Project: Theory of LLMs

By Jayin Khanna

Breakdown of paper: Efficient and Minimax Optimal In-context Nonparametric Regression with Transformers

https://arxiv.org/abs/2601.15014

Theory of LLMs: Idea of paper

What I removed from latex

Project buffer

▼ Questions to Answer

1. use his settings, begin equation end equation
2. Good notation: Set of all trainable params notation as a tuple
3. neural tangent kernel (what is this?; how is it relevant in the paper):
   → Neural Tangent Kernel- https://rbcborealis.com/research-blogs/neural-tangent-kernel-applications/
4. vectors as bold
5. how are they invoking READ in the paper: is it very very imp in their formulations; M bound; why do they need it; will it break their formulations; In fact what even is READ function;
6. why they are doing what
7. what is the design matrix notation in regression
8. 17→ 18→ 19 in
    1. to understand in B9, B10
9.

▼ Relevant Papers

1. https://arxiv.org/abs/2605.27567
2. https://arxiv.org/abs/2402.01817
3. https://arxiv.org/abs/2106.01345
4. https://arxiv.org/abs/2202.05607

▼ The Problem Statement

What We Are Trying to Do

We have an unknown function $m : [0,1]^d \to \mathbb{R}$. We cannot see m directly. We only see noisy evaluations of it: $Y_i = m(X_i) + \varepsilon_i$, where $X_i \in [0,1]^d$ is an input, $Y_i \in \mathbb{R}$ is the noisy output, and ε_i is noise with $\mathbb{E}[\varepsilon_i \mid X_i] = 0$.

Given n such pairs $(X_1, Y_1), \ldots, (X_n, Y_n)$ and a new input X_{n+1}, we want to predict $Y_{n+1} = m(X_{n+1}) + \varepsilon_{n+1}$. Since ε_{n+1} is unpredictable noise, the best we can do is predict $m(X_{n+1})$ itself. This is standard nonparametric regression. The twist in this paper is **how** we build the estimator — using a transformer that learns from many such problems simultaneously.

▼ What $m^{(\gamma)}$ Is, and How it is obtained

Formally, for each $\gamma \in \{1, 2, \ldots, \Gamma\}$, the function $m^{(\gamma)} : [0,1]^d \to [-M, M]$ is drawn independently from a distribution $P_{\mathcal{H}}$ over the Hölder ball $\mathcal{H}(d, \alpha, M)$: $m^{(\gamma)} \overset{\text{i.i.d.}}{\sim} P_{\mathcal{H}}, \quad \gamma = 1, \ldots, \Gamma.$

So $m^{(1)}, m^{(2)}, \ldots, m^{(\Gamma)}$ are Γ different unknown regression functions, each drawn from the same distribution but independently of each other. They are all α-Hölder smooth and bounded by M, but otherwise different.

Why do we need many functions? Because we want the transformer to learn a general estimation strategy that works for any α-Hölder function, not just one specific m. By training on many different $m^{(\gamma)}$'s, the transformer is forced to learn the general structure of the problem rather than memorising one particular function.

▼ Pretraining Sequence

A **pretraining sequence** with index γ is simply the complete dataset generated from the γ-th regression function $m^{(\gamma)}$. It consists of n context pairs $(X_1^{(\gamma)}, Y_1^{(\gamma)}), \ldots, (X_n^{(\gamma)}, Y_n^{(\gamma)})$, a query input $X_{n+1}^{(\gamma)}$, and a query label $Y_{n+1}^{(\gamma)}$ to predict.

For each $i \in \{1, \ldots, n+1\}$: $X_i^{(\gamma)} \overset{\text{i.i.d.}}{\sim} f_X$ on $[0,1]^d$, $\quad Y_i^{(\gamma)} = m^{(\gamma)}(X_i^{(\gamma)}) + \varepsilon_i^{(\gamma)}$, $\quad \varepsilon_i^{(\gamma)} \overset{\text{i.i.d.}}{\sim} P_\varepsilon$.

Everything within sequence γ is generated from the same function $m^{(\gamma)}$. Different sequences $\gamma \neq \gamma'$ come from different functions $m^{(\gamma)} \neq m^{(\gamma')}$. Write the context dataset as $D_n^{(\gamma)} := \{(X_i^{(\gamma)}, Y_i^{(\gamma)})\}_{i=1}^n$. The task for sequence γ is: given $D_n^{(\gamma)}$ and $X_{n+1}^{(\gamma)}$, predict $Y_{n+1}^{(\gamma)}$.

