Minimax-optimal nonparametric in-context learning with transformers

Michelle Ching, Ioana Popescu, Nico Smith

Supervisors: Tianyi Ma, William G. Underwood, Richard J. Samworth

Statistical Laboratory, DPMMS

Summer 2025

Table of contents

1. Transformers and their ability to perform regression

2. Why local polynomial regression?

3. Approximation error: Our construction

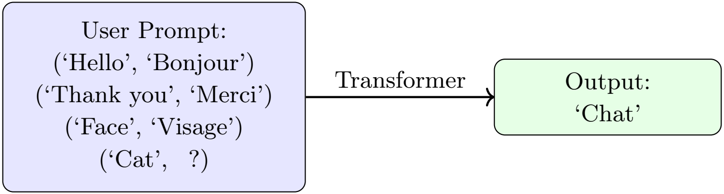
4. Generalisation error

5. References

In-context learning (ICL)

Large Language Models (LLMs) such as ChatGPT, LLaMA, and Gemini have shown remarkable in-context learning (ICL) capabilities.

Given a few examples of a task in their prompt, they can often generalise and produce accurate outputs, even if the task was not part of their training set.



This behaviour suggests that LLMs can learn how to learn from examples in their input.

In-context learning (ICL)

‘Delivered on time, functions great and long battery life’
Positive

‘My first item was faulty. Was delivered a new one that works great though’
Neutral

‘Not as advertised, customer support was unhelpful when I complained’
Negative

‘Not the best option if you want something to last. Poor battery life also’
?

$\downarrow$ Transformer

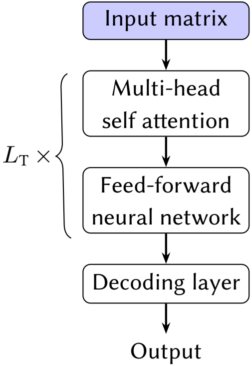
Negative

Transformers

A transformer (TF) is an ML architecture that first embeds words as tokens, denoted $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{n+1} \in \mathbb{R}^{d_\mathrm{e}}$, and then performs pre-trained operations on them.

These operations consist of multi-headed self attention and feed-forward neural network (FNN) layers, stacked in L_T blocks.

The input to a transformer is $\boldsymbol{H} \in \mathbb{R}^{d_\mathrm{e} \times (n+1)}$, with ith column $\boldsymbol{h}_i$. Combined with the decoding layer, the output is a value in $\mathbb{R}$.

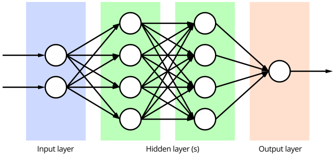


Transformers

- FNN: Applies linear transformations and activations coordinatewise:

$$\boldsymbol{h}_i \mapsto \sigma(\boldsymbol{W}\boldsymbol{h}_i + \boldsymbol{b})$$

with weights $\boldsymbol{W} \in \mathbb{R}^{d_e \times d_e}$, bias $\boldsymbol{b} \in \mathbb{R}^{d_e}$ and $\sigma(x) = \mathrm{ReLU}(x) = \max(x, 0)$.



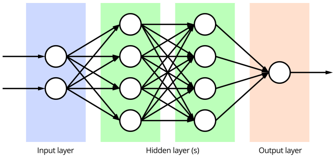
Source*

Transformers

- FNN: Applies linear transformations and activations coordinatewise:

$$\boldsymbol{h}_i \mapsto \sigma(\boldsymbol{W}\boldsymbol{h}_i + \boldsymbol{b})$$

with weights $\boldsymbol{W} \in \mathbb{R}^{d_e \times d_e}$, bias $\boldsymbol{b} \in \mathbb{R}^{d_e}$ and $\sigma(x) = \mathrm{ReLU}(x) = \max(x, 0)$.



Source*

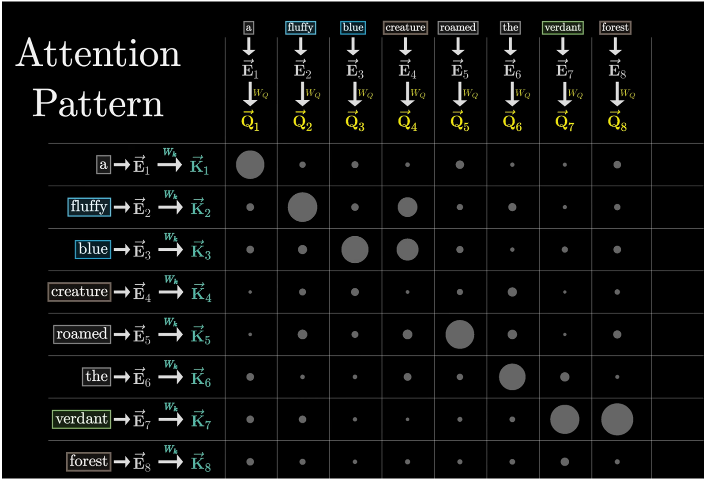
- Attention: Can interact columns:

$$\boldsymbol{h}_i \mapsto \sum_{j=1}^{n} \sigma\big(\langle \boldsymbol{Q}\boldsymbol{h}_i, \boldsymbol{K}\boldsymbol{h}_j \rangle\big)\, \boldsymbol{V}\boldsymbol{h}_j$$

with query $\boldsymbol{Q}$, key $\boldsymbol{K}$ and value $\boldsymbol{V}$ matrices in $\mathbb{R}^{d_e \times d_e}$.

