural network is trained to approximate the indicator function. Interestingly, the need for a significantly smaller transformer size is evident when employing a 3-layer neural network. This highlights the importance of considering deeper neural networks, showcasing their relevance in addressing computational efficiency and resource requirements.

4.2. Linear Functions

A linear function corresponding to the linear regression problem is $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$. Notice that for ReLU activation function σ, we have $f(\mathbf{x}) = \dfrac{\sigma(\mathbf{w}^\top \mathbf{x}) - \sigma(-\mathbf{w}^\top \mathbf{x})}{2}$. This is a 2-layer neural network, so we can directly apply Theorem 1 and get:

Theorem 4. *Let $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$ be a linear function. There exists a $2L$-layer transformer with*

$$\max_{l \in [2L]} M^{(l)} \leq \mathcal{O}(\epsilon^{-2}), \max_{l \in [2L]} D^{(l)} \leq \mathcal{O}(\epsilon^{-2}),$$

such that it performs approximate gradient descent on a 2-layer, width 2 neural network which equals $f(\mathbf{x})$: the l-th layer's output is $\mathbf{h}_i^{(2l)} = [\mathbf{x}_i; y_i'; \hat{\mathbf{w}}^l; \mathbf{0}; 1; t_i]$ for $i \in [N+1]$, and

$$\hat{\mathbf{w}}^{(l)} = \mathrm{Proj}_{\mathcal{W}}(\hat{\mathbf{w}}^{(l-1)} - \eta(\nabla \mathrm{L_N}(\hat{\mathbf{w}}^{(l-1)}) + \epsilon(\hat{\mathbf{w}}^{(l-1)}))),$$

where $\|\epsilon(\mathbf{w})\|_2 \leq \epsilon$. Moreover, the prediction of the transformer $\hat{y}_{N+1}$ satisfies

$$|\hat{y}_{N+1} - y_{N+1}| \leq \mathcal{O}(\epsilon).$$

This theorem is equivalent to Theorem 1 in essence since $f(\mathbf{x})$ can be exactly represented by the 2-layer neural network. So unlike the case in the classification problem, there is no neural network approximation involved, and it is equivalent for the transformer to learn the linear function or learn the 2-layer neural network in-context.

4.3. Smooth Functions

We use the approximation results of deep neural networks achieved by Yarotsky 2017. We denote by $F_{d,n}$ the unit ball in $\mathcal{W}^{n,\infty}([0,1]^d)$:

$$F_{d,n} = \{f \in \mathcal{W}^{n,\infty}([0,1]^d) : \|f\|_{\mathcal{W}^{n,\infty}([0,1]^d)} \leq 1\}.$$

Lemma 4.4. *For any d, n and $\epsilon \in (0,1)$, there is a ReLU network architecture that*

- *is capable of expressing any function from $F_{d,n}$ with error ϵ;*
- *has the depth at most $c(\ln(1/\epsilon) + 1)$ and at most $c\epsilon^{-d/n}(\ln(1/\epsilon)+1)$ computation units, with some constant $c = c(d,n)$.*

Lemma 4.5. *Let $f \in C^2([0,1]^d)$ be a nonlinear function. Then, for any fixed L, a depth-L ReLU network approximating with error $\epsilon \in (0,1)$ must have at least $c\epsilon^{-1/(2(L-1))}$ computation units, with some constant $c = c(f,L) > 0$.*

Below we state a formal theorem for the resource of a transformer it takes to learn the smooth nonlinear function.

Theorem 5. *For any d and $n > 2$, let $f \in \mathcal{W}^{n,\infty}([0,1]^d)$. Choose any $\epsilon > 0$, there exists a $\mathcal{O}(\ln^2(1/\delta)L)$-layer ($k_\delta L$-layer) transformer with*

$$\max_{l \in [k_\delta L]} M^{(l)} \leq \mathcal{O}(\frac{1}{\delta^{2d/n}\epsilon^2}), \max_{l \in [k_\delta L]} D^{(l)} \leq \mathcal{O}(\frac{1}{\delta^{2d/n}\epsilon^2})$$

such that it performs approximate gradient descent on a $c(\ln(1/\delta)+1)$-layer, $\delta^{-d/n}$-wide neural network which δ-approximates $f(\mathbf{x})$: the l-th layer's output is $\mathbf{h}_i^{(2l)} = [\mathbf{x}_i; y_i'; \hat{\mathbf{w}}^l; \mathbf{0}; 1; t_i]$ for $i \in [N+1]$, and

$$\hat{\mathbf{w}}^{(l)} = \mathrm{Proj}_{\mathcal{W}}(\hat{\mathbf{w}}^{(l-1)} - \eta(\nabla \mathrm{L_N}(\hat{\mathbf{w}}^{(l-1)}) + \epsilon(\hat{\mathbf{w}}^{(l-1)}))),$$

where $\|\epsilon(\mathbf{w})\|_2 \leq \epsilon$. Moreover, the prediction of the transformer $\hat{y}_{N+1}$ satisfies

$$|\hat{y}_{N+1} - y_{N+1}| \leq \mathcal{O}(\epsilon + \delta).$$

Remark 4.6. Linear function is a special case of the smooth function discussed here, but the result on linear function is more 'accurate' in the sense that the upper bound for number of heads, hidden layer and the prediction error is tighter than the result in smooth function, and we want to separate them because linear function is an especially important function class in the realm of supervised learning.

Remark 4.7. Both the neural networks in Lemma 4.4 and Lemma 4.5 can δ-approximate the target function $f(\mathbf{x})$, but if we substitute the deep neural network in Theorem 5 with the shallow network in Lemma 4.5, then the transformer needs to learn an at least $\mathcal{O}(\delta^{-1/2(L-1)})$-wide neural network, and the upper bound for the number of heads and hidden dimension of the transformer will become $\mathcal{O}(\delta^{-1/(L-1)})\epsilon^{-2})$. If $2(L-1)d < n$, then it costs a transformer less resource to learn the deep network than the shallow network in order to do in context learning on $f(\mathbf{x})$.

Intuitively, the smoother the function is and the lower dimension the input $\mathbf{x}$ is, the better deep neural networks perform in terms of approximation and transformer resource cost.

5. Algorithm Selection

Recall our definition of in-context learning in Section 2, ICL is not limited to transformers pretrained on single task dataset. Bai et al. 2024 found that a transformer trained on both linear regression and classification data, when given prompts on linear regression or classification during the inference stage, can approach the baseline algorithm on both tasks. This phenomenon is called algorithm selection. Now we abstract it as follows: Given data $\mathbf{x} \sim P_x, \mathbf{w} \in P_w$, the corresponding y is generated as $y = \langle \mathbf{w}, \mathbf{x} \rangle$ for regression tasks and $y = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)$ for classification task. A transformer pretrained on the meta data generated above can approach task-specific supervised learning algorithm (least squares for linear regression and logistic regression for classification) on both tasks.

A pre-trained transformer should automatically categorize the prompt into two classes: linear regression and classification, by utilizing an indicator function I_{cls}. Then the transformer only needs to learn the function $I_{\text{cls}} \cdot \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle) + (1 - I_{\text{cls}}) \cdot \langle \mathbf{w}, \mathbf{x} \rangle$. Now we first construct the indicator function and prove that it can be realized by a single attention layer following Bai et al. 2024.

Lemma 5.1 (Indicator function)**.** *An attention layer with 6 heads can implement the indicator function I_{cls} of the form*

$$I_{cls}(\mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} I(y_i),$$

where

$$I(y) = \begin{cases} 1 & y \in \{0, 1\} \\ 0 & y \in [-\epsilon, \epsilon] \cup [1 - \epsilon, 1 + \epsilon] \\ \textit{linear interpolation} & \textit{otherwise} \end{cases}$$

.

When the ys in the prompt are all $0, 1$s, we can classify the problem into a classification problem, otherwise it can be classified into a linear regression problem. The ϵ in the indicator function I is to ensure its continuity, and we can always choose ϵ small enough to ensure the classification accuracy.

Now we can provide the full theorem for algorithm selection.