▼ How we get the Data

There are Γ sequences, each of length $n+1$, giving $\Gamma(n+1)$ data points in total. Each row is one pretraining sequence:

$$\begin{aligned}
&Sequence\ \gamma = 1 : m^{(1)} \longrightarrow (X_1^{(1)}, Y_1^{(1)}), \ldots, (X_n^{(1)}, Y_n^{(1)}), \quad X_{n+1}^{(1)}, Y_{n+1}^{(1)} \\
&Sequence\ \gamma = 2 : m^{(2)} \longrightarrow (X_1^{(2)}, Y_1^{(2)}), \ldots, (X_n^{(2)}, Y_n^{(2)}), \quad X_{n+1}^{(2)}, Y_{n+1}^{(2)} \\
&\qquad\qquad \vdots \\
&Sequence\ \gamma = \Gamma : m^{(\Gamma)} \longrightarrow (X_1^{(\Gamma)}, Y_1^{(\Gamma)}), \ldots, (X_n^{(\Gamma)}, Y_n^{(\Gamma)}), \quad X_{n+1}^{(\Gamma)}, Y_{n+1}^{(\Gamma)}
\end{aligned}$$

▼ How the Transformer Is Used

The transformer TF_θ is **not** run on one data point at a time. It is **not** doing gradient descent at test time. It is **not** a sequential model that processes $X_1, X_2, \ldots$ one by one.

Instead, for a single sequence γ, the transformer takes **all n context pairs plus the query input simultaneously** as one single input matrix and outputs a single prediction. All n context pairs are packed into the first n rows. The query $X_{n+1}^{(\gamma)}$ is packed into the last row with a 0 in the response column (because $Y_{n+1}^{(\gamma)}$ is unknown) and a 1 in the flag column. The transformer processes this entire matrix at once: $\mathrm{TF}_\theta(Z^{(\gamma)}) \in \mathbb{R}^{(n+1)\times d_e}$. The attention mechanism allows every row to interact with every other row simultaneously — this is how the transformer sees all n context pairs when making its prediction, not sequentially but all at once through the attention scores.

The prediction is then read out as

$$\hat{f}_\Gamma(D_n^{(\gamma)}, X_{n+1}^{(\gamma)}) = \mathrm{Read}_{M,d}(\mathrm{TF}_\theta(Z^{(\gamma)})) = (-M) \vee [\mathrm{TF}_\theta(Z^{(\gamma)})]_{n+1,\,d+1} \wedge M$$

One forward pass through the transformer, one prediction. No gradient descent at test time.

▼ The Empirical Risk Minimisation: How the Transformer Is Trained

Gradient descent happens only during training, not at test time.

During training, the transformer parameters θ are chosen to minimise the empirical risk over all Γ pretraining sequences simultaneously:

$$\hat{f}_\Gamma \in \underset{f \in \mathcal{F}(d_e, d_{\text{ffn}}, L, B, M)}{\operatorname{argmin}} \hat{R}_\Gamma(f),$$

where the empirical risk is

$$\hat{R}_\Gamma(f) := \frac{1}{\Gamma} \sum_{\gamma=1}^{\Gamma} \left(Y_{n+1}^{(\gamma)} - f(D_n^{(\gamma)}, X_{n+1}^{(\gamma)}) \right)^2.$$

This says: for each pretraining sequence γ, run the transformer on the full context $D_n^{(\gamma)}$ and query $X_{n+1}^{(\gamma)}$, get a prediction, compare it to the true label $Y_{n+1}^{(\gamma)}$, square the error, and average across all Γ sequences.

In practice this minimisation is done by gradient descent on θ (e.g. Adam, AdamW) — but this is training-time gradient descent on the **parameters** θ, not test-time gradient descent on the data.

▼ Why $R(\hat{f}_\Gamma)$ Is Random Even Though $R(\cdot)$ Involves an Expectation

This is a subtle point that must be understood precisely.

$R(\cdot)$ is a **functional** — it takes any function f as input and returns a number. For any fixed deterministic f, say $f^\star$: $R(f^\star) = \mathbb{E}_{\text{test}}\left[(Y_{n+1} - f^\star(D_n, X_{n+1}))^2\right]$ is a fixed deterministic number. The expectation over the test sequence has been taken, leaving no randomness.

But $\hat{f}_\Gamma$ is **not** a fixed deterministic function. Its weights $\hat{\theta}$ are chosen by minimizing $\hat{R}_\Gamma$ over the random training data. Different training runs give different $\hat{\theta}$, hence different $\hat{f}_\Gamma$. So when we plug $\hat{f}_\Gamma$ into $R(\cdot)$: $R(\hat{f}_\Gamma) = \mathbb{E}_{\text{test}}\left[(Y_{n+1} - \hat{f}_\Gamma(D_n, X_{n+1}))^2\right]$, the inner expectation over the test sequence is taken with $\hat{f}_\Gamma$ held fixed at whatever training produced. But $\hat{f}_\Gamma$ itself is still random — it varies across training runs. So $R(\hat{f}_\Gamma)$ is random: it is a deterministic functional $R(\cdot)$ evaluated at a random input $\hat{f}_\Gamma$.

$$\mathbb{E}[R(\hat{f}_\Gamma)] = \mathbb{E}_{\text{train}}\left[\mathbb{E}_{\text{test}}\left[(Y_{n+1} - \hat{f}_\Gamma(D_n, X_{n+1}))^2\right]\right]$$

- **Inner** $\mathbb{E}_{\text{test}}$: for a fixed realisation of the training data (hence fixed $\hat{f}_\Gamma$), average over fresh test sequences. This gives $R(\hat{f}_\Gamma)$, which still depends on which training data we get.
- **Outer** $\mathbb{E}_{\text{train}}$: average over all possible training datasets. The result is a single deterministic number. This is what Theorem 3.2 bounds.