*: https://dailysciencefactsatit.blogspot.com/2023/06/the-game-changing-inventions-and.html

Transformers



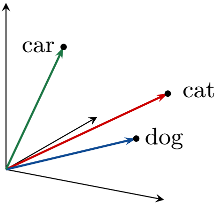
Source: 3Blue1Brown†

†: Attention in transformers, step-by-step | Deep Learning Chapter 6, YouTube, https://www.youtube.com/watch?v=eMlx5fFNoYc&t=1065s

Reframing as a regression task

A transformer embeds words as vectors in a d_{e}-dimensional space (in practice: $d_{\mathrm{e}} \approx 10000$).

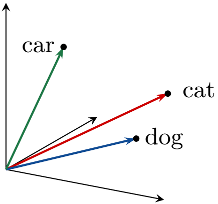
ICL performs regression over these embeddings: from n input-output examples, the model infers a relation and applies it to a new test point.



Reframing as a regression task

A transformer embeds words as vectors in a d_e-dimensional space (in practice: $d_e \approx 10000$).

ICL performs regression over these embeddings: from n input-output examples, the model infers a relation and applies it to a new test point.



Goal: Analyse the theoretical capabilities of transformers to perform regression.

Limitations: We work with tabular data, where regression is more sensible and well-studied – input is in $\mathbb{R}^d$ with $d \approx 10$, and output is in $\mathbb{R}$.

Problem set-up: ICL as regression

Set-up: Let f_X be a density on $\mathbb{R}^d$ and $P_{\mathcal{H}}$ be a distribution on the Hölder class $\mathcal{H}(\alpha, L)$ for some α, L. We consider our data:

- Sample: $s = (\boldsymbol{x}_i, y_i)_{i=1}^{n+1}$ i.i.d. pairs where $\boldsymbol{x}_i \sim f_X$, $m \sim P_{\mathcal{H}}$, ε_i is a bounded noise term and $y_i = m(\boldsymbol{x}_i) + \varepsilon_i$.

$$s : \boxed{\boxed{\boldsymbol{x}_1,\ y_1}\ ,\ \boxed{\boldsymbol{x}_2,\ y_2}\ ,\ \ldots\ ,\ \boxed{\boldsymbol{x}_{n+1},\ y_{n+1}}}$$

- Training set: $\mathcal{S} = (\boldsymbol{x}_i^{(\gamma)}, y_i^{(\gamma)})_{i \in [n+1], \gamma \in [\Gamma]}$ of independent copies of s:

$$\mathcal{S} : \boxed{\boxed{s^{(1)}}\ ,\ \boxed{s^{(2)}}\ ,\ \ldots\ ,\ \boxed{s^{(\Gamma)}}}$$

Assume $\|\boldsymbol{x}_i\|_\infty \leq M_x$, $|y_i| \leq M_y$ almost surely.

Input matrix

Input $\left\{(\boldsymbol{x}_i, y_i)_{i=1}^n; \boldsymbol{x}_{n+1}\right\}$ is embedded, with positional encoding, into:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_n & \boldsymbol{x}_{n+1} \\ y_1 & \cdots & y_n & 0 \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} \\ \mathcal{I}_1 & \cdots & \mathcal{I}_n & \mathcal{I}_{n+1} \\ 1 & \cdots & 1 & 1 \end{bmatrix} \in \mathbb{R}^{d_e \times (n+1)}$$

$$\mathcal{I}_j = \left(\mathbb{1}_{\{j=n+1\}}, \cos\left(\frac{j\pi}{2(n+1)}\right), \sin\left(\frac{j\pi}{2(n+1)}\right)\right)^\top$$

This allows us to distinguish the columns in our constructions.

Remark. In practice, LLMs add positional encoding to the word embedding before the first layer of attention.

Problem set-up: ICL as regression

Denote our transformer class by $\mathcal{T}$. For $f \in \mathcal{T}$, we define

$$\text{Sample loss} : \ell(f,s) = \frac{1}{2}\Big(f\big(\{\boldsymbol{x}_i, y_i\}_{i=1}^n; \boldsymbol{x}_{n+1}\big) - y_{n+1}\Big)^2$$

$$\text{Population risk} : R(f) = \mathbb{E}_s\big[\ell(f,s)\big]$$

$$\text{Empirical risk} : R_\Gamma(f) = \frac{1}{2\Gamma}\sum_{\gamma=1}^{\Gamma}\Big(f\big(\{\boldsymbol{x}_i^{(\gamma)}, y_i^{(\gamma)}\}_{i=1}^n; \boldsymbol{x}_{n+1}^{(\gamma)}\big) - y_{n+1}^{(\gamma)}\Big)^2.$$