Theorem 6. *Given $\epsilon > 0, \delta > 0$, there exists a $(c+2)L+1$-layer transformer (the constant c is stated in Theorem 3) with*

$$\max_{l \in [(c+2)L]} M^{(l)} \leq \mathcal{O}(\frac{1}{\delta \epsilon^2}), \max_{l \in [(c+2)L]} D^{(l)} \leq \mathcal{O}(\frac{1}{\delta \epsilon^2})$$

that can perform In-context algorithm selection on linear regression and classification tasks, where its output satisfies

$$|\hat{y} - y_{N+1}| \leq \mathcal{O}(\epsilon + \delta).$$

Remark 5.2. Algorithm selection is in fact part of in-context learning according to our definition of ICL in Section 2. This phenomenon doesn't have to be limited to 2 training tasks as in our theorem. A deeper implication of our theorem is its potential to generalize from the 2-algorithm selection setting to an n-algorithm selection setting. For n different tasks, a transformer simply needs to learn n indication functions I_i for each task i, and the final prediction in the inference stage is $\sum_{i=1}^{n} I_i \cdot f_i$, where each f_i is the neural network which approximates the function corresponding to task i that transformer learned.

We posit that the essence of In-Context Learning (ICL) lies in algorithm selection. What distinguishes ICL from supervised learning is its unique capability to provide predictions without necessitating any prior knowledge about the input data. In ICL, the system autonomously determines the most suitable algorithm based on the contextual information available, offering a versatile and adaptive approach to prediction tasks.

6. Conclusion

We provide results on explaining transformers learning n-layer neural networks through approximate gradient descent and view in-context learning of a function as the process of learning neural networks which approximate this function. We also provide a theoretical guarantee for transformers to implement in-context algorithm selection.

Our work shed some light on a comprehensive understanding of in-context learning. In fact, our results suggest that transformers can in-context learn any function classes that can be approximated by neural networks. We also present the resource required for a transformer to learn indicator functions and smooth functions (in particular C^n functions), showing 2-layer neural networks aren't sufficient for in-context learning due to the explosion of transformer size. Also, the smoother the function is and the lower dimension the input is the better deep neural networks perform in terms of transformer resource cost. We believe our work brings a new perspective to the understanding of in-context learning and opens up new directions for empirical exploration of in-context learning.

Acknowledgements

The corresponding author Shuai Li is supported by National Natural Science Foundation of China (92270201).

Impact Statement

The ethical aspects of in-context learning lies in the training of LLM itself. An LLM may need access to a wide range of contextual information, raising concerns about user

privacy and data protection. The collection and utilization of personal data in the learning process should adhere to ethical standards and legal regulations. The future societal consequences may be the impact on employment patterns as certain tasks become automated. Job displacement and the need for new skills to navigate a changing job market are potential societal consequences.

References

Akyürek, E., Schuurmans, D., Andreas, J., Ma, T., and Zhou, D. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*, 2022.

Bai, Y., Chen, F., Wang, H., Xiong, C., and Mei, S. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *Advances in neural information processing systems*, 36, 2024.

Barron, A. R. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

Chen, M., Du, J., Pasunuru, R., Mihaylov, T., Iyer, S., Stoyanov, V., and Kozareva, Z. Improving in-context few-shot learning via self-supervised training. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3558–3573, 2022.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

Dai, D., Sun, Y., Dong, L., Hao, Y., Sui, Z., and Wei, F. Why can gpt learn in-context? language models secretly perform gradient descent as meta optimizers. *arXiv preprint arXiv:2212.10559*, 2022.

Dong, Q., Li, L., Dai, D., Zheng, C., Wu, Z., Chang, B., Sun, X., Xu, J., and Sui, Z. A survey for in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

Eldan, R. and Shamir, O. The power of depth for feedforward neural networks. In *Conference on learning theory*, pp. 907–940. PMLR, 2016.

Fu, H., Guo, T., Bai, Y., and Mei, S. What can a single attention layer learn? a study through the random features lens. *Advances in Neural Information Processing Systems*, 36, 2024.

Garg, S., Tsipras, D., Liang, P. S., and Valiant, G. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Liang, S. and Srikant, R. Why deep neural networks for function approximation? *arXiv preprint arXiv:1610.04161*, 2016.

Min, S., Lewis, M., Zettlemoyer, L., and Hajishirzi, H. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943*, 2021.

Min, S., Lyu, X., Holtzman, A., Artetxe, M., Lewis, M., Hajishirzi, H., and Zettlemoyer, L. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.

Safran, I. and Shamir, O. Depth-width tradeoffs in approximating natural functions with neural networks. In *International conference on machine learning*, pp. 2979–2987. PMLR, 2017.

Siegel, J. W. and Xu, J. Approximation rates for neural networks with general activation functions. *Neural Networks*, 128:313–321, 2020.

Von Oswald, J., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A., and Vladymyrov, M. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pp. 35151–35174. PMLR, 2023.

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Xie, S. M., Raghunathan, A., Liang, P., and Ma, T. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*, 2021.

Yarotsky, D. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.

A. Experiment

In Figure 1 and Figure 2, we show that a pretrained transformer can learn a quadratic function and a 3-layer neural network without any parameter update. In the pretraining stage we set the learning rate of stochastic gradient descent to 3×10^{-4} and train for 100000 steps with batch size 32. In the inference stage we generate prompts $(\mathbf{x}, y)$s according to the corresponding function (quadratic function and 3 layer neural network in our case) with $\mathbf{x}$s generated from a standard Gaussian distribution.

Model architecture Our experiment setup follows Garg et al. 2022. We train a transformer model (GPT-2 structure) with 12 layers, 8 attention heads and 256 hidden dimensions. Each transformer layer contains an attention layer and an MLP layer as in Definition 2.1 and Definition 2.2.

Quadratic regression (Figure 1). We consider a d dimensional quadratic regression task with in-context examples of the form $\mathbf{z} = (\mathbf{x}, y) \in \mathbb{R}^d \times \mathbb{R}$, where $\mathbf{x}$s are sampled i.i.d. from standard Gaussian distribution, and $y = \mathbf{w}^\top \mathbf{x}^{\odot 2}$, where $\mathbf{w} \sim \mathcal{N}(0, \mathbf{\Sigma})$. We set dimension $d = 20$ in Figure 1.

3 layer neural network (Figure 2). We consider a 3 layer neural network task with in-context examples of the form $\mathbf{z} = (\mathbf{x}, y) \in \mathbb{R}^d \times \mathbb{R}$, where $\mathbf{x}$s are sampled i.i.d. from standard Gaussian distribution, and $y = \mathbf{W}^{(3)}(r(\mathbf{W}^{(2)}((r(\mathbf{W}^{(1)}\mathbf{x})))))$. as in Definition 2.4, where $\mathbf{W}_{ij} \sim \mathcal{N}(0, \Sigma)$, and we set the activation function r to be ReLU activation. We set the width of each hidden layer to be $K = 50$, and the input dimension $d = 20$.

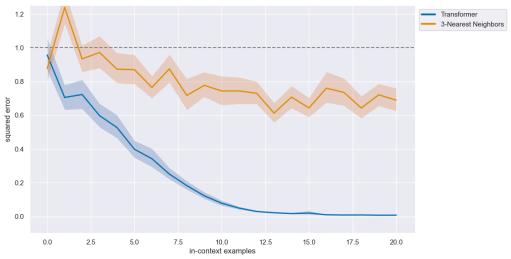

Figure 1. quadratic function

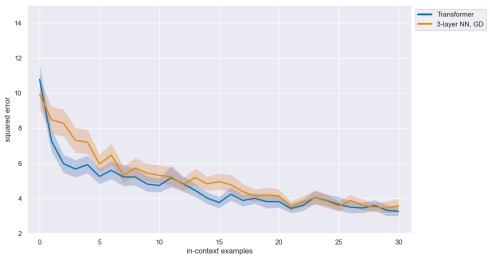

Figure 2. 3 layer neural network

Experimental results Our results on the in-context learning ability of transformer is shown in Figure 1 and Figure 2. The transformer learns quadratic function well as we can see from Figure 1 that the test loss goes to 0 as in-context example increases and 3-Nearest Neighbors can't solve quadratic regression. The transformer can also learn the 3-layer neural network, where the test loss curve matches the gradient descent baseline, supporting our theoretical results.

B. In-context Learning of Neural Networks

We only provide the proof of Theorem 2 here because Theorem 1 is a special case of Theorem 2 thus we only provide the proof of the more genral case. We provide Theorem 1 separately for two reasons: first it is the most simple and empirically verified case (transformers can learn 2-layer neural networks in-context), second it is the initial condition for recursion of the general case in terms of transformer layers.