▼ The Paper's Main Theorem (Theorem 3.2)

The paper proves: there exists a transformer class $\mathcal{F}(d_e, d_{\text{ffn}}, L, B, M)$ with $L = \Theta(\log n)$ layers such that the ERM $\hat{f}_\Gamma$, trained on $\Gamma \geq Cn^{2\alpha/(2\alpha+d)} \log^3(en)$ pretraining sequences, satisfies

$$\mathbb{E}_{\text{train}}\left[\mathbb{E}_{\text{test}}\left[(Y_{n+1} - \hat{f}_\Gamma(D_n, X_{n+1}))^2\right]\right] - \sigma^2 \leq Cn^{-2\alpha/(2\alpha+d)},$$

for all $\alpha > 0$ and all $d \in \mathbb{N}$.

The rate $n^{-2\alpha/(2\alpha+d)}$ is the **minimax lower bound** — for any estimator $\hat{f}$ whatsoever, there exists some $m \in \mathcal{H}(d, \alpha, M)$ such that $R(\hat{f}) - \sigma^2 \geq c \cdot n^{-2\alpha/(2\alpha+d)}$ for an absolute constant $c > 0$.

No estimator can beat this rate in the worst case. The paper shows the transformer matches it — this is what minimax optimal means.

▼ The Three Obstacles to Proving This

To prove $\mathbb{E}[R(\hat{f}_\Gamma)] - \sigma^2 \leq Cn^{-2\alpha/(2\alpha+d)}$, three things must be true simultaneously.

Obstacle 1 — Approximation:
→ The transformer class $\mathcal{F}$ must contain a function that is actually good. The paper gives an explicit function that lies in F and does this
→ Even if we train perfectly, if no $f \in \mathcal{F}$ achieves the minimax rate, the bound fails.
→ This requires constructing an explicit $f^\star \in \mathcal{F}$ with $R(f^\star) - \sigma^2 \leq Cn^{-2\alpha/(2\alpha+d)}$.

Obstacle 2 — Generalisation:
→ The ERM $\hat{f}_\Gamma$ must actually find a good function, not just memorise training data. Even if $f^\star \in \mathcal{F}$ exists, the ERM minimises $\hat{R}_\Gamma$, not R.
→ It could overfit — achieving small $\hat{R}_\Gamma$ but large R.
→ We need $\hat{R}_\Gamma$ to be close to R uniformly over all $f \in \mathcal{F}$ simultaneously, so no function can appear misleadingly good on training data.

Obstacle 3 — Efficiency:
→ Both obstacles must be resolved with as few parameters and as few pretraining sequences Γ as possible.
→ **This is the paper's main novelty. Prior work resolved Obstacles 1 and 2 but needed polynomially many parameters in n.**

Why "fixed before seeing the data" matters for concentration. For $f^\star$ fixed and deterministic, the training losses $\ell^{(\gamma)}_{f^\star} = (Y^{(\gamma)}_{n+1} - f^\star(D^{(\gamma)}_n, X^{(\gamma)}_{n+1}))^2$ are i.i.d. across γ, because the training sequences are i.i.d. and $f^\star$ does not depend on any of them. So $\mathbb{E}_{\text{train}}[\hat{R}_\Gamma(f^\star)] = R(f^\star)$ exactly. For $\hat{f}_\Gamma$, this completely breaks down — $\hat{f}_\Gamma$ was trained on sequence γ, so $\ell^{(\gamma)}_{\hat{f}_\Gamma}$ is not independent of $D^{(\gamma)}_n$.

▼ Why Knowing $R(f^\star) - \sigma^2 \leq Cn^{-2\alpha/(2\alpha+d)}$ Is Not Enough

$R(f^\star) - \sigma^2 \le Cn^{-2\alpha/(2\alpha+d)}$ says the hand-constructed transformer $f^\star$ achieves the minimax rate. But $f^\star$ is not what gets deployed — $\hat{f}_\Gamma$ is.