Define the empirical minimiser

$$\hat{f} \in \operatorname{argmin}_{f_T \in \mathcal{T}} R_\Gamma(f).$$

Problem set-up: ICL as regression

Denote our transformer class by $\mathcal{T}$. For $f \in \mathcal{T}$, we define

$$\text{Sample loss}: \ell(f,s) = \frac{1}{2}\Big(f\big(\{\boldsymbol{x}_i, y_i\}_{i=1}^n; \boldsymbol{x}_{n+1}\big) - y_{n+1}\Big)^2$$

$$\text{Population risk}: R(f) = \mathbb{E}_s\big[\ell(f,s)\big]$$

$$\text{Empirical risk}: R_\Gamma(f) = \frac{1}{2\Gamma}\sum_{\gamma=1}^{\Gamma}\Big(f\big(\{\boldsymbol{x}_i^{(\gamma)}, y_i^{(\gamma)}\}_{i=1}^n; \boldsymbol{x}_{n+1}^{(\gamma)}\big) - y_{n+1}^{(\gamma)}\Big)^2.$$

Define the empirical minimiser

$$\hat{f} \in \operatorname{argmin}_{f_T \in \mathcal{T}} R_\Gamma(f).$$

Aim: Show the empirical risk minimiser $\hat{f}$, trained on $\mathcal{S}$, can perform well on a new regression task.

Problem set-up

Goal: Bound $\mathbb{E}_{\mathcal{S}}[R(\hat{f})] = \mathcal{O}\big(n^{-\frac{2\alpha}{2\alpha+d}}\big)$, the optimal non-parametric regression minimax rate.

Theorem (Kim et al., 2024). There exists a universal constant $C > 0$ such that:

$$\mathbb{E}_{\mathcal{S}}\left[R(\hat{f})\right] \leq 2 \inf_{f \in \mathcal{T}} R(f) + C\left(\frac{M_y^2}{\Gamma} \log \mathcal{N}\left(\mathcal{T}, \|\cdot\|_{L^\infty}, \frac{1}{\Gamma}\right) + \frac{M_y}{\Gamma}\right)$$

where $\mathcal{N}(\mathcal{T}, \|\cdot\|_{L^\infty}, \delta)$ is the covering number of $\mathcal{T}$ of radius δ with respect to the norm $\|\cdot\|_{L^\infty}$.

Problem set-up

Goal: Bound $\mathbb{E}_{\mathcal{S}}[R(\hat{f})] = \mathcal{O}\big(n^{-\frac{2\alpha}{2\alpha+d}}\big)$, the optimal non-parametric regression minimax rate.

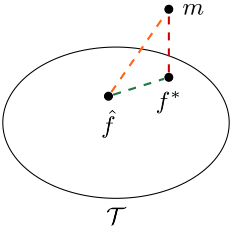
Theorem (Kim et al., 2024). There exists a universal constant $C > 0$ such that:

$$\mathbb{E}_{\mathcal{S}}\left[R(\hat{f})\right] \leq 2 \inf_{f \in \mathcal{T}} R(f) + C\left(\frac{M_y^2}{\Gamma} \log \mathcal{N}\left(\mathcal{T}, \|\cdot\|_{L^\infty}, \frac{1}{\Gamma}\right) + \frac{M_y}{\Gamma}\right)$$

where $\mathcal{N}(\mathcal{T}, \|\cdot\|_{L^\infty}, \delta)$ is the covering number of $\mathcal{T}$ of radius δ with respect to the norm $\|\cdot\|_{L^\infty}$.

Approximation error: How well can a transformer perform? We bound this with an explicit construction f^*, implementing local polynomial regression.

Generalisation error: How badly could the empirical risk minimizer $\hat{f}$ possibly deviate from the true minimizer m?



Local polynomial regression

For $m \in \mathcal{H}(\alpha, L)$, local polynomial regression can obtain the minimax optimal pointwise squared error risk of $\mathcal{O}\left(n^{-\frac{2\alpha}{2\alpha+d}}\right)$.

We fit a degree p polynomial locally at $\boldsymbol{x}_{n+1}$, using weighted least squares with kernel:

$$K_h(\boldsymbol{x}_{n+1} - \boldsymbol{x}_i) := \frac{1}{h^d} K\left(\frac{\boldsymbol{x}_{n+1} - \boldsymbol{x}_i}{h}\right)$$

Note: We illustrate with $d = 1$.

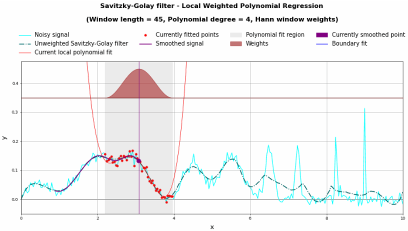


Figure: Local weighted polynomial regression (degree=4) smoothing a signal

Prior works

Prior works:

- linear and ridge regression (Bai et al., 2023)
- Lasso (Bai et al., 2023)
- Nadaraya–Watson (Shen et al., 2025)
- nonparametric regression with basis expansion (Kim et al., 2024)

Local polynomial regression can be written as a weighted least squares problem, whose solution can be computed by gradient descent.