First we need an approximation theorem of neural networks (Siegel & Xu 2020 corollary 1). We denote $\Sigma_d^n(\sigma) = \{\sum_{i=1}^n \beta_i \sigma(\boldsymbol{\omega}_i \cdot \mathbf{x} + b_i) : \omega_i \in \mathbb{R}^d, b_i, \beta_i \in \mathbb{R}\}$. We also define $\mathcal{B}_s$ to be the space of functions $f : \mathbb{R}^d \to \mathbb{R}$ with bounded Barron norm

$$\|f\|_{\mathcal{B}^s} = \int_{\mathbb{R}^d} (1 + |\boldsymbol{\omega}|)^s |\hat{f}(\boldsymbol{\omega})| d\boldsymbol{\omega}.$$

Lemma 3.3 (NN approximation)**.** *Let $\Omega \subset \mathbb{R}^d$ be a bounded domain. If the activation function $\sigma \in W^{m,\infty}(\mathbb{R})$ is non-zero and there exists a $\nu \in \Sigma_1^{n_0}(\sigma)$ which satisfies the polynomial decay condition*

$$|\nu^{(k)}(t)| \leq C_p(1 + |t|)^{-p}$$

for $0 \leq k \leq m$ *and some* $p > 1$*, we have*

$$\begin{aligned}&\inf_{f_n \in \Sigma_d^n(\sigma)} \|f - f_n\|_{H^m(\Omega)} \\ &\leq |\Omega|^{\frac{1}{2}} \sqrt{n_0} C(p, m, \textit{dim}(\Omega), \sigma) n^{-\frac{1}{2}} \|f\|_{\mathcal{B}^{m+1}}\end{aligned}$$

for any $f \in \mathcal{B}^{m+1}$.

Proof. We first observe that $\nu \in \Sigma_1^{n_0}(\sigma)$ implies that

$$\Sigma_d^n(\nu) \subset \Sigma_d^{nn_0}(\sigma).$$

So we only need to prove that the result without the $\sqrt{n_0}$ term holds for σ satisfying the polynomial decay condition itself. The decay condition implies that $\sigma \in L^1(\mathbb{R})$ and thus the Fourier transform of σ is well-defined. Since $\sigma \neq 0$, we have

$$0 \neq \hat{\sigma}(a) = \frac{1}{2\pi} \int_{\mathbb{R}} \sigma(\boldsymbol{\omega} \cdot \mathbf{x} + b) e^{-ia(\boldsymbol{\omega} \cdot \mathbf{x} + b)} db,$$

so we have

$$e^{ia\omega \cdot x} = \frac{1}{2\pi \hat{\sigma}(a)} \int_{\mathbb{R}} \sigma(\omega \cdot x + b) e^{-iab} db.$$

Thus

$$\begin{aligned}f(x) &= \int_{\mathbb{R}^d} e^{i\omega x} \hat{f}(\omega) d\omega \\ &= \int_{\mathbb{R}^d} \int_{\mathbb{R}} \frac{1}{2\pi \hat{\sigma}(a)} \sigma(\frac{\omega}{a} x + a) \hat{f}(\omega) e^{-iab} db d\omega.\end{aligned}$$

The above integral is on an unbounded domain, but the decay assumption on the Fourier transform of f allows us to normalize the integral in the ω deriction. By the triangle inequality and the boundedness of $x \in \Omega$, we have

$$|\frac{\omega}{a} \cdot x + b| \geq \max(0, |b| - \frac{R|\omega|}{|a|}).$$

where R is the maximum norm of an element of Ω. WLOG, we can translate Ω so that it contains the origin and $R \leq \operatorname{diam}(\Omega)$. Combining this with the polynomial decay of ω implies that

$$\begin{aligned}|\sigma^{(k)}(\frac{\omega}{a} \cdot x + b)| &\leq C_p (1 + |\frac{\omega}{a} \cdot x + b|)^{-p} \\ &\leq C_p (1 + \max(0, |b| - \frac{R|\omega|}{|a|}))^{-p}.\end{aligned}$$

Thus the function h defined by

$$h(b, \omega) = (1 + \max(0, |b| - \frac{R|\omega|}{|a|}))^{-p}$$

provides an upper bound on $\sigma^{(k)}(\frac{\omega}{a} \cdot x + b)$ uniformly in x. Moreover, we calculate that

$$\begin{aligned}\int_{\mathbb{R}} h(b, \omega) db &= \int_{|b| \leq \frac{R|\omega|}{|a|}} db + 2 \int_{b > \frac{R|\omega|}{|a|}} (1 + b - \frac{R|\omega|}{|a|})^{-p} db \\ &= 2R|a|^{-1}|\omega| + 2[(1-p)^{-1} \times (1 + b - \frac{R|\omega|}{|a|})^{1-p}]_{\frac{R|\omega|}{|a|}}^{\infty} \\ &= 2R|a|^{-1}|\omega| + \frac{2}{p-1} \\ &\leq C_1(p, \operatorname{diam}(\Omega), \sigma)(1 + |\omega|).\end{aligned}$$

Combining the above with our assumption on the Fourier transform we get

$$I(p,\Omega,\sigma,f) = \int_{\mathbb{R}^d}\int_{\mathbb{R}}(1+|\omega|)^m h(b,\omega)|\hat{f}(\omega)|dbd\omega \tag{1}$$
$$\leq C_1(p,\mathrm{diam}(\Omega),\sigma)\|\mathrm{f}\|_{\mathcal{B}^{\mathrm{m+1}}}. \tag{2}$$

Now we use this to introduce a probability measure λ on $\mathbb{R}^{d+1}$ given by

$$d\lambda = \frac{1}{I(p,\Omega,\sigma,f)}(1+|\omega|)^m h(b,\omega)|\hat{f}(\omega)|dbd\omega,$$

using this we write

$$f(x) = \mathbb{E}_{d\lambda}(J(\omega,b)e^{i\theta(\omega,b)}\sigma(\frac{\omega}{a}x+b)),$$

where

$$\theta(\omega,b) = \theta(\hat{f}(\omega)) - \theta(\hat{\sigma}(a)) - ab$$

and

$$J(\omega,b) = (2\pi|\hat{\sigma}(a)|)^{-1}I(p,\Omega,\sigma,f)(1+|\omega|)^{-m}h(b,\omega)^{-1}.$$

We denote the real part of $e^{i\theta(\omega,b)}$ as $\chi(\omega,b)\in[-1,1]$, then we have

$$f(x) = \mathbb{E}_{d\lambda}(J(\omega,b)\chi(\omega,b)\sigma(\frac{\omega}{a}x+b)).$$

We denote $f(x) = \mathbb{E}_{d\lambda}(f_{\omega,b}(x))$. Then we use Lemma 1 from Barron 1993 to conclude that for each n there exists an f_n which is a convex combination of at most n distinct $f_{\omega,b}$, and thus $f_n\in\Sigma_d^n(\sigma)$, such that

$$\|f-f_n\|_{H^m}(\Omega) \leq Cn^{-\frac{1}{2}}, \tag{3}$$

where $C = \sup_{\omega,b}\|f_{\omega,b}\|_{H^m(\Omega)}$. Now, since Ω is bounded, it has finite measure, and we use Cauchy-Schwartz to get

$$\|f_{\omega,b}\|_{H^m(\Omega)}\| \leq |\Omega|^{\frac{1}{2}}\|f_{\omega,b}\|_{W^{m,\infty}(\Omega)},$$

so we only need to bound $\|D_x^\alpha f_{\omega,b}\|_{L^\infty(\Omega)}$ for each $|\alpha|\leq m$.

$$\begin{aligned}\|D_x^\alpha f_{\omega,b}\|_{L^\infty(\Omega)} &\leq \|J(\omega,b)D_x^\alpha\sigma(a^{-1}\omega x+b)\|_{L^\infty(\Omega)}\\ &\leq (2\pi|a^{|\alpha|}\hat{\sigma(a)}|)^{-1}I(p,\Omega,\sigma,f)(1+|\omega|)^{-m})\times\|h(b,\omega)^{-1}D_x^\alpha\sigma(a^{-1}\omega x+b)\|_{L^\infty(\Omega)}.\end{aligned}$$

Since $|\alpha|\leq m, \sigma\in W^{m,\infty}$, we have

$$|D_x^\alpha\sigma(a^{-1}\omega x+b)| \leq |a|^{-|\alpha|}(1+|\omega|)^m\sigma^{(|\alpha|)}(a^{-1}\omega x+b).$$

