TRANSFORMERS PERFORM IN-CONTEXT LEARNING THROUGH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformer based neural sequence models exhibit remarkable ability to do in-context learning. Given some training examples, a pre-trained model can make accurate predictions on a novel input. This paper studies why transformers can learn different types of function classes in context. We first show by construction that transformers implement approximate gradient descent on parameters of neural networks and provide an upper bound for number of heads, hidden dimension, and number of layers of the transformer. We also show that transformers can learn deep and narrow neural networks, which has better approximation capabilities compared to shallow and wide neural networks, using less resource. Our results move beyond linearity in terms of in-context learning instances and provide an 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 NLP problems where large language models 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 can neural sequence models learn in-context remain a black box. Previous work mainly focus on two perspectives of in-context learning. One perspective explores what function classes can transformers learn in-context(Garg et al., 2022), another explains why transformers can implement learning algorithms (Akyürek et al., 2022).

The paper tries 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 adapt the former perspective to investigate the hypothesis that when trained properly, transformers can perform approximate gradient descent on parameters of neural networks without any parameter update or fine tuning, and these neural networks are 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 the neural networks. We start by investigating on 2-layer neural networks and then generalize it to the n-layer neural networks setting. Upper bound for the number of heads, hidden dimension as well as number of layers needed of the transformer is provided, among which the number of layers is presented in a recursive fashion for the n-layer neural networks setting.

In Section 4, we view neural networks as bridges for transformers to learn function classes in-context, and provide an analysis on the resources it cost for a transformer to approximate the same function classes through neural networks of different depths and widths. We showcase that for transformers to learn indicator functions in-context, 2-layer neural networks are not sufficient as bridges since it will cause the number of heads of the transformer to be unacceptably large (for certain function classes), while deeper and narrower neural networks which achieve the same approximation accuracy cost less resource (number of parameter matrices) of the transformer. We also present a condition on

when deep networks does better than shallow ones in terms of approximating smooth functions and requiring smaller transformer size.

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, while Akyürek et al. (2022), Von Oswald et al. (2023) and Dai et al. (2022) all understand in-context learning as transformers performing gradient descent. These works all only focus on linear models or their variants without providing an error bound for gradient descent steps. A more recent work (Bai et al., 2023) also investigates gradient descent on more general functions, like 2-layer neural networks and demonstrates the model selection ability of transformers. We extend the their result on 2-layer neural networks to an n-layer neural networks setting and also provide a tighter bound on the number of heads required of the transformer.

Neural networks and approximation theorems People are interested in the the approximation abilities of neural networks. Many results have shown the universal approximation property of neural networks in approximating different function classes (Hornik et al., 1989; Hornik, 1991; Barron, 1993). While these universal approximation theorems focus on neural networks with certain depths, more recent work starts to explore the expressing power of deep neural networks due to their development and success. Yarotsky (2017); Liang and Srikant (2016) both show the approximation abilities of deep neural networks. Safran and Shamir (2017) shows the width-depth tradeoffs of neural networks by proving the inapproximability with 2-layer neural networks an 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 feed forward layer, which in essence is an 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 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:*

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

where σ is an activation function (not necessarily restricted to the ReLU function). For each column:

$$[\mathrm{Attn}_{\boldsymbol{\theta}}(\mathbf{H})]_{\mathrm{i}} = \mathbf{h}_{\mathrm{i}} + \frac{1}{\mathrm{N}}\sum_{\mathrm{m}=1}^{\mathrm{M}}\sum_{\mathrm{j}=1}^{\mathrm{N}}\sigma\left(\langle \mathbf{Q}_{\mathrm{m}}\mathbf{h}_{\mathrm{i}}, \mathbf{K}_{\mathrm{m}}\mathbf{h}_{\mathrm{j}}\rangle\right)\cdot \mathbf{V}_{\mathrm{m}}\mathbf{h}_{\mathrm{j}}.$$

Definition 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 σ is an activation function. For each column:

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

We doesn't require the transformer activation function to be restricted to a specific type, and next we give a definition on the class of activation functions we focus on.

Definition 3. *(General decay condition) We call an activation function σ satisfies the general decay condition if $\sigma \in W^{m,\infty}_{loc}(\mathbb{R})$ 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*

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

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

We briefly note that here $W^{m,\infty}$ denotes the Sobolev space, and most common activation functions do satisfy the general decay condition (Siegel and Xu, 2020).