Procedures

To fit a polynomial at x_{n+1}, we must find coefficients for each monomial in $\boldsymbol{P}_h(x_{n+1} - x_i)$, where $\boldsymbol{P}(z) = (1, z, z^2, \ldots, z^p)^\top$, $\boldsymbol{P}_h(\cdot) := \boldsymbol{P}\left(\frac{\cdot}{h}\right)$. Let

$$\boldsymbol{X} := \begin{pmatrix} \boldsymbol{P}_h(x_{n+1} - x_1)^\top \\ \vdots \\ \boldsymbol{P}_h(x_{n+1} - x_n)^\top \end{pmatrix} \in \mathbb{R}^{n\times(p+1)} \qquad \boldsymbol{Y} := (y_1 \cdots y_n)^\top \in \mathbb{R}^n$$

$$\boldsymbol{W} := \operatorname{diag}\big(K_h(x_{n+1} - x_1), \ldots, K_h(x_{n+1} - x_n)\big) \in \mathbb{R}^{n\times n}$$

Procedures

To fit a polynomial at x_{n+1}, we must find coefficients for each monomial in $\boldsymbol{P}_h(x_{n+1} - x_i)$, where $\boldsymbol{P}(z) = (1, z, z^2, \ldots, z^p)^\top$, $\boldsymbol{P}_h(\cdot) := \boldsymbol{P}\left(\frac{\cdot}{h}\right)$. Let

$$\boldsymbol{X} := \begin{pmatrix} \boldsymbol{P}_h(x_{n+1} - x_1)^\top \\ \vdots \\ \boldsymbol{P}_h(x_{n+1} - x_n)^\top \end{pmatrix} \in \mathbb{R}^{n\times(p+1)} \qquad \boldsymbol{Y} := (y_1 \cdots y_n)^\top \in \mathbb{R}^n$$

$$\boldsymbol{W} := \operatorname{diag}\big(K_h(x_{n+1} - x_1), \ldots, K_h(x_{n+1} - x_n)\big) \in \mathbb{R}^{n\times n}$$

And compute the weighted least squares estimator:

$$\begin{aligned} \boldsymbol{\beta}^* &= \underset{\boldsymbol{\beta}\in\mathbb{R}^d}{\operatorname{argmin}} (\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{Y})^\top \boldsymbol{W} (\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{Y}) \\ &= (\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{Y} \\ &= \left(\boldsymbol{X}_W^\top \boldsymbol{X}_W\right)^{-1} \boldsymbol{X}_W^\top \boldsymbol{Y}_W \end{aligned}$$

where $\boldsymbol{X}_W := \sqrt{\boldsymbol{W}}\boldsymbol{X}$, $\boldsymbol{Y}_W := \sqrt{\boldsymbol{W}}\boldsymbol{Y}$

Procedures

We fit $\beta_0 + \beta_1(x_{n+1} - x_i) + \cdots + \beta_p(x_{n+1} - x_i)^p$ as an expansion around x_{n+1}, so we approximate the regression function $m(x_{n+1}) \approx \beta_0$.

Our construction has 2 steps:

Procedures

We fit $\beta_0 + \beta_1(x_{n+1} - x_i) + \cdots + \beta_p(x_{n+1} - x_i)^p$ as an expansion around x_{n+1}, so we approximate the regression function $m(x_{n+1}) \approx \beta_0$.

Our construction has 2 steps:

1: Construct and multiply:

$$\boldsymbol{P}_h(x_{n+1} - x_i),\ \sqrt{K_h(x_{n+1} - x_i)} \quad \rightarrow \quad \boldsymbol{P}_h(x_{n+1} - x_i)\sqrt{K_h(x_{n+1} - x_i)}$$

Procedures

We fit $\beta_0 + \beta_1(x_{n+1} - x_i) + \cdots + \beta_p(x_{n+1} - x_i)^p$ as an expansion around x_{n+1}, so we approximate the regression function $m(x_{n+1}) \approx \beta_0$.

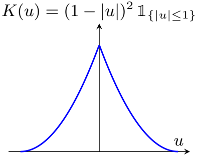
Our construction has 2 steps:

1: Construct and multiply:

$$\boldsymbol{P}_h(x_{n+1} - x_i),\ \sqrt{K_h(x_{n+1} - x_i)} \quad \rightarrow \quad \boldsymbol{P}_h(x_{n+1} - x_i)\sqrt{K_h(x_{n+1} - x_i)}$$

2: Perform gradient descent using the pairs

$$\left(\sqrt{K_h(x_{n+1} - x_i)}\boldsymbol{P}_h(x_{n+1} - x_i), \sqrt{K_h(x_{n+1} - x_i)}y_i\right)$$