So we get

$$\|D_x^\alpha f_{\omega,b}\|_{L^\infty(\Omega)} \leq (2\pi|a^{|\alpha|}\hat{\sigma}(a)|)^{-1}I(p,\Omega,\sigma,f)\times\|h(b,\omega)^{-1}\sigma^{(|\alpha|)}(a^{-1}\omega x+b)\|_{L^\infty(\Omega)}.$$

What's more we have

$$\|h(b,\omega)^{-1}\sigma^{(|\alpha|)}(a^{-1}\omega x+b)\|_{L^\infty(\Omega)} \leq C_p.$$

So we obtain

$$\sup_{\omega,b}\|f_{\omega,b}\|_{H^m(\Omega)} \leq |\Omega|^{\frac{1}{2}}(2\pi|\hat{\sigma}(a)|)^{-1}I(p,\Omega,\sigma,f)C_p\sum_{|\alpha|\leq m}|a|^{-|\alpha|}.$$

Finally using Equations (1) and (3) we get

$$\inf_{f_n\in\Sigma_d^n(\sigma)}\|f-f_n\|_{H^m(\Omega)} \leq |\Omega|^{\frac{1}{2}}\sqrt{n_0}C(p,m,\dim(\Omega),\sigma)n^{-\frac{1}{2}}\|f\|_{\mathcal{B}^{m+1}}.$$

□

Now we are ready to give a proof of Theorem 2.

Theorem 2 (ICGD on n-layer NNs)**.** *Under Assumption 3.1 and Assumption 3.2, there exists a family of a_n-layer transformers (with activation functions satisfying the general decay condition in Definition 2.3) such that for any input data $(\mathcal{D}, \mathbf{x}_{N+1})$ and any $\mathbf{w}$, a transformer performs approximate gradient descent on the n-layer neural networks parameter $\mathbf{w}$:*

$$\mathbf{w}_\eta^+ = \mathrm{Proj}_{\mathcal{W}}\left(\mathbf{w} - \eta(\nabla \mathrm{L_N}(\mathbf{w}) + \epsilon(\mathbf{w}))\right),\ \|\epsilon(\mathbf{w})\|_2 \le \eta\epsilon,$$

where a_n satisfies $\mathcal{O}(a_n) = \mathcal{O}(n) + \mathcal{O}(a_{n-1})$. Furthermore, the upper bound for number of heads and hidden dimension for the transformer is :

$$\max_{l\in[a_n]} M^{(l)} \le \mathcal{O}(nK^2\epsilon^{-2}), \max_{l\in[a_n]} D^{(l)} \le \mathcal{O}(nK^2\epsilon^{-2}),$$

where K denotes the maximum width of the neural networks: $K = \max\{K_0, K_1, \cdots, K_n\}$.

Proof. We note that

$$\nabla_{\mathbf{w}} L_N(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \partial_1 l(\mathrm{pred}(\mathbf{x_i}; \mathbf{w}), \mathrm{y_i}) \cdot \nabla_{\mathbf{w}} \mathrm{pred}(\mathbf{x_i}; \mathbf{w}), \tag{4}$$

where $\partial_1 l$ is the partial derivative of l with respect to the first component. Recall in Definition 2.4 we denoted $\mathbf{v}_{i,j}$ to be the j-th row of $\mathbf{W}^{(i)}$. Then we have

$$\nabla_{\mathbf{v}_{i,j}} \mathrm{pred}(\mathbf{x}; \mathbf{w}) = \sum_{k=1}^{K_{n-1}} u_k r'(\mathbf{v}_{n-1,k} r(\mathbf{W}^{(n-2)} r(\cdots))) \nabla_{\mathbf{v}_{i,j}} \mathbf{v}_{n-1,k} r(\mathbf{W}^{(n-2)} r(\cdots)).$$

for $i \in [n-1], j \in [K_i]$ and

$$\nabla_{u_k} \mathrm{pred}(\mathbf{x}; \mathbf{w}) = r(\mathbf{v}_{n-1,k}^\top r(\cdots)).$$

We'll later show that it is this difference in gradient that mainly contributes to the growth of transformer layers required.

Now we use Lemma 3.3 to approximate the functions $r(t), \partial_1 l(t, y)$ and $s \cdot r'(t)$. Note that in the following we denote $\boldsymbol{\omega}_i \cdot \mathbf{x} + b_i$ as $\langle \mathbf{a}_i, [\mathbf{x}; 1]\rangle$.

- The function $r(t)$ is approximated by $\bar{r}(t)$ on $[-R_1, R_1]$:

  $$\bar{r}(t) = \sum_{m=1}^{M_1} \beta_m^1 \sigma(\langle \mathbf{a}_m^1, [t; 1]\rangle) \quad \text{with} \quad M_1 \le \mathcal{O}(\epsilon_r^{-2})$$

  such that $\|r(t) - \bar{r}(t)\|_{L^\infty([-R_1, R_1])} \le \epsilon_r$.

- The function $(t, y) \mapsto \partial_1 l(t, y)$ is approximated by $g(t, y)$ on $[-R_2, R_2]^2$:

  $$g(t, y) = \sum_{m=1}^{M_2} \beta_m^2 \sigma(\langle \mathbf{a}_m^2, [t; y; 1]\rangle) \quad \text{with} \quad M_2 \le \mathcal{O}(\epsilon_l^{-2})$$

  such that $\|g(t, y) - \partial_1 l(t, y)\|_{L^\infty([-R_2, R_2]^2)} \le \epsilon_l$.

- The function $(s, t) \mapsto s \cdot r'(t)$ is approximated by $P(s, t)$ on $[-R_3, R_3]^2$:

  $$P(s, t) = \sum_{m=1}^{M_3} \beta_m^3 \sigma(\langle \mathbf{a}_m^3, [s; t; 1]\rangle) \quad \text{with} \quad M_3 \le \mathcal{O}(\epsilon_p^{-2})$$

  such that $\|P(s, t) - s \cdot r'(t)\|_{L^\infty([-R_2, R_2]^2)} \le \epsilon_p$.

- The function $(u, v) \mapsto u \cdot v$ is approximated by $Q(u, v)$ on $[-R_4, R_4]^2$:

  $$Q(u, v) = \sum_{m=1}^{M_4} \beta_m^4 \sigma(\langle \mathbf{a}_m^4, [u; v; 1]\rangle) \quad \text{with} \quad M_4 \le \mathcal{O}(\epsilon_q^{-2})$$

  such that $\|Q(u, v) - u \cdot v\|_{L^\infty([-R_4, R_4]^2)} \le \epsilon_q$.

$n-2$ **attention only layers:** In the first attention-only layer, the transformer maps

$$\mathbf{h}_i = [\mathbf{x}_i; y_i; \mathbf{w}; \mathbf{0}; 1; t_i] \mapsto \mathbf{h}_i^{(1)} = [\mathbf{x}_i; y_i; \mathbf{w}; \mathbf{W}^{(2)}(\bar{r}(\mathbf{W}^{(1)}\mathbf{x})); \mathbf{0}; 1; t_i],$$

in the second attention-only layer, the transformer maps

$$\mathbf{h}_i^{(1)} \mapsto \mathbf{h}_i^{(2)},$$

where

$$\mathbf{h}_i^{(1)} = [\mathbf{x}_i; y_i; \mathbf{w}; \mathbf{W}^{(2)}(\bar{r}(\mathbf{W}^{(1)}\mathbf{x})); \mathbf{0}; 1; t_i],$$

and

$$\mathbf{h}_i^{(2)} = [\mathbf{x}_i; y_i; \mathbf{w}; \mathbf{W}^{(3)}(\bar{r}(\mathbf{W}^{(2)}(\bar{r}(\mathbf{W}^{(1)}\mathbf{x})))); \mathbf{0}; 1; t_i].$$

In the p-th layer, the transformer maps

$$\mathbf{h}_i^{(p-1)} \mapsto \mathbf{h}_i^{(p)},$$

where

$$\mathbf{h}_i^{(p-1)} = [\mathbf{x}_i; y_i; \mathbf{w}; \mathbf{W}^{(2)}(\bar{r}(\mathbf{W}^{(1)}\mathbf{x})); \cdots; \mathbf{W}^{(p)}(\bar{r}(\mathbf{W}^{(p-1)}\cdots\bar{r}(\mathbf{W}^{(1)}\mathbf{x}))); \mathbf{0}; 1; t_i],$$

and

$$\mathbf{h}_i^{(p)} = [\mathbf{x}_i; y_i; \mathbf{w}; \mathbf{W}^{(2)}(\bar{r}(\mathbf{W}^{(1)}\mathbf{x})); \cdots; \mathbf{W}^{(p+1)}(\bar{r}(\mathbf{W}^{(p)}\cdots\bar{r}(\mathbf{W}^{(1)}\mathbf{x}))); \mathbf{0}; 1; t_i].$$

We then prove why a transformer can achieve this mapping by taking the p-th layer as an example.