2.2 NEURAL NETWORKS

We formulate the mathematical representation of an n-layer neural network as below:

Definition 4. *(n-layer neural networks) We denote the output of an n-layer neural network on the input $x \in \mathbb{R}^d$ as*

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

where r is an activation function 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 vector.

2.3 IN-CONTEXT LEARNING

Here we introduce our in-context learning (ICL) setting. The training examples (prompts) are sampled from a distribution P, denoted 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 instance 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 $\mathbf{p}_i$ are vectors in hidden space of the form

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

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 $\tilde{\mathbf{H}}$ in the position next to y_N.

3 GRADIENT DESCENT ON MULTI-LAYER NEURAL NETWORK

We begin our analysis on 2-layer neural networks, and then generalize it to the n-layer networks setting. To perform gradient descent on the neural networks, we consider the following optimization problem on the loss function:

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

Now we present necessary assumptions 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:

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

Assumption 1. *Both the activation function r in neural networks and the loss function l has finite Barron norm.*

As discussed in Barron (1993), B_s is closed to multiplication, linear combination and translation. Also, sigmoidal functions, functions with derivatives of sufficiently high order and Boolean functions are all in B_s, so this should include most common activation functions and loss functions in practice. We also point out that in Bai et al. (2023) they 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 2. *$\mathcal{W}$ as the domain of $\mathbf{w}$ is compact and there exists some MLP layer parameter such that the MLP layer projects $\mathbf{w}$ to $\mathcal{W}$.*

3.1 Gradient Descent on 2-layer neural networks

A 2-layer neural network can be written as

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

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

We note that

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

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

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

We show below that a 2-layer transformer can compute one step of approximate gradient descent, following the intuition of Siegel and 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 $\mathbf{w}$.

Theorem 1 (ICGD on 2-layer NNs)**.** *Under* **Assumption 1** *and* **Assumption 2***, there exists a family of 2-layer transformers (with activation functions satisfying the general decay condition) 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 $\mathbf{w}$:*

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

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

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

Remark 1. *We note here 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.*

3.2 Gradient Descent on n-layer neural networks

As in Definition 4, n-layer neural networks can be formulated as

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

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

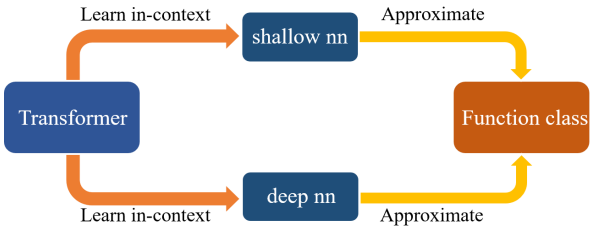

Figure 1: Neural Networks as bridges for transformers to learn function classes in-context. Though transformers can learn both deep and shallow neural networks in-context, learning them requires different transformer sizes.

Theorem 2 (ICGD on n-layer NNs)**.** *Under* **Assumption 1** *and* **Assumption 2***, there exists a family of a_n-layer transformers (with activation functions satisfying the general decay condition) 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 $\mathbf{w}$:*

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

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 :

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

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

Remark 2. *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.*

Trade-off of width and depth of neural networks We note here that people are interested in the approximation capabilities of neural networks, thus it's worthwhile discussing how the trade-off between width and depth of neural networks can affect the approximation error of the gradient descent step. If we control the error in the approximate gradient descent to be ϵ, 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 maintaing the same approximation error.

4 Neural networks as bridges for transformer to 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. Our theorem 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 as shown in Fig. 1. But how much resources 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 two types of function classes: the indicator functions of L_2 balls and smooth functions.

4.1 Indicator functions of L_2 balls

Safran and 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 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 and 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 2. *Given $\delta > 0$, for any activation function σ satisfying Assumption 1 in Eldan and 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. Now using these lemmas we obtain the following theorem on transformer resource it takes to learn these two networks.