Concrete Example:
Suppose $\mathcal{F}$ contains two functions: $f^\star$ with $R(f^\star) = \sigma^2 + 0.001$ (very good), and f^{bad} with $R(f^{\text{bad}}) = \sigma^2 + 100$ (terrible). Suppose by bad luck the training data makes $\hat{R}_\Gamma(f^{\text{bad}}) < \hat{R}_\Gamma(f^\star)$ — f^{bad} looks better on training sequences even though it is much worse in truth. Then the ERM picks $\hat{f}_\Gamma = f^{\text{bad}}$, and $R(\hat{f}_\Gamma) = \sigma^2 + 100$. Knowing $R(f^\star) - \sigma^2 = 0.001$ did not help at all.

Uniform concentration requirement:

→ For this bad event to be unlikely, we need $\hat{R}_\Gamma$ to be close to R for all $f \in \mathcal{F}$ simultaneously — so no bad function can look artificially good on training data. This is the **uniform concentration** requirement.

▼ What Uniform Concentration Means

For a single fixed $f^\star$, concentration means $\hat{R}_\Gamma(f^\star) \approx R(f^\star)$ with high probability as $\Gamma \to \infty$. By Hoeffding's inequality (requiring $|\ell_{f^\star}^{(\gamma)}| \le (2M+1)^2$):

$$\mathbb{P}\Big(|\hat{R}_\Gamma(f^\star) - R(f^\star)| > t\Big) \le 2\exp\Big(-\tfrac{2\Gamma t^2}{(2M+1)^4}\Big).$$

Uniform concentration means the same holds for every $f \in \mathcal{F}$ simultaneously: $\sup_{f\in\mathcal{F}} |\hat{R}_\Gamma(f) - R(f)| \le \varepsilon$ with high probability. In words: the training data gives an honest picture of the true quality of every function in $\mathcal{F}$, all at once.

Why this implies the ERM works. If uniform concentration holds with slack ε:

$$R(\hat{f}_\Gamma) \le \hat{R}_\Gamma(\hat{f}_\Gamma) + \varepsilon \le \hat{R}_\Gamma(f^\star) + \varepsilon \le R(f^\star) + 2\varepsilon.$$

- Step 1: uniform concentration applied to $\hat{f}_\Gamma$.
- Step 2: $\hat{f}_\Gamma$ minimises $\hat{R}_\Gamma$, so $\hat{R}_\Gamma(\hat{f}_\Gamma) \le \hat{R}_\Gamma(f^\star)$, since $\hat{f}_\Gamma = \operatorname{argmin}_{f\in\mathcal{F}} \hat{R}_\Gamma(f)$ and $f^\star \in \mathcal{F}$.
- Step 3: uniform concentration applied to $f^\star$.

So the ERM is at most 2ε worse than $f^\star$ in population risk.

▼ The Full Decomposition — How Everything Combines

$$\mathbb{E}[R(\hat{f}_\Gamma)] - \sigma^2 \le \underbrace{\mathbb{E}\left[\sup_{f\in\mathcal{F}} |R(f) - \hat{R}_\Gamma(f)|\right]}_{\text{(A): Obstacle 2 — generalisation}} + \underbrace{\mathbb{E}[\hat{R}_\Gamma(f^\star) - R(f^\star)]}_{=0} + \underbrace{R(f^\star) - \sigma^2}_{\text{(D): Obstacle 1 — approximation}}$$

Where each expectation is over:

Term	Expectation over	Why
$\mathbb{E}[R(\hat{f}_\Gamma)]$	Training data	$\hat{f}_\Gamma$ is random — depends on training data
Term (A)	Training data	$\hat{R}_\Gamma(f)$ is random — depends on training data
Term (B)	Training data	$\hat{R}_\Gamma(f^\star)$ is random; equals zero in expectation because $f^\star$ is fixed and unbiased
Term (D)	None	$f^\star$, $R(f^\star)$, σ^2 are all deterministic

▼ Why term (B) is exactly zero in expectation

Since $f^\star$ is fixed and each training sequence is an independent draw from the same distribution:

$$\mathbb{E}_{\text{train}}[\hat{R}_\Gamma(f^\star)] = \tfrac{1}{\Gamma}\textstyle\sum_{\gamma=1}^{\Gamma} \mathbb{E}_{\text{train}}\Big[(Y_{n+1}^{(\gamma)} - f^\star(D_n^{(\gamma)}, X_{n+1}^{(\gamma)}))^2\Big] = R(f^\star).$$

So $\mathbb{E}[\hat{R}_\Gamma(f^\star) - R(f^\star)] = 0$ exactly.

Bounding each remaining term:

- **Term (A)** $= O(n^{-2\alpha/(2\alpha+d)})$:
  - by Lemma C.4 (covering numbers of $\mathcal{F}$) and
  - Theorem D.1 (ERM risk bound), provided $\Gamma \ge Cn^{2\alpha/(2\alpha+d)}\log^3 n$.

- **Term (D)** $= O(n^{-2\alpha/(2\alpha+d)})$:
  - by Theorem 3.1 ($f^\star$ approximates f_{LocPol} to within C/n) and
  - Theorem 2.5 (f_{LocPol} achieves the minimax rate).