We denote the k-th element of $\mathbf{W}^{(p)}(\bar{r}(\mathbf{W}^{(p-1)}\cdots\bar{r}(\mathbf{W}^{(1)}\mathbf{x})))$ as $\overline{\text{pred}}_{p,k}(\mathbf{x})$. Consider the matrices $\{\mathbf{Q}^{(p)}_{k',k,m}, \mathbf{K}^{(p)}_{k',k,m}, \mathbf{V}^{(p)}_{k',k,m}\}_{k'\in[K_{p+1}],k\in[K_p],m\in[M_1]}$ so that for all $i,j \in N+1$, we have

$$\mathbf{Q}^{(p)}_{k',k,m}\mathbf{h}_i = \begin{bmatrix} \mathbf{a}^1_m[1]\cdot\overline{\text{pred}}_{p,k}(\mathbf{x}_i) \\ \mathbf{a}^1_m[2] \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{K}^{(p)}_{k',k,m}\mathbf{h}_j = \begin{bmatrix} 1 \\ 1 \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{V}^{(p)}_{k',k,m}\mathbf{h}_j = \beta^1_m \cdot \mathbf{v}_{p+1,k'}[k]\mathbf{e}_{D_{k'}}.$$

Here $D_{k'}$ denotes the place in the column vector $\mathbf{h}_i^{(p)}$ where the sum below is stored. Then we compute the update on the column $\mathbf{h}_i^{(p-1)}$:

$$\sum_{m\in[M_1],k\in[K_p],k'\in[K_{p+1}]} \sigma(\left\langle \mathbf{Q}^{(p)}_{k',k,m}\mathbf{h}_i, \mathbf{K}^{(p)}_{k',k,m}\mathbf{h}_j \right\rangle)\mathbf{V}^{(p)}_{k',k,m}\mathbf{h}_j = \sum_{k'=1}^{K_{p+1}} \left(\sum_{k=1}^{K_p} \mathbf{v}_{p+1,k'}[k]\bar{r}(\overline{\text{pred}}_{p,k}(\mathbf{x}))\cdot\mathbf{e}_{D_{k'}} \right).$$

This is exactly $\mathbf{W}^{(p+1)}(\bar{r}(\mathbf{W}^{(p)}\cdots\bar{r}(\mathbf{W}^{(1)}\mathbf{x}_i)))$.

The Attention sub-layer of the $n-1$th layer: The $n-1$-th attention sub-layer follows the protocol of the mapping of the previous layers.Thus after the $n-1$-th attention sub-layer, we have $\overline{\text{pred}}_n(\mathbf{x}_i;\mathbf{w})$ stored in the output column $\mathbf{h}_i^{(n-1.5)}$.

Now we calculate $|\overline{\text{pred}}_n(\mathbf{x}_i) - \text{pred}_n(\mathbf{x}_i)|$. We first consider:

$$\bar{r}(\mathbf{W}^{(2)}(\bar{r}(\mathbf{W}^{(1)}\mathbf{x}))) - r(\mathbf{W}^{(2)}(r(\mathbf{W}^{(1)}\mathbf{x}))).$$

Its first element is $\bar{r}(\mathbf{v}_{2,1}\bar{r}(\mathbf{W}^{(1)}\mathbf{x})) - r(\mathbf{v}_{2,1}r(\mathbf{W}^{(1)}\mathbf{x}))$, we have

$$\begin{aligned} |\bar{r}(\mathbf{v}_{2,1}\bar{r}(\mathbf{W}^{(1)}\mathbf{x})) - r(\mathbf{v}_{2,1}r(\mathbf{W}^{(1)}\mathbf{x}))| &\le |\bar{r}(\mathbf{v}_{2,1}\bar{r}(\mathbf{W}^{(1)}\mathbf{x})) - r(\mathbf{v}_{2,1}\bar{r}(\mathbf{W}^{(1)}\mathbf{x}))| \\ &\quad + |r(\mathbf{v}_{2,1}\bar{r}(\mathbf{W}^{(1)}\mathbf{x})) - r(\mathbf{v}_{2,1}r(\mathbf{W}^{(1)}\mathbf{x}))| \\ &\le \epsilon_r + K_1 L_r m_{2,1}\epsilon_r, \end{aligned}$$

where $L_r = \max|r'(t)|$, $m_{2,1}$ is the element with the largest absolute value in $\mathbf{v}_{2,1}$. We denote $K := \max\{K_0,\cdots,K_n\}, m := \max_{i\in[n],j\in K_i} m_{i,j}$, and $L := mKL_r$, then by induction we get

$$|\overline{\text{pred}}_n(\mathbf{x}_i) - \text{pred}_n(\mathbf{x}_i)| \le \frac{L^{n-1}-1}{L-1} mK\epsilon_r.$$

To clarify, the n here is the depth of the neural network, not the transformer layer.

The MLP sub-layer of the $n-1$-th layer: In this feed forward layer we pick matrices $\mathbf{W}_1, \mathbf{W}_2$ such that $\mathbf{W}_1$ maps

$$\mathbf{W}_1\mathbf{h}_i^{(n-1.5)} = [\mathbf{a}_m^2[1] \cdot \overline{\text{pred}}(\mathbf{x}_i; \mathbf{w}) + \mathbf{a}_m^2[2] \cdot y_i' + \mathbf{a}_m^2[3] - R_2(1-t_i)]_{m\in[M_2]},$$

and the entries of $\mathbf{W}_2$ are $(\mathbf{W}_2)_{(j,m)} = \beta_m^2 1\{j = D_0\}$, where D_0 is the place in $\mathbf{h}_i^{(n-1.5)}$ right next to $\mathbf{W}^{(n)}(\bar{r}(\mathbf{W}^{(n-1)}\cdots\bar{r}(\mathbf{W}^{(1)}\mathbf{x})))$. So

$$\begin{aligned}\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{h}_i^{(n-1.5)}) &= \sum_{m\in[M_3]} \sigma(\langle \beta_m^2, [\overline{\text{pred}}(\mathbf{x}; \mathbf{w}); y_i'; 1]\rangle - R_2(1-t_j)) \cdot \beta_m^2 \mathbf{e}_{D_0} \\ &= 1\{t_j = 1\} \cdot g(\overline{\text{pred}}(\mathbf{x}_i; \mathbf{w}), y_i') \cdot \mathbf{e}_{D_0}.\end{aligned}$$

So if we abbreviate $g_i = 1\{t_j = 1\} \cdot g(\overline{\text{pred}}(\mathbf{x}_i; \mathbf{w}), y_i')$, we get the output of this sub-layer is

$$\mathbf{h}_i^{(n-1)} = [\mathbf{x}_i; y_i; \mathbf{w}; \mathbf{W}^{(2)}(\bar{r}(\mathbf{W}^{(1)}\mathbf{x})); \cdots; \mathbf{W}^{(n)}(\bar{r}(\mathbf{W}^{(n-1)}\cdots\bar{r}(\mathbf{W}^{(1)}\mathbf{x}))); g_i; \mathbf{0}; 1; t_i].$$

$$\begin{aligned}|g_i - \partial_1 l(\text{pred}(\mathbf{x}_i; \mathbf{w}), y_i)| &\le |g(\overline{\text{pred}}(\mathbf{x}_i; \mathbf{w}), y_i) - \partial_1 l(\overline{\text{pred}}(\mathbf{x}_i; \mathbf{w}), y_i)| + |\partial_1 l(\overline{\text{pred}}(\mathbf{x}_i; \mathbf{w}), y_i) - \partial_1 l(\text{pred}(\mathbf{x}_i; \mathbf{w}), y_i)| \\ &\le \epsilon_l + \frac{L^{n-1}-1}{L-1} mKL_l\epsilon_r \\ &\le \epsilon_l + \frac{L_l}{L_r} L^{n-1}\epsilon_r.\end{aligned}$$

where $L_l = \max_{(t,y)\in[-R_2,R_2]^2} |\partial_{tt}^2 l(t, y)|$.