Theorem 3. *Let $f(\mathbf{x}) = \mathbf{1}(\|A\mathbf{x} + \mathbf{b}\| \leq r)$ be the indicator of unit ball. Consider two neural networks $\mathrm{NN_A}$ and $\mathrm{NN_B}$ that approximates the function f with error $\epsilon \in (0, 1)$, where $\mathrm{NN_A}$ is a 2-layer, width $\mathcal{O}(\exp(\epsilon^{-1/4}))$ and $\mathrm{NN_B}$ is a 3-layer, width $\mathcal{O}(\epsilon^{-1})$. It takes two transformers $\mathrm{TF_A}, \mathrm{TF_B}$ to learn these two neural networks. $\mathrm{TF_A}$ needs $\mathcal{O}(\exp(2\epsilon^{-1/4}))$ parameter matrices and $\mathrm{TF_B}$ needs $\mathcal{O}(\epsilon^{-1})$ parameter matrices.*

Proof. Recall Theorem 2 and we know that the number of transformer layers required is $\mathcal{O}(nK^2)$ (Here we the gradient error ϵ is absorbed in the $\mathcal{O}$ notation since it is a fixed constant independent of the networks approximation error). So the transformer size is determined by number of layers times number of heads, which is $\mathcal{O}(n^2K^2)$. Now bring in the width of $\mathrm{NN_A}$ and $\mathrm{NN_B}$ immediately yields the results, concluding the proof. □

The theorem points out the exponential explosion in transformer size if a 2-layer neural network is learned to approximate the indicator. However, a 3-layer neural network requires much smaller transformer size. This showcases the neccessity of bringing deeper neural networks into consideration.

4.2 SMOOTH FUNCTIONS

We use the approximation results of deep neural networks achieved by Yarotsky (2017).

Lemma 3. *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. *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 $c = c(f, L) > 0$.*

While Lemma 3 gives the upper bound for a relatively deep neural network, Lemma 4 show the slow approximation of smooth functions by shallow networks.

Below we state a formal theorem for the resource of a transformer it takes to learn two neural network: one is deep and narrow, another is shallow and wide.

Theorem 4. *For any d and $n > 2$, let $f \in \mathcal{W}^{n,\infty}([0, 1]^d)$. Consider two neural networks $\mathrm{NN_A}$ and $\mathrm{NN_B}$ that approximates the function f with error $\epsilon \in (0, 1)$. $\mathrm{NN_A}$ has depth $\mathcal{O}(\ln(1/\epsilon) + 1)$ and width $\mathcal{O}(\epsilon^{-d/n})$ as in Lemma 3, and $\mathrm{NN_B}$ has depth L and width $\mathcal{O}(\epsilon^{-1/(2(L-1))})$ as in Lemma 4. We also have two transformers, $\mathrm{TF_A}$ and $\mathrm{TF_B}$, pre-trained to learn these neural networks. If we fix the accuracy of gradient descent in Theorem 2, then $\mathrm{TF_A}$ needs $\mathcal{O}((\ln(1/\epsilon)^3\epsilon^{-2d/n}))$ parameter matrices and $\mathrm{TF_B}$ needs $\mathcal{O}(\epsilon^{-1/(L-1)})$ parameter matrices. In particular, if $2(L-1)d < n$, then it costs a transformer less resource to train neural network A than B.*

This theorem can be proved in exactly the same way as Theorem 3 thus we ommit the proof. 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 CONCLUSION

We provide results on transformers learning n-layer neural networks through gradient descent and view in-context learning of a function as the process of learning neural networks which approximate this function. We also emphasize that our results gives a theoretical bound on the size of the transformer required to implement in-context learning.

Our work shed some light on deeper 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 this way. 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.

REFERENCES

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*, 2022.

Yu Bai, Fan Chen, Huan Wang, Caiming Xiong, and Song Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. *arXiv preprint arXiv:2306.04637*, 2023.

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

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Zhifang Sui, and Furu Wei. Why can gpt learn in-context? language models secretly perform gradient descent as meta optimizers. *arXiv preprint arXiv:2212.10559*, 2022.

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

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

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

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

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

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.

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

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

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pages 35151–35174. PMLR, 2023.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*, 2021.

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

A THEOREM 2

We provide the proof of Theorem 2 here. We note that 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 and 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 5. *Let $\Omega \subset \mathbb{R}^d$ be a bounded domain. If the activation function $\sigma \in W^{m,\infty}_{loc}(\mathbb{R})$ is non-zero and there exists a $\nu \in \Sigma^{n_0}_1(\sigma)$ 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

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

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

Note that when $m = 0$, the Sobolev space H^0 is in effect the L^2 space, and we set $m = 0$ in the proof of Theorem 1.