Both terms are $O(n^{-2\alpha/(2\alpha+d)})$, completing the proof of Theorem 3.2.

▼ Appendix B

▼ Lemma B.7

▼ Importance of B.7

Refer to section:

- **Obstacle 1 (Approximation):** Construct an explicit $f^* \in \mathcal{F}$ with $R(f^*) - \sigma^2 \leq Cn^{-2\alpha/(2\alpha+d)}$. This is Term D.
- **Obstacle 2 (Generalisation):** Show the ERM $\hat{f}_\Gamma$ is not much worse than f^*. This is Term A.
- **Obstacle 3 (Efficiency):** Do both with only $\Theta(\log n)$ parameters.

Lemma B.7 is important for **Obstacle 1**. Specifically, it is the very first step in constructing the explicit transformer f^* — called f_{TF} in the paper — that approximates the local polynomial estimator.

The chain is:

$$\underbrace{\text{Lemma B.7}}_{\text{Stage 1: kernel + centred covariates}} \rightarrow \underbrace{\text{Lemma B.8}}_{\text{Stage 2: approximate } \tilde{X},\tilde{Y}} \rightarrow \underbrace{\text{Lemmas B.9, B.10}}_{\text{Stage 3: gradient descent}} \rightarrow \underbrace{\text{Proposition B.11}}_{\text{transformer} \approx f_{\mathrm{LocPol}}} \rightarrow \underbrace{\text{Theorem 3.1}}_{R(f_{\mathrm{TF}}) \approx R(f_{\mathrm{LocPol}})} \rightarrow \underbrace{\text{Theorem 3.2}}_{\text{main result}}$$

Recall: the local polynomial estimator needs $\tilde{X}$ and $\tilde{Y}$

$\tilde{X}_{i\nu} = \frac{1}{\sqrt{n}} \sqrt{K_h(X_i - X_{n+1})} \cdot \frac{(X_i - X_{n+1})^\nu}{\nu!\, h^{|\nu|}}$

$\tilde{Y}_i = \frac{1}{\sqrt{n}} \sqrt{K_h(X_i - X_{n+1})} \cdot Y_i$

Every entry of $\tilde{X}$ and $\tilde{Y}$ is a product of two ingredients:

- The **centered covariate** $\frac{X_i - X_{n+1}}{h}$ — how far training point i is from the query, scaled by bandwidth.
- The **square root kernel weight** $\sqrt{K_h(X_i - X_{n+1})}$ — how much weight to give training point i.

Before we can compute $\tilde{X}$ and $\tilde{Y}$ inside the transformer, we first need these two ingredients available in the token rows. Lemma B.7 proves that **3 transformer blocks suffice to compute both the parts exactly and store them in the token matrix in the gaps provided (The zero columns)**

▼ Lemma B.7 — Formal Statement

Given

Let $X_1, \ldots, X_n \in [0,1]^d$

$Y_1, \ldots, Y_n \in \mathbb{R}$,

$X_{n+1} \in [0,1]^d$,

bandwidth $h > 0$.

Using the specific kernel $K(x) = (1 - \|x\|_1)_+^2$.

Define:

- $K_h(X, X_{n+1}) \in \mathbb{R}^n$ (i-th entry is $K_h(X_i - X_{n+1})$ — the kernel weights)
- $\sqrt{K_h(X, X_{n+1})} \in \mathbb{R}^n$ — entrywise square root of the above

Lemma Statement:

Let $d_e \geq 2d + 4$ and $d_{\mathrm{ffn}} \geq 2d + 2$.

Set $B = 1 \vee (d/h) \vee (n^{-1/2} h^{-d/2})$

Then there exists a transformer $\mathrm{TF} \in \mathcal{T}(d_e, d_{\mathrm{ffn}}, 3, B)$ — with 3 blocks and weights bounded by B — such that:

$$\mathrm{TF} \circ \mathrm{Embed}_{d_e}\left((X_i, Y_i)_{i=1}^n, X_{n+1}\right) = \begin{pmatrix} X & Y & \frac{X - \mathbf{1}_n X_{n+1}^T}{h} & \frac{\sqrt{K_h(X, X_{n+1})}}{\sqrt{n}} & 0_{n \times (d_e - 2d - 4)} & \mathbf{1}_n & 0_n \\ X_{n+1}^T & 0 & 0_d^T & n^{-1/2} h^{-d/2} & 0_{d_e - 2d - 4}^T & 1 & 1 \end{pmatrix}$$

▼ Proof of Lemma B.7

▼ Need to Prove:

there exist explicit weight matrices for 3 transformer blocks such that when we pass the embedded data through them, every training row i ends up containing $\frac{X_i - X_{n+1}}{h}$ and $\frac{\sqrt{K_h(X_i - X_{n+1})}}{\sqrt{n}}$ in designated columns, exactly, with no approximation error.