Other layers to compute the approximate gradient: Now, we look at the gradient of the neural networks again:

$$\nabla_{\mathbf{v}_{i',j'}}\text{pred}(\mathbf{x}; \mathbf{w}) = \sum_{k=1}^{K_n} u_k r'(\mathbf{v}_{n-1,k}^r(\mathbf{W}^{(n-2)}r(\cdots)))\nabla_{\mathbf{v}_{i',j'}}\mathbf{v}_{n-1,k}r(\mathbf{W}^{(n-2)}r(\cdots))$$

for $i' \in [n-1], j' \in [K_i]$ and

$$\nabla_{u_k}\text{pred}(\mathbf{x}; \mathbf{w}) = r(\mathbf{v}_{n-1,k}r(\cdots)).$$

We observe that the term $\nabla_{\mathbf{v}_{i',j'}}\mathbf{v}_{n-1,k}r(\mathbf{W}^{(n-2)}r(\cdots))$ is in fact the gradient of an $n-1$ layer neural network, and a transformer needs a_{n-1} layers to compute and store these gradients for all $i' \in [n-1], j' \in [K_{i'}]$. After these a_{n-1} layers, the approximate gradients are already stored in the hidden space of $\mathbf{h}_i$, and we denote the approximation of $\nabla_{\mathbf{v}_{i',j'}}\mathbf{v}_{n-1,k}r(\mathbf{W}^{(n-2)}r(\cdots))$ as $s_{i',j'}^{(n-1)}$. Now consider the matrices $\{\mathbf{Q}_{k,m}, \mathbf{K}_{k,m}, \mathbf{V}_{k,m}\}_{k\in[K_n],m\in[M_3]}$ so that for all $i, j \in [N+1]$ we have

$$\mathbf{Q}_{k,m}\mathbf{h}_i = \begin{bmatrix} \mathbf{a}_m^3[1] \\ \mathbf{a}_m^3[2] \cdot \overline{\text{pred}}_{n-1,k}(\mathbf{x}_i; \mathbf{w}) \\ \mathbf{a}_m^3[3] \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{K}_{k,m}\mathbf{h}_j = \begin{bmatrix} s_{i',j'}^{(n-1)} \\ 1 \\ 1 \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{V}_{k,m}\mathbf{h}_j = \beta_m^3 \cdot u_k \mathbf{e}_{D_{i',j'}}.$$

Here $D_{i',j'}$ denotes the place where we store the gradient of $\mathbf{v}_{i',j'}$. A simple calculation yields

$$\sum_{m\in[M_1],k\in[K_n]} \sigma(\langle \mathbf{Q}_{k,m}\mathbf{h}_i, \mathbf{K}_{k,m}\mathbf{h}_j\rangle)\mathbf{V}_{k,m}\mathbf{h}_j = \sum_{k=1}^{K_n} u_k P(s_{i',j'}^{(n-1)}, \overline{\text{pred}}_{n-1,k}(\mathbf{x}_i; \mathbf{w})) \cdot \mathbf{e}_{D_{i',j'}},$$

which approximates $\nabla_{\mathbf{v}_{i',j'}}\text{pred}(\mathbf{x}_i; \mathbf{w})$.

Now that all the terms that approximate $\nabla_{\mathbf{v}_{i,j}}\text{pred}(\mathbf{x}_i; \mathbf{w})$ are stored in the hidden space in a consecutive way, then they automatically form the approximation of $\nabla_{\mathbf{w}}\text{pred}(\mathbf{x}_i; \mathbf{w})$, we denote the approximation as $\mathbf{b}_i$, we simply need another attention layer to compute the gradient of loss function.

Consider the matrices $\{\mathbf{Q}_{k,m}, \mathbf{K}_{k,m}, \mathbf{V}_{k,m}\}_{m\in[M_4],k\in[\sum_{i=1}^n K_i]}$ so that for all $i, j \in [N+1]$ we have

$$\mathbf{Q}_m\mathbf{h}_i = \begin{bmatrix} \mathbf{a}_m^4[1] \\ \mathbf{a}_m^4[2] \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{K}_m\mathbf{h}_j = \begin{bmatrix} g_j \\ \mathbf{b}_j[k] \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{V}_m\mathbf{h}_j = -\frac{(N+1)\eta\beta_m^4}{N}\cdot\mathbf{e}_{D_k}.$$

Now we get

$$\begin{aligned}
\mathbf{g}(\mathbf{w}) &:= \frac{1}{N+1}\sum_{j=1}^{N+1}\sum_{(k,m)}\sigma(\langle\mathbf{Q}_{k,m}\mathbf{h}_i, \mathbf{K}_{k,m}\mathbf{h}_j\rangle)\mathbf{V}_{k,m}\mathbf{h}_j \\
&= -\frac{\eta}{N}\sum_{j=1}^{N+1}\sum_{k=1}^{\sum_{i=1}^n K_i} Q(g_j, \mathbf{b}_j[k])\cdot\mathbf{e}_{D_k} \\
&= -\frac{\eta}{N}\sum_{j=1}^{N+1}\text{approximate}(\partial_1 \mathrm{l}(\mathrm{pred}(\mathbf{x_j};\mathbf{w}), \mathrm{y_j}))\cdot\text{approximate}(\nabla_{\mathbf{w}}\mathrm{pred}(\mathbf{x_j};\mathbf{w})).
\end{aligned}$$

Thus $\eta^{-1}\mathbf{g}(\mathbf{w})$ approximates $\nabla\hat{L}_N(\mathbf{w})$, and requiring $\|\eta^{-1}\mathbf{g}(\mathbf{w}) + \nabla\hat{L}_N(\mathbf{w})\|_2 \le \epsilon$ yields the upper bound for the number of heads: $\mathcal{O}(nK^2\epsilon^{-2})$ and hidden dimension: $\mathcal{O}(nK^2\epsilon^{-2})$.

Total number of layers: From the analysis above, the total number of transformer layers required is $\mathcal{O}(a_n) = \mathcal{O}(a_{n-1}) + \mathcal{O}(n)$, and it is straightforward to check that a_n is of order $\mathcal{O}(n^2)$. This completes the proof. □

Corollary 3.7 (multi-step ICGD on n-layer NNs)**.** *For $l \ge 1$, suppose $\nabla L_N(\mathbf{w})$ is M-Lipschitz on $\mathcal{W}$ and $L_N(\mathbf{w})$ is strongly convex with modulus m. Then under Assumption 3.1 and Assumption 3.2, the (la_n)-layer transformers in Theorem 2 approximates gradient descent and the output satisfies:*

$$\|\hat{\mathbf{w}}^l - \mathbf{w}_{\mathrm{True}}\|_2 \le (M^{-1}(1+\eta M)^l + (1-\eta\frac{2Mm}{M+m})^{\frac{l}{2}})\epsilon,$$

where $\hat{\mathbf{w}}^l$ is the output of the transformer, $\mathbf{w}_{\mathrm{True}}$ is the true value of the neural networks parameter.

Proof. we have

$$\|\hat{\mathbf{w}}^l - \mathbf{w}_{\mathrm{True}}\|_2 \le \|\hat{\mathbf{w}}^l - \mathbf{w}_{\mathrm{GD}}^l\|_2 + \|\mathbf{w}_{\mathrm{GD}}^l - \mathbf{w}_{\mathrm{True}}\|_2.$$

Now we bound the two terms separately.