Proof. We first observe that $\nu \in \Sigma^{n_0}_1(\sigma)$ implies that

$$\Sigma^n_d(\nu) \subset \Sigma^{nn_0}_d(\sigma)$$

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

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

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

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

$$\leq C_1(p,\mathrm{diam}(\Omega),\sigma)\|\mathrm{f}\|_{\mathcal{B}^{m+1}} \tag{2}$$

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 Eqs. (1) and (3) we get

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

□

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

Theorem 2 (ICGD on n-layer NNs)**.** *Under* **Assumption 1** *and* **Assumption 2***, there exists a family of a_n-layer transformers (with activation functions satisfying the general decay condition) 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 $\mathbf{w}$:*

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

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 :

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

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. Moreover we have

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

Now we use Lemma 3 to approximate the functions $r(t), \partial_1 l(t, y)$ and $s \cdot r'(t)$. Note that in the following expressions 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 $[-R_1, R_1]$:

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

  such that $\|r(t) - \bar{r}(t)\|_{L^2([-R_1, R_1])} \leq \epsilon_r$.

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

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

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

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

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

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

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

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

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

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

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

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 taking the p-th layer as an example.

We denote the k-th row of $\mathbf{W}^{(p)}(\bar{r}(\mathbf{W}^{(p-1)} \cdots \bar{r}(\mathbf{W}^{(1)}\mathbf{x})))$ as $\overline{\text{pred}}_{p,k}(\mathbf{x})$, which is a number. 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

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

Here $D_{k'}$ denotes the hidden space in the column vector $\mathbf{h}_i$ where the sum below is stored. Then we compute the update on the column h_i:

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

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

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

$$\mathbf{W}_1\mathbf{h}_i^{(0.5)} = [\mathbf{a}^2_m[1] \cdot \overline{\text{pred}}(\mathbf{x}_i; \mathbf{w}) + \mathbf{a}^2_m[2] \cdot y'_i + \mathbf{a}^2_m[3] - R_2(1-t_i)]_{m \in [M_2]},$$

and the entries of $\mathbf{W}_2$ are $(\mathbf{W}_2)_{(j,m)} = \beta^2_m 1\{j = D_0 + 2\}$. So

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

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 sublayer is

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

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

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

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

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

We observe that the term $\nabla_{\mathbf{v}_{i,j}} \mathbf{v}_{n-1,k}^\top r(\mathbf{W}^{(n-2)} r(\cdots))$ 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 $\nabla_{\mathbf{v}_{i,j}} \mathbf{v}_{n-1,k}^\top r(\mathbf{W}^{(n-2)} r(\cdots))$ 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

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

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

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

Now that all the terms that approximate $\nabla_{\mathbf{v}_{i,j}} \text{pred}(\mathbf{x}; \mathbf{w})$ are stored in the hidden space (we denote them as $v_{i,j}$), we simply need another layer to compute the gradient of loss function.

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

$$\mathbf{Q}_{k,m}\mathbf{h}_i = \begin{bmatrix} \mathbf{a}_m^4[1] \cdot g_i \\ \mathbf{a}_m^4[2] \\ \mathbf{a}_m^4[3] \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{K}_{k,m}\mathbf{h}_j = \begin{bmatrix} 1 \\ v_{i,j} \\ 1 \\ \mathbf{0} \end{bmatrix}, \quad \mathbf{V}_{k,m}\mathbf{h}_j = -\frac{(N+1)\eta\beta_m^4}{N} \cdot \mathbf{e}_{D_{i,j}}.$$

Now we get

$$\begin{aligned} \mathbf{g}(\mathbf{w}) &:= \frac{1}{N+1} \sum_{i=1}^{N+1} \sum_{(k,m)} \sigma(\langle \mathbf{Q}_{k,m}\mathbf{h}_i, \mathbf{K}_{k,m}\mathbf{h}_j \rangle) \mathbf{V}_{k,m}\mathbf{h}_j \\ &= -\frac{\eta}{N} \sum_{i=1}^{N+1} \text{approximate}(\partial_1 \text{l}(\text{pred}(\mathbf{x}_\text{i}; \mathbf{w}), \text{y}_\text{i})) \cdot \text{approximate}(\nabla_{\mathbf{w}} \text{pred}(\mathbf{x}_\text{i}; \mathbf{w})) \end{aligned}$$

Thus $\eta^{-1}\mathbf{g}(\mathbf{w})$ approximates $\nabla \hat{L}_N(\mathbf{w})$, and requiring $\|\eta^{-1}\mathbf{g}(\mathbf{w}) + \nabla \hat{L}_N(\mathbf{w})\|_{L^2} \leq \epsilon$ 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. □