→ The proof constructs the weight matrices explicitly and verifies the output.

▼ Given:

$$Z^{(0)} = \mathrm{Embed}_{d_e}\left((X_i, Y_i)_{i=1}^n, X_{n+1}\right) = \begin{pmatrix} X_1^T & Y_1 & 0_{d_e - d - 2}^T & 0 \\ X_2^T & Y_2 & 0_{d_e - d - 2}^T & 0 \\ \vdots & \vdots & \vdots & \vdots \\ X_n^T & Y_n & 0_{d_e - d - 2}^T & 0 \\ X_{n+1}^T & 0 & 0_{d_e - d - 2}^T & 1 \end{pmatrix} \in \mathbb{R}^{(n+1) \times d_e}$$

Label the columns as slots 1 through d_e:

- Slots 1 to d: the covariate X_i (or X_{n+1} for the query row)
- Slot $d + 1$: the label Y_i (zero for query row)
- Slots $d + 2$ to $d_e - 1$: all zeros — empty space for future computation
- Slot d_e: the positional flag — 0 for training rows, 1 for the query row

The requirement $d_e \geq 2d + 4$ ensures there are enough empty slots for everything we need to write.

Starting Point: $Z^{(0)}$:

The input after embedding is:

$$Z_{\text{in}} = \begin{pmatrix} X^T & Y & 0_{n \times (d_e - d - 2)} & 0_n \\ X_{n+1}^T & 0 & 0_{d_e - d - 2}^T & 1 \end{pmatrix} \in \mathbb{R}^{(n+1) \times d_e}$$

where $X \in \mathbb{R}^{n \times d}$ has rows $X_1^T, \ldots, X_n^T$, and $Y = (Y_1, \ldots, Y_n)^T$.

▼ Block 1:

Attention layer:

$\mathrm{Attn}^{(1)}$ **is the identity.** The paper sets $Q^{(1)} = K^{(1)} = V^{(1)} = 0$, so $\mathrm{Attn}^{(1)}(Z_{\text{in}}) = Z_{\text{in}}$
Attention does nothing in Block 1

FFN layer: $\mathrm{FFN}^{(1)}$

The paper sets:

$$b_1^{(1)} = -\mathbf{1}_{d_{\text{ffn}}} \in \mathbb{R}^{d_{\text{ffn}}}$$

$$W_1^{(1)} = \begin{pmatrix} I_{d \times d} & 0_{d \times (d_e - d - 1)} & \mathbf{1}_d \\ 0_{(d_{\text{ffn}} - d) \times d} & 0_{(d_{\text{ffn}} - d) \times (d_e - d - 1)} & 0_{d_{\text{ffn}} - d} \end{pmatrix} \in \mathbb{R}^{d_{\text{ffn}} \times d_e}$$

$$W_2^{(1)} = \begin{pmatrix} 0_{(d+1) \times d} & 0_{(d+1) \times (d_{\text{ffn}} - d)} \\ I_{d \times d} & 0_{d \times (d_{\text{ffn}} - d)} \\ 0_{(d_e - 2d - 1) \times d} & 0_{(d_e - 2d - 1) \times (d_{\text{ffn}} - d)} \end{pmatrix} \in \mathbb{R}^{d_e \times d_{\text{ffn}}}$$

$$b_2^{(1)} = (0_{d+1}^T,\ \mathbf{0}_d^T,\ 0_{d_e - 2d - 2},\ 0,\ 1,\ 0)^T$$

What this FFN computes. For any row $Z_{i,\cdot} = (x^T, y, 0 \ldots, 0, f)$ where $x \in \mathbb{R}^d$, $y \in \mathbb{R}$, and f belongs to [0,1]

$$W_1^{(1)} Z_{i,\cdot}^T + b_1^{(1)} = \begin{pmatrix} x + (f - 1)\mathbf{1}_d \\ -\mathbf{1}_{d_{\text{ffn}} - d} \end{pmatrix}$$

After ReLU:

$$\mathrm{ReLU}\big(W_1^{(1)} Z_{i,\cdot}^T + b_1^{(1)}\big) = \begin{pmatrix} \mathrm{ReLU}(x + (f-1)\mathbf{1}_d) \\ 0_{d_{\text{ffn}}-d} \end{pmatrix}$$

For the **query row**: $f = 1$, so this is $\mathrm{ReLU}(X_{n+1} + 0) = X_{n+1}$ (since $X_{n+1} \in [0,1]^d \geq 0$).