Steps of construction

Denoting $\boldsymbol{\Delta}_j := \boldsymbol{x}_{n+1} - \boldsymbol{x}_j$, column j undergoes these changes:

$$\begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \boldsymbol{\Delta}_j/h \\ \mathbf{0}_{2D+1} \\ \mathcal{I}_j \\ 1 \end{bmatrix} \xmapsto{\text{F}} \begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \boldsymbol{\Delta}_j/h \\ \mathbf{0}_{D-d} \\ \sqrt{K_h(\boldsymbol{\Delta}_j)} \\ \mathbf{0}_{D+1} \\ \mathcal{I}_j \\ 1 \end{bmatrix} \xmapsto{\text{A/F}} \begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \boldsymbol{P}_h(\boldsymbol{\Delta}_j) \\ \sqrt{K_h(\boldsymbol{\Delta}_j)} \\ \mathbf{0}_{D+1} \\ \mathcal{I}_j \\ 1 \end{bmatrix} \xmapsto{\text{A/F}} \begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \boldsymbol{P}_h(\boldsymbol{\Delta}_j) \\ \sqrt{K_h(\boldsymbol{\Delta}_j)} \\ \sqrt{K_h(\boldsymbol{\Delta}_j)}\,\boldsymbol{P}_h(\boldsymbol{\Delta}_j) \\ \sqrt{K_h(\boldsymbol{\Delta}_j)}\,y_j \\ \mathcal{I}_j \\ 1 \end{bmatrix} \xmapsto{\text{A}} \begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \boldsymbol{\beta} \\ * \end{bmatrix}$$

We then extract β_0. To perform these operations, we need to multiply, copy and add within/between the columns.

$$\begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \boldsymbol{\Delta}_j/h \\ \mathbf{0}_{2D+1} \\ \mathcal{I}_j \\ 1 \end{bmatrix} \overset{\text{F}}{\longmapsto} \begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \boldsymbol{\Delta}_j/h \\ \mathbf{0}_{D-d} \\ \sqrt{K_h(\boldsymbol{\Delta}_j)} \\ \mathbf{0}_{D+1} \\ \mathcal{I}_j \\ 1 \end{bmatrix} \overset{\text{A}}{\longmapsto} \begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \boldsymbol{P}_h(\boldsymbol{\Delta}_j) \\ \sqrt{K_h(\boldsymbol{\Delta}_j)} \\ \mathbf{0}_{D+1} \\ \mathcal{I}_j \\ 1 \end{bmatrix} \overset{\text{A}}{\longmapsto} \begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \boldsymbol{P}_h(\boldsymbol{\Delta}_j) \\ \sqrt{K_h(\boldsymbol{\Delta}_j)} \\ \sqrt{K_h(\boldsymbol{\Delta}_j)}\,\boldsymbol{P}_h(\boldsymbol{\Delta}_j) \\ \sqrt{K_h(\boldsymbol{\Delta}_j)}\,y_j \\ \mathcal{I}_j \\ 1 \end{bmatrix}$$

Exact construction (more attention): Allows us to build the monomials exactly.

Step 1: Two approaches

$$
\begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \color{red}{\boldsymbol{\Delta}_j/h} \\ \mathbf{0}_{2D+1} \\ \mathcal{I}_j \\ 1 \end{bmatrix}
\xmapsto{\mathrm{F}}
\begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \boldsymbol{\Delta}_j/h \\ \mathbf{0}_{D-d} \\ \color{red}{\sqrt{K_h(\boldsymbol{\Delta}_j)}} \\ \mathbf{0}_{D+1} \\ \mathcal{I}_j \\ 1 \end{bmatrix}
\xmapsto{\mathrm{F}}
\begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \color{red}{\boldsymbol{P}_h(\boldsymbol{\Delta}_j) + \boldsymbol{\delta}_j} \\ \sqrt{K_h(\boldsymbol{\Delta}_j)} \\ \mathbf{0}_{D+1} \\ \mathcal{I}_j \\ 1 \end{bmatrix}
\xmapsto{\mathrm{F}}
\begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \boldsymbol{P}_h(\boldsymbol{\Delta}_j) + \boldsymbol{\delta}_j \\ \sqrt{K_h(\boldsymbol{\Delta}_j)} \\ \color{red}{\sqrt{K_h(\boldsymbol{\Delta}_j)}\,\boldsymbol{P}_h(\boldsymbol{\Delta}_j) + \boldsymbol{\varepsilon}_j} \\ \color{red}{\sqrt{K_h(\boldsymbol{\Delta}_j)}\,y_j + \epsilon_j} \\ \mathcal{I}_j \\ 1 \end{bmatrix}
$$

Approximate construction (more FNN): Can only approximate the monomials with piecewise linear functions. It is quicker and requires less positional encoding.

Remark. A FNN with width $\mathcal{O}(N)$, depth $\mathcal{O}(L)$ can approximate these functions with error $\mathcal{O}\left(N^{-L}\right)$.

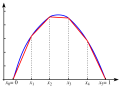


Figure: Piecewise linear approximation of a quadratic

Step 2: Performing gradient descent

Attention can perform 1 step of GD in 1 layer of attention (2 heads).

We rename $\boldsymbol{x}_j = \sqrt{K_h(\boldsymbol{\Delta}_j)}\boldsymbol{P}_h(\boldsymbol{\Delta}_j)$ and $y_j = \sqrt{K_h(\boldsymbol{\Delta}_j)}y_j$.