Let $C = 1 + \eta M$. We prove by induction that

$$\|\hat{\mathbf{w}}^l - \mathbf{w}_{\mathrm{GD}}^l\|_2 \le \frac{C^l - 1}{C-1}\cdot\eta\epsilon.$$

for all $l \ge 0$. $l = 0$ follows by setting $\hat{\mathbf{w}}^0 = \mathbf{w}_{\mathrm{GD}}^0 = \mathbf{w}^0$. Suppose the result holds for l. Then for $l+1$ we have

$$\|\hat{\mathbf{w}}^{(l+1)} - \mathbf{w}_{\mathrm{GD}}^{(l+1)}\|_2 \le \|\hat{\mathbf{w}}^l - \eta(\nabla f(\hat{\mathbf{w}}^l) - \epsilon^l) - (\mathbf{w}_{GD}^l - \eta\nabla f(\mathbf{w}_{GD}^l))\|_2 \le C\|\hat{\mathbf{w}} - \mathbf{w}_{\mathrm{GD}}^l\|_2 + \eta\epsilon \le \frac{C^{l+1}-1}{C-1}\eta\epsilon.$$

Thus we get

$$\|\hat{\mathbf{w}}^{l+1} - \mathbf{w}_{\mathrm{GD}}^{l+1}\|_2 \le \frac{C^l}{1+\eta M - 1}\cdot\eta\epsilon = M^{-1}(1+\eta M)^l\epsilon. \tag{5}$$

For the second term, let $c = 1 - \dfrac{2Mm\eta}{M+m}$ and $\mathbf{w}_* = \mathbf{w}_{\mathrm{True}}$. Due to the iteration of gradient descent, we have

$$\|\mathbf{w}_{\mathrm{GD}}^{l+1} - \mathbf{w}_*\|_2^2 = \|\mathbf{w}_{\mathrm{GD}}^l - \mathbf{w}_*\|_2^2 - 2\eta\nabla L(\mathbf{w}_{\mathrm{GD}}^l)(\mathbf{w}_{\mathrm{GD}}^l - \mathbf{w}_*) + \eta^2\|\nabla L(\mathbf{w}_{\mathrm{GD}}^l)\|_2^2.$$

Here we use the inequality

$$\nabla L(\mathbf{w}_{\mathrm{GD}}^l)\cdot(\mathbf{w}_{\mathrm{GD}}^l - \mathbf{w}_*) = (\nabla L(\mathbf{w}_{\mathrm{GD}}^l) - \nabla L(\mathbf{w}_*))\cdot(\mathbf{w}_{\mathrm{GD}}^l - \mathbf{w}_*) \ge \frac{Mm}{M+m}\|\mathbf{w}_{\mathrm{GD}}^l - \mathbf{w}_*\|_2^2 + \frac{1}{M+m}\|\nabla L(\mathbf{w}_{\mathrm{GD}}^l)\|_2^2.$$

It then follows that

$$\|\mathbf{w}_{\mathrm{GD}}^{l} - \mathbf{w}_*\|_2^2 \leq (1 - \frac{2Mm\eta}{M+m})\|\mathbf{w}_{\mathrm{GD}}^{l-1} - \mathbf{w}_*\|_2^2 - (\frac{2\eta}{M+m} - \eta^2)\|\nabla L(\mathbf{w}_{\mathrm{GD}}^{l})\|_2^2 \tag{6}$$

$$\leq c\|\mathbf{w}_{\mathrm{GD}}^{l-1} - \mathbf{w}_*\|_2^2 \tag{7}$$

when η is small enough. Combining Equation (5) and Equation (6) immediately yields the result.

□

C. In-context Learning of Function Classes

We provide the proof of theorems in Section 4 here.

Theorem 3. *For any given $\epsilon > 0$, let $f(\mathbf{x}) = \mathbf{1}(\|A\mathbf{x} + \mathbf{b}\| \leq r)$ be the indicator of unit ball. There exists a cL-layer transformer with*

$$\max_{l\in[cL]} M^{(l)} \leq \mathcal{O}(\frac{1}{\delta\epsilon^2}), \max_{l\in[cL]} D^{(l)} \leq \mathcal{O}(\frac{1}{\delta\epsilon^2}),$$

where c is a constant, such that it performs approximate gradient descent on a 3-layer, $\mathcal{O}(\delta^{-1/2})$-wide neural network which δ-approximates $f(\mathbf{x})$: the l-th layer's output is $\mathbf{h}_i^{(cl)} = [\mathbf{x}_i; y_i'; \hat{\mathbf{w}}^l; \mathbf{0}; 1; t_i]$ for $i \in [N+1]$, and

$$\hat{\mathbf{w}}^{(l)} = \mathrm{Proj}_{\mathcal{W}}(\hat{\mathbf{w}}^{(l-1)} - \eta(\nabla \mathrm{L_N}(\hat{\mathbf{w}}^{(l-1)}) + \epsilon(\hat{\mathbf{w}}^{(l-1)}))),$$

where $\|\epsilon(\mathbf{w})\|_2 \leq \epsilon$. Moreover, the prediction of the transformer $\hat{y}_{N+1}$ satisfies

$$|\hat{y}_{N+1} - y_{N+1}| \leq \mathcal{O}(\epsilon + \delta).$$

Proof. By Lemma 4.2, there exists a 3-layer neural network with width $\mathcal{O}(\delta^{-1/2})$ that δ-approximates $f(\mathbf{x})$. By Theorem 2 we know that for a transformer to learn an n-layer, K-width neural network, the upper bound for the number of heads and hidden layer is

$$\max_{l\in[a_n]} M^{(l)} \leq \mathcal{O}(nK^2\epsilon^{-2}), \quad \max_{l\in[a_n]} D^{(l)} \leq \mathcal{O}(nK^2\epsilon^{-2}).$$

Thus an a_n layer transformer of the size above can perform one step of approximate gradient decent on the neural network parameter. Notice that $a_n = \mathcal{O}(n^2)$, letting $n = 3$ and $K = \mathcal{O}(\delta^{-1/2})$ immediately yields:

$$\max_{l\in[cL]} M^{(l)} \leq \mathcal{O}(\delta^{-1}\epsilon^{-2}), \quad \max_{l\in[cL]} D^{(l)} \leq \mathcal{O}(\delta^{-1}\epsilon^{-2}),$$

where $c = \mathcal{O}(9)$ is a constant. Also, by Corollary 3.7 we have for proper L $|\hat{y}_{N+1} - y_{\mathrm{NN}}| \leq \mathcal{O}(\epsilon)$ (see remark), and we know that y_{NN} δ-approximates y_{N+1}, thus we get

$$|\hat{y}_{N+1} - y_{N+1}| \leq \mathcal{O}(\epsilon + \delta),$$

this completes the proof.

□

Theorem 4. *Let $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x}\rangle$ be a linear function. There exists a $2L$-layer transformer with*

$$\max_{l\in[2L]} M^{(l)} \leq \mathcal{O}(\epsilon^{-2}), \max_{l\in[2L]} D^{(l)} \leq \mathcal{O}(\epsilon^{-2}),$$

such that it performs approximate gradient descent on a 2-layer, width 2 neural network which equals $f(\mathbf{x})$: the l-th layer's output is $\mathbf{h}_i^{(2l)} = [\mathbf{x}_i; y_i'; \hat{\mathbf{w}}^l; \mathbf{0}; 1; t_i]$ for $i \in [N+1]$, and

$$\hat{\mathbf{w}}^{(l)} = \mathrm{Proj}_{\mathcal{W}}(\hat{\mathbf{w}}^{(l-1)} - \eta(\nabla \mathrm{L_N}(\hat{\mathbf{w}}^{(l-1)}) + \epsilon(\hat{\mathbf{w}}^{(l-1)}))),$$

where $\|\epsilon(\mathbf{w})\|_2 \leq \epsilon$. Moreover, the prediction of the transformer $\hat{y}_{N+1}$ satisfies

$$|\hat{y}_{N+1} - y_{N+1}| \leq \mathcal{O}(\epsilon).$$

Proof. Since $\langle \mathbf{w}, \mathbf{x} \rangle = \dfrac{\sigma(\mathbf{w}^\top x) - \sigma(-\mathbf{w}^\top x)}{2}$, we know that a 2-layer, width d neural network exactly represents the linear function $f(\mathbf{x})$, so the transformer only needs to learn the neural network in-context. By Theorem 1 we know that for a transformer to learn the 2-layer network, the upper bound for the number of heads and hidden layer is

$$\max_{l\in[2L]} M^{(l)} \leq \mathcal{O}(\epsilon^{-2}), \quad \max_{l\in[2L]} D^{(l)} \leq \mathcal{O}(nK^2\epsilon^{-2}).$$

Thus a 2 layer transformer of the size above can perform one step of approximate gradient decent on the neural network parameter.