- **Training row**: $f = 0$, so this is $\mathrm{ReLU}(x - \mathbf{1}_d)$. Since $x = X_i \in [0,1]^d$, each coordinate $X_{i,j} \leq 1$, so $X_{i,j} - 1 \leq 0$, meaning $\mathrm{ReLU}(X_i - \mathbf{1}_d) = 0_d$

$$\mathrm{ReLU}(\cdots)\big|_{\text{train}} = 0_{d_{\text{ffn}}}$$

- **Query row** ($f = 1$, $x = X_{n+1} \in [0,1]^d$): top block is $\mathrm{ReLU}(X_{n+1} + 0) = X_{n+1}$ (all entries ≥ 0). Bottom: 0

$$\mathrm{ReLU}(\cdots)\big|_{\text{query}} = \begin{pmatrix} X_{n+1} \\ 0_{d_{\text{ffn}}-d} \end{pmatrix}$$

So across all $n+1$ rows, the ReLU output matrix is:

$$R_1 := \mathrm{ReLU}\Big(W_1^{(1)} Z_{\text{in}}^T + b_1^{(1)} \mathbf{1}_{n+1}^T\Big) = \begin{pmatrix} 0_d & \cdots & 0_d & X_{n+1} \\ 0_{d_{\text{ffn}}-d} & \cdots & 0_{d_{\text{ffn}}-d} & 0_{d_{\text{ffn}}-d} \end{pmatrix} \in \mathbb{R}^{d_{\text{ffn}} \times (n+1)}$$

Only the last column (query) is nonzero.

Second linear + bias ($d_e \times d_{\text{ffn}}$):

$$W_2^{(1)} = \begin{pmatrix} 0_{(d+1)\times d} & 0_{(d+1)\times(d_{\text{ffn}}-d)} \\ I_{d\times d} & 0_{d\times(d_{\text{ffn}}-d)} \\ 0_{(d_e-2d-1)\times d} & 0 \end{pmatrix}, \qquad b_2^{(1)} = (0_{d_e-2}^T,\ 1,\ 0)^T$$

$$W_2^{(1)} R_1 = \begin{pmatrix} 0_{(d+1)\times n} & 0_{d+1} \\ 0_{d\times n} & X_{n+1} \\ 0_{(d_e-2d-1)\times n} & 0_{d_e-2d-1} \end{pmatrix} \in \mathbb{R}^{d_e \times (n+1)}$$

Adding $b_2^{(1)} \mathbf{1}_{n+1}^T$ puts a 1 in the $(d_e - 1)$-th row of every column. Transposing and adding to Z_{in}:

$$Z^{(1)} = \begin{pmatrix} X_1^T & Y_1 & 0_d^T & 0 \cdots 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_n^T & Y_n & 0_d^T & 0 \cdots 0 & 1 & 0 \\ X_{n+1}^T & 0 & X_{n+1}^T & 0 \cdots 0 & 1 & 1 \end{pmatrix}$$

Result

$$Z^{(1)} = \begin{pmatrix} X & Y & 0_{n\times d} & 0_{n\times(d_e-2d-3)} & \mathbf{1}_n & 0_n \\ X_{n+1}^T & 0 & X_{n+1}^T & 0_{d_e-2d-3}^T & 1 & 1 \end{pmatrix} \in \mathbb{R}^{(n+1)\times d_e}.$$

The end goal of B.7 is to take the raw embedded data (X_i, Y_i) and produce, using attention + FFN layers only, two derived quantities needed for local polynomial regression:

1. The **centred covariates** $(X_i - X_{n+1})/h$ — how far each context point is from the query point, in each dimension.
2. The **kernel weights** $\sqrt{K_h(X_i - X_{n+1})}$ — how much each context point should count, based on its distance to the query.

Both require knowing X_{n+1} (the query point) at every row — but in the raw embedding, X_{n+1} only lives in the very last row (the query row). The context rows ($i \leq n$) have no idea what X_{n+1} is.

What Block 1 does: it's the broadcasting step. It takes X_{n+1}, which only appears in row $n+1$, and **copies it into a brand-new column-block that appears in every row** — including the context rows. Look at the result:

$$Z^{(1)} = \begin{pmatrix} X & Y & 0_{n\times d} & \cdots & \mathbf{1}_n & 0_n \\ X_{n+1}^T & 0 & X_{n+1}^T & \cdots & 1 & 1 \end{pmatrix}$$

After Block 1, every row — context and query alike — now has X_{n+1} available in slots $d+2$ to $2d+1$. The context rows can now compute $X_i - X_{n+1}$ in Block 2.

▼ Block 2

Attention Layer $\mathrm{Attn}^{(2)}$

We now compute the attention layer of Block 2, $\mathrm{Attn}^{(2)}$, applied to $Z^{(1)}$ (the output from Block 1).
The matrices

$$Q^{(2)} := \frac{1}{\sqrt{d_e}} \begin{pmatrix} 0_{(d_e-2)\times d_e} \\ \mathbf{1}_{d_e}^T \\ -\mathbf{1}_{d_e}^T \end{pmatrix}, \quad K^{(2)} := \frac{1}{\sqrt{d_e}} \begin{pmatrix} 0_{(d_e-2)\times d_e} \\ \mathbf{1}_{d_e}^T \\ 0_{d_e}^T \end{pmatrix}, \quad V^{(2)} := \begin{pmatrix} 0_{(d+1)\times(d+1)} & 0_{(d+1)\times d} & 0 \\ 0_{d\times(d+1)} & I_{d\times d} & 0 \\ 0 & 0 & 0_{(d_e-2d-1)\times(d_e-2d-1)} \end{pmatrix}$$

all in $\mathbb{R}^{d_e\times d_e}$
Recall $\mathrm{Attn}(Z) = Z + ZQ(ZK)^T ZV$ (Definition 2.1).