At the t^{th} step, each column i contains $\boldsymbol{h}_i = \left[\boldsymbol{x}_i,\ y_i,\ \boldsymbol{\beta}_{GD}^t,\ \ast\right]^\top$ and we want to construct $\boldsymbol{\beta}_{GD}^{t+1}$.

$$\cdots \overset{\mathrm{A}}{\longmapsto} \begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \boldsymbol{\beta}_{GD}^t \\ \ast \end{bmatrix} \overset{\mathrm{A}}{\longmapsto} \begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \boldsymbol{\beta}_{GD}^{t+1} \\ \ast \end{bmatrix} \overset{\mathrm{A}}{\longmapsto} \begin{bmatrix} \boldsymbol{x}_j \\ y_j \\ \boldsymbol{\beta}_{GD}^{t+2} \\ \ast \end{bmatrix} \overset{\mathrm{A}}{\longmapsto} \cdots$$

We run gradient descent for τ steps.

Step 2: Performing gradient descent

Our loss function

$$\hat{L}_n(\boldsymbol{\beta}) = \frac{1}{2n}\sum_{j=1}^{n}(\langle \boldsymbol{\beta}, \boldsymbol{x}_j\rangle - y_j)^2$$

gives update formula:

$$\boldsymbol{\beta}_{GD}^{t+1} = \boldsymbol{\beta}_{GD}^{t} - \eta\nabla\hat{L}_n(\boldsymbol{\beta}_{GD}^{t}) = \boldsymbol{\beta}_{GD}^{t} - \frac{\eta}{n}\sum_{j=1}^{n}(\langle \boldsymbol{\beta}_{GD}^{\top}, \boldsymbol{x}_j\rangle - y_j)\boldsymbol{x}_j$$

We construct the term $(\langle \boldsymbol{\beta}_{GD}^{\top}, \boldsymbol{x}_j\rangle - y_j)\boldsymbol{x}_j$ exactly with attention, using $u - v = \sigma(u-v) - \sigma(v-u)$. Recall the form of an attention head:

$$\boldsymbol{h}_i \mapsto \boldsymbol{h}_i + A(\boldsymbol{h}_i) \quad \text{where} \quad A(\boldsymbol{h}_i) = \sum_{j=1}^{n}\sigma\big(\langle \boldsymbol{Q}\boldsymbol{h}_i, \boldsymbol{K}\boldsymbol{h}_j\rangle\big)\,\boldsymbol{V}\boldsymbol{h}_j$$

Step 2: Performing gradient descent

Remark. Recall $\boldsymbol{X}_W = \sqrt{\boldsymbol{W}}\boldsymbol{X}$. To choose the step size η, we look at :

$$\nabla\nabla^\top \hat{L}_n(\boldsymbol{\beta}) = \frac{1}{2n}\nabla\nabla^\top(\boldsymbol{X}_W\boldsymbol{\beta} - Y_W)^\top(\boldsymbol{X}_W\boldsymbol{\beta} - Y_W) = \frac{1}{n}\boldsymbol{X}_W^\top\boldsymbol{X}_W$$

So choose $\eta = \frac{1}{\lambda_+} \leq_{\mathbb{P}} \frac{1}{\lambda_{\max}(\boldsymbol{X}_W^\top\boldsymbol{X}_W/n)}$, where $\lambda_- \leq \lambda_{\min} \leq \lambda_{\max} \leq \lambda_+$, w.h.p.

Step 2 (approximate construction)

Remark. For the approximate construction using FNNs, we initialise with slightly wrong data $\tilde{\boldsymbol{x}}_i$, $\tilde{y}_i$, with error bounded by $\tilde{\varepsilon}$ (in fact data refers to $\sqrt{K_h(\boldsymbol{\Delta}_i)}\boldsymbol{P}_h(\boldsymbol{\Delta}_i)$ and $\sqrt{K_h(\boldsymbol{\Delta}_i)}y_i$).

This error accumulates at each step since we use the wrong loss:

$$\tilde{L}_n(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\boldsymbol{\beta}^\top \tilde{\boldsymbol{x}}_i, \tilde{y}_i) = \frac{1}{2n}\sum_{i=1}^{n}(\boldsymbol{\beta}^\top \tilde{\boldsymbol{x}}_i - \tilde{y}_i)^2$$

However, after t steps, it only accumulates linearly as $\tilde{\varepsilon}\eta t$.

Approximation error (exact construction)

Recall m is the true regression function, $\hat{m}_n$ is the local polynomial regression estimator and f^* is our constructed transformer. We use this to bound the term

$$\inf_{f \in \mathcal{T}} R(f).$$

Approximation error (exact construction)

Recall m is the true regression function, $\hat{m}_n$ is the local polynomial regression estimator and f^* is our constructed transformer. We use this to bound the term

$$\inf_{f \in \mathcal{T}} R(f).$$

Exact construction: Choosing bandwidth $h = h^* = n^{-\frac{1}{2\alpha+d}}$ and defining the noise term $\varepsilon := y - m$ gives

$$\begin{aligned} R(f^*) &= \mathbb{E}\left[(f^* - m)^2\right] + \mathbb{E}\left[(m - y)^2\right] \\ &\leq 2\|f^* - \hat{m}_n\|_\infty^2 + 2\mathbb{E}\left[(\hat{m}_n - m)^2\right] + \mathrm{Var}(\varepsilon) \\ &= \boxed{\mathcal{O}\left(e^{-\frac{\lambda_- \tau}{2\lambda_+}} n^{\frac{d}{2\alpha+d}}\right) + \mathcal{O}\left(n^{-\frac{2\alpha}{2\alpha+d}}\right) + \mathrm{Var}(\varepsilon)} \end{aligned}$$

where for n sufficiently large: $\lambda_- \leq \lambda_{\min} \leq \lambda_{\max} \leq \lambda_+$, w.h.p.