Also, by Corollary 3.7 we have for proper L $|\hat{y}_{N+1} - y_{\mathrm{NN}}| \leq \mathcal{O}(\epsilon)$ (see remark), and we know that y_{NN} equals y_{N+1}, thus we get

$$|\hat{y}_{N+1} - y_{N+1}| \leq \mathcal{O}(\epsilon),$$

this completes the proof. □

Theorem 5. *For any d and $n > 2$, let $f \in \mathcal{W}^{n,\infty}([0,1]^d)$. Choose any $\epsilon > 0$, there exists a $\mathcal{O}(\ln^2(1/\delta)L)$-layer ($k_\delta L$-layer) transformer with*

$$\max_{l\in[k_\delta L]} M^{(l)} \leq \mathcal{O}(\frac{1}{\delta^{2d/n}\epsilon^2}), \max_{l\in[k_\delta L]} D^{(l)} \leq \mathcal{O}(\frac{1}{\delta^{2d/n}\epsilon^2})$$

such that it performs approximate gradient descent on a $c(\ln(1/\delta)+1)$-layer, $\delta^{-d/n}$-wide neural network which δ-approximates $f(\mathbf{x})$: the l-th layer's output is $\mathbf{h}_i^{(2l)} = [\mathbf{x}_i; y_i'; \hat{\mathbf{w}}^l; \mathbf{0}; 1; t_i]$ for $i \in [N+1]$, and

$$\hat{\mathbf{w}}^{(l)} = \mathrm{Proj}_{\mathcal{W}}(\hat{\mathbf{w}}^{(l-1)} - \eta(\nabla \mathrm{L_N}(\hat{\mathbf{w}}^{(l-1)}) + \epsilon(\hat{\mathbf{w}}^{(l-1)}))),$$

where $\|\epsilon(\mathbf{w})\|_2 \leq \epsilon$. Moreover, the prediction of the transformer $\hat{y}_{N+1}$ satisfies

$$|\hat{y}_{N+1} - y_{N+1}| \leq \mathcal{O}(\epsilon + \delta).$$

Proof. By Lemma 4.4, there exists a $c(\ln(1/\delta)+1)$-layer neural network with width $\mathcal{O}(\delta^{-d/n})$ that δ-approximates $f(\mathbf{x})$. By Theorem 2 we know that for a transformer to learn an n-layer, K-width neural network, the upper bound for the number of heads and hidden layer is

$$\max_{l\in[a_n]} M^{(l)} \leq \mathcal{O}(nK^2\epsilon^{-2}), \quad \max_{l\in[a_n]} D^{(l)} \leq \mathcal{O}(nK^2\epsilon^{-2}).$$

Thus an a_n layer transformer of the size above can perform one step of approximate gradient decent on the neural network parameter. Notice that $a_n = \mathcal{O}(n^2)$, letting $n = c(\ln(1/\delta)+1)$ and $K = \mathcal{O}(\delta^{-d/n})$ immediately yields:

$$\max_{l\in[k_\delta L]} M^{(l)} \leq \mathcal{O}(\delta^{-2d/n}\epsilon^{-2}), \quad \max_{l\in[k_\delta L]} D^{(l)} \leq \mathcal{O}(\delta^{-2d/n}\epsilon^{-2}),$$

where $k_{delta} = c^2(\ln(1/\delta)+1)^2$. Also, by Corollary 3.7 we have for proper L, $|\hat{y}_{N+1} - y_{\mathrm{NN}}| \leq \mathcal{O}(\epsilon)$ (see remark), and we know that y_{NN} δ-approximates y_{N+1}, thus we get

$$|\hat{y}_{N+1} - y_{N+1}| \leq \mathcal{O}(\epsilon + \delta),$$

this completes the proof. □

Of course, linear function is a special case of the smooth function, but the result in linear function is more 'accurate' in the sense that the upper bound for number of heads, hidden layer and the prediction error is tighter than the result in smooth function, and we want to separate them since linear function is an especially important function class in the realm of supervised learning.

D. In-context Algorithm Selection

Lemma 5.1 (Indicator function)**.** *An attention layer with 6 heads can implement the indicator function I_{cls} of the form*

$$I_{cls}(\mathcal{D}) = \frac{1}{N}\sum_{i=1}^{N} I(y_i),$$

where

$$I(y) = \begin{cases} 1 & y \in \{0,1\} \\ 0 & y \in [-\epsilon, \epsilon] \cup [1-\epsilon, 1+\epsilon] \\ \textit{linear interpolation} & \textit{otherwise} \end{cases}$$

.

The proof of this lemma follow Bai et al. 2024.

Proof. We first notice that

$$\begin{aligned} I(y) &= \sigma(\frac{y+\epsilon}{\epsilon}) - 2\sigma(\frac{y}{\epsilon}) + \sigma(\frac{y-\epsilon}{\epsilon}) + \sigma(\frac{y-(1-\epsilon)}{\epsilon}) - 2\sigma(\frac{y-1}{\epsilon}) + \sigma(\frac{y-(1+\epsilon)}{\epsilon}) \\ &:= \sum_{m=1}^{6} a_m \sigma(b_m y + c_m). \end{aligned}$$

We can construnct an attention layer with parameters $\theta = \{(\mathbf{Q}_m, \mathbf{K}_m, \mathbf{V}_m)\}_{m=1}^{6}$ with 6 heads such that

$$\mathbf{Q}_m \mathbf{h}_i = [b_m; c_m; \mathbf{0}_{D-2}], \quad \mathbf{K}_m \mathbf{h}_j = [y_j; 1; \mathbf{0}_{D-2}], \quad \mathbf{V}_m \mathbf{h}_j = [\frac{N+1}{N} a_m \cdot t_j; \mathbf{0}_{D-1}],$$

then for every $i \in [N+1]$ we have

$$\begin{aligned} &\sum_{m=1}^{6} \frac{1}{N+1} \sum_{j \in [N+1]} \sigma(\langle \mathbf{Q}_m \mathbf{h}_i, \mathbf{K}_m \mathbf{h}_j \rangle)[\mathbf{V}_m \mathbf{h}_j]_1 \\ =&\frac{1}{N}\sum_{j=1}^{N} I(y) = I_{\text{cls}}(\mathcal{D}). \end{aligned}$$

This completes the proof. □

Theorem 6. *Given $\epsilon > 0, \delta > 0$, there exists a $(c+2)L+1$-layer transformer (the constant c is stated in Theorem 3) with*

$$\max_{l \in [(c+2)L]} M^{(l)} \le \mathcal{O}(\frac{1}{\delta\epsilon^2}), \max_{l \in [(c+2)L]} D^{(l)} \le \mathcal{O}(\frac{1}{\delta\epsilon^2})$$

that can perform In-context algorithm selection on linear regression and classification tasks, where its output satisfies

$$|\hat{y} - y_{N+1}| \le \mathcal{O}(\epsilon + \delta).$$

Proof. In the pretraining stage, the transformer already learned a 2-layer NN to represent the linear function $\langle \mathbf{w}, \mathbf{x} \rangle$, and a 3-layer NN to approximate the classification function $\text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)$. We know that by Theorem 4 and Theorem 3 (set $A = \mathbf{w}^\top, \mathbf{b} = \mathbf{0}, r = 0$), we have two transformers with

$$\begin{aligned} &\max_{l \in [cL]} \le \mathcal{O}(\delta^{-1}\epsilon^{-2}), \quad \max_{l \in [cL]} D^{(l)} \le \mathcal{O}(\delta^{-1}\epsilon^{-2}) \\ &\max_{l \in [2L]} M^{(l)} \le \mathcal{O}(\epsilon^{-2}), \quad \max_{l \in 2L} D^{(l)} \le \mathcal{O}(\epsilon^{-2}) \end{aligned}$$

that outputs prediction $\hat{y}_{\text{cls}}$ and $\hat{y}_{\text{lr}}$ respectively. What's more ,

$$|\hat{y}_{\text{cls}} - y_{N+1}^{\text{cls}}| \leq \mathcal{O}(\epsilon + \delta),$$
$$|\hat{y}_{\text{ls}} - y_{N+1}^{\text{ls}}| \leq \mathcal{O}(\epsilon),$$

where y_{N+1} represents the real value of y corresponding to x_{N+1}.

These two transformers can be joined into a single transformer by adding the 2 layers of the second transformer below the first c-layer transformer. The $c+2$ layers together implements one step of approximate gradient descent on both tasks. Now we join the parameters of the two transformers to get a transformer that outputs

$$\hat{y} = I_{\text{cls}}\hat{y}_{\text{cls}} + (1 - I_{\text{cls}})\hat{y}_{\text{lr}}.$$

The number of heads and hidden dimension of the transformer satisfies

$$\max_{l \in [(c+2)L]} \leq \mathcal{O}(\delta^{-1}\epsilon^{-2}), \qquad \max_{l \in [(c+2)L]} D^{(l)} \leq \mathcal{O}(\delta^{-1}\epsilon^{-2}).$$

We assume for linear regression, the data generated is not concentrated around $\{0, 1\}$. Then the indicator function is either 1 or 0. This yields that the final prediction of the transformer $\hat{y}$ can $\mathcal{O}(\epsilon + \delta)$-approximate y_{N+1}, which completes the proof. □