Approximation error (exact construction)

Recall m is the true regression function, $\hat{m}_n$ is the local polynomial regression estimator and f^* is our constructed transformer. We use this to bound the term

$$\inf_{f \in \mathcal{T}} R(f).$$

Exact construction: Choosing bandwidth $h = h^* = n^{-\frac{1}{2\alpha+d}}$ and defining the noise term $\varepsilon := y - m$ gives

$$\begin{aligned} R(f^*) &= \mathbb{E}\left[(f^* - m)^2\right] + \mathbb{E}\left[(m-y)^2\right] \\ &\leq 2\|f^* - \hat{m}_n\|_\infty^2 + 2\mathbb{E}\left[(\hat{m}_n - m)^2\right] + \mathrm{Var}(\varepsilon) \\ &= \boxed{\mathcal{O}\left(e^{-\frac{\lambda_- \tau}{2\lambda_+}} n^{\frac{d}{2\alpha+d}}\right) + \mathcal{O}\left(n^{-\frac{2\alpha}{2\alpha+d}}\right) + \mathrm{Var}(\varepsilon)} \end{aligned}$$

where for n sufficiently large: $\lambda_- \leq \lambda_{\min} \leq \lambda_{\max} \leq \lambda_+$, w.h.p.

So we require $\boxed{\tau \geq \frac{2\lambda_+}{\lambda_-} \log n}$ to match minimax rate.

Approximation error (FNN construction)

FNN construction: Choosing FNNs with width $\mathcal{O}(N)$, depth $\mathcal{O}(L)$:

$$R(f^*) \leq \mathcal{O}\left(n^{\frac{\alpha+p+d}{2\alpha+d}} N^{-L} \tau\right) + \mathcal{O}\left(e^{-\frac{\lambda_- \tau}{2\lambda_+}} n^{\frac{d}{2\alpha+d}}\right) + \mathcal{O}\left(n^{-\frac{2\alpha}{2\alpha+d}}\right) + \mathrm{Var}(\varepsilon)$$

where the first term comes from error in data, accumulated over τ steps of gradient descent. So to match the minimax rate, we take

$$\tau \geq \frac{2\lambda_+}{\lambda_-} \log n, \qquad N = \text{const}, \qquad L = \mathcal{O}\big(\log n\big).$$

Generalisation error & covering numbers

We now want to bound the generalisation error term

$$\frac{M_y^2}{\Gamma} \log \mathcal{N}\Big(\mathcal{T}, \|\cdot\|_{L^\infty}, \frac{1}{\Gamma}\Big) + \frac{M_y}{\Gamma}$$

where $|y_i| \leq M_y$ a.s. and Γ is our number of training samples.

Generalisation error & covering numbers

We now want to bound the generalisation error term

$$\frac{M_y^2}{\Gamma} \log \mathcal{N}\left(\mathcal{T}, \|\cdot\|_{L^\infty}, \frac{1}{\Gamma}\right) + \frac{M_y}{\Gamma}$$

where $|y_i| \leq M_y$ a.s. and Γ is our number of training samples.

The covering number $\mathcal{N}$ for our transformer class $\mathcal{T}$ is well studied (Bai et al., 2023). It depends on width, depth, number of attention heads, parameter size, and embedding dimension.

These parameter bounds give us a Lipschitz constant for $\mathcal{T}$ so we can construct an $\|\cdot\|_{L^\infty}$ net for $\mathcal{T}$ of a given size.

Generalisation error & covering numbers

$$\frac{M_y^2}{\Gamma} \log \mathcal{N}\Big(\mathcal{T}, \|\cdot\|_{L^\infty}, \frac{1}{\Gamma}\Big) + \frac{M_y}{\Gamma}$$

For τ steps of gradient descent:

Exact construction:

$$\log \mathcal{N}\Big(\mathcal{T}, \|\cdot\|_{L^\infty}, \frac{1}{\Gamma}\Big) \leq \mathcal{O}\Big(\tau^2 \log(\Gamma\tau)\Big)$$

$$\boxed{\text{Exact Generalisation Error} \leq \mathcal{O}\big(\frac{\tau^2 \log(\Gamma\tau)}{\Gamma}\big)}$$

Generalisation error & covering numbers

$$\frac{M_y^2}{\Gamma}\log\mathcal{N}\Big(\mathcal{T},\|\cdot\|_{L^\infty},\frac{1}{\Gamma}\Big)+\frac{M_y}{\Gamma}$$

For τ steps of gradient descent:

Exact construction:

$$\log\mathcal{N}\Big(\mathcal{T},\|\cdot\|_{L^\infty},\frac{1}{\Gamma}\Big)\leq\mathcal{O}\Big(\tau^2\log(\Gamma\tau)\Big)$$

$$\boxed{\text{Exact Generalisation Error}\leq\mathcal{O}\big(\frac{\tau^2\log(\Gamma\tau)}{\Gamma}\big)}$$

FNN construction: width $\mathcal{O}(N)$, depth $\mathcal{O}(L)$

$$\log\mathcal{N}\Big(\mathcal{T},\|\cdot\|_{L^\infty},\frac{1}{\Gamma}\Big)\leq\mathcal{O}\Big(N^2(\tau+L)^2\log\big(\Gamma N(\tau+L)\big)\Big)$$

$$\boxed{\text{FNN Generalisation Error}\leq\mathcal{O}\Big(\frac{N^2(\tau+L)^2\log\big(\Gamma N(\tau+L)\big)}{\Gamma}\Big)}$$

Choosing Γ

$$\boxed{\text{Exact Generalisation Error} \leq \mathcal{O}\Big(\frac{\tau^2 \log(\Gamma\tau)}{\Gamma}\Big)}$$

$$\boxed{\text{FNN Generalisation Error} \leq \mathcal{O}\big(\frac{N^2(\tau+L)^2 \log(\Gamma N(\tau+L))}{\Gamma}\big)}$$

Recall we earlier took

$$\tau \geq \frac{2\lambda_+}{\lambda_-}\log(n), \qquad N = \text{const}, \qquad L = \mathcal{O}\big(\log(n)\big)$$

so to match the minimax rate of $\mathcal{O}\big(n^{-\frac{2\alpha}{2\alpha+d}}\big)$, we need to take:

$$\text{Exact:} \qquad \Gamma \geq \mathcal{O}\big(n^{\frac{2\alpha}{2\alpha+d}}(\log n)^3\big)$$

$$\text{FNN:} \qquad \Gamma \geq \mathcal{O}\big(n^{\frac{2\alpha}{2\alpha+d}}(\log n)^3\big)$$

Final results

Lemma. In the exact set up with $h = h^* = n^{-\frac{1}{2\alpha+d}}$, $\tau \geq \frac{2\lambda_+}{\lambda_-} \log(n)$, and $\Gamma \geq \mathcal{O}\big(n^{\frac{2\alpha}{2\alpha+d}} \log(n)^3\big)$, we achieve the non-parametric minimax rate

$$\mathbb{E}_{\mathcal{S}}[R(\hat{f})] \leq \mathcal{O}\Big(n^{-\frac{2\alpha}{2\alpha+d}}\Big) + \mathsf{Var}(\varepsilon)$$

Final results

Lemma. In the exact set up with $h = h^* = n^{-\frac{1}{2\alpha+d}}$, $\tau \geq \frac{2\lambda_+}{\lambda_-}\log(n)$, and $\Gamma \geq \mathcal{O}\big(n^{\frac{2\alpha}{2\alpha+d}}\log(n)^3\big)$, we achieve the non-parametric minimax rate

$$\mathbb{E}_{\mathcal{S}}[R(\hat{f})] \leq \mathcal{O}\big(n^{-\frac{2\alpha}{2\alpha+d}}\big) + \mathsf{Var}(\varepsilon)$$

Lemma. In the FNN set up with $h = h^* = n^{-\frac{1}{2\alpha+d}}$, $\tau \geq \frac{2\lambda_+}{\lambda_-}\log(n)$, $\Gamma \geq \mathcal{O}\big(n^{\frac{2\alpha}{2\alpha+d}}\log(n)^3\big)$, and constant N, $L = \mathcal{O}\big(\log n\big)$, we achieve the non-parametric minimax rate

$$\mathbb{E}_{\mathcal{S}}[R(\hat{f})] \leq \mathcal{O}\big(n^{-\frac{2\alpha}{2\alpha+d}}\big) + \mathsf{Var}(\varepsilon)$$

Conclusion

Achievements:

1. We obtain the optimal non-parametric regression minimax rate of $\mathcal{O}\left(n^{-\frac{2\alpha}{2\alpha+d}}\right)$ of pointwise squared error risk in both the FNN construction and exact construction.

Conclusion

Achievements:

1. We obtain the optimal non-parametric regression minimax rate of $\mathcal{O}\left(n^{-\frac{2\alpha}{2\alpha+d}}\right)$ of pointwise squared error risk in both the FNN construction and exact construction.

2. Γ, number of training samples, is controlled by $\mathcal{O}\left(n^{\frac{2\alpha}{2\alpha+d}}\log(n)^3\right)$, which is less than previous requirements in the literature.

References I

Y. Bai, F. Chen, H. Wang, C. Xiong, and S. Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection, 2023. URL `https://arxiv.org/abs/2306.04637`.

J. Kim, T. Nakamaki, and T. Suzuki. Transformers are minimax optimal nonparametric in-context learners. In The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024. URL `https://openreview.net/forum?id=hF6vatntqc`.

Z. Shen, A. Hsu, R. Lai, and W. Liao. Understanding in-context learning on structured manifolds: Bridging attention to kernel methods, 2025. URL `https://arxiv.org/abs/2506.10959`.