→ **Calculating** $Z^{(1)}Q^{(2)}$

$Q^{(2)}$ is zero **except its last two rows**, which are the constant row vectors $\mathbf{1}_{d_e}^T/\sqrt{d_e}$ and $-\mathbf{1}_{d_e}^T/\sqrt{d_e}$.

$$(Z^{(1)}Q^{(2)})_{i,j} = \frac{Z^{(1)}_{i,d_e-1} - Z^{(1)}_{i,d_e}}{\sqrt{d_e}} \quad (\text{same for every } j!)$$

- **Context row** $i \le n$**:** $1-0=1 \Rightarrow$ row i of $Z^{(1)}Q^{(2)}$ is $\frac{1}{\sqrt{d_e}}\mathbf{1}_{d_e}^T$.
- **Query row** $n+1$**:** $1-1=0 \Rightarrow$ row $n+1$ is $0_{d_e}^T$.

$$Z^{(1)}Q^{(2)} = \frac{1}{\sqrt{d_e}} \begin{pmatrix} \mathbf{1}_n \mathbf{1}_{d_e}^T \\ 0_{d_e}^T \end{pmatrix} \in \mathbb{R}^{(n+1)\times d_e}.$$

- rows $1,\dots,n$: constant row $\frac{1}{\sqrt{d_e}}(1,1,\dots,1)$ (d_e entries),
- row $n+1$ is all zeros.

$Z^{(1)}K^{(2)}$ **and Its Transpose**

$K^{(2)}$ is zero except row d_e-1, which equals $\mathbf{1}_{d_e}^T/\sqrt{d_e}$. $\Rightarrow (Z^{(1)}K^{(2)})_{i,j} = Z^{(1)}_{i,d_e-1}/\sqrt{d_e}$, for every j
Since column d_e-1 is **all ones for every row** (context and query):
$Z^{(1)}K^{(2)} = \frac{1}{\sqrt{d_e}}\mathbf{1}_{n+1}\mathbf{1}_{d_e}^T \quad \Longrightarrow \quad (Z^{(1)}K^{(2)})^T = \frac{1}{\sqrt{d_e}}\mathbf{1}_{d_e}\mathbf{1}_{n+1}^T \in \mathbb{R}^{d_e\times(n+1)}$.
d_e rows and $n+1$ columns, every entry equal to $\frac{1}{\sqrt{d_e}}$

→ **Calculating** $Z^{(1)}Q^{(2)}\,(Z^{(1)}K^{(2)})^T$

$$Z^{(1)}Q^{(2)}(Z^{(1)}K^{(2)})^T = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \end{pmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}$$

rows 1,2,3,..... n are all-ones: $\frac{1}{\sqrt{d_e}} \cdot \frac{1}{\sqrt{d_e}} \cdot d_e = 1$ since each dot product sums d_e terms of $\frac{1}{\sqrt{d_e}} \cdot \frac{1}{\sqrt{d_e}}$.
row $n+1$ is all-zeros

→ $Z^{(1)}V^{(2)}$

$$Z^{(1)}V^{(2)} = \begin{pmatrix} 0_{n\times(d+1)} & 0_{n\times d} & 0_{n\times(d_e-2d-1)} \\ 0_{d+1}^T & X_{n+1}^T & 0_{d_e-2d-1}^T \end{pmatrix} \in \mathbb{R}^{(n+1)\times d_e}$$

Every row is entirely zero except the last row ($n+1$), which has $X_{n+1}^T = (x_1,\dots,x_d)$ in columns $d+2$ through $2d+1$, and zero everywhere else.

→ **Calc** $Z^{(1)}Q^{(2)}(Z^{(1)}K^{(2)})^T Z^{(1)}V^{(2)}$

$$Z^{(1)}Q^{(2)}(Z^{(1)}K^{(2)})^T Z^{(1)}V^{(2)} = \begin{pmatrix} 0_{n\times(d+1)} & \mathbf{1}_n X_{n+1}^T & 0_{n\times(d_e-2d-1)} \\ 0_{d_e}^T & & \end{pmatrix} \in \mathbb{R}^{(n+1)\times d_e}$$

Rows $1,\dots,n$ all carry $X_{n+1}^T = (x_1,\dots,x_d)$ in columns $d+2,\dots,2d+1$ — every context row now has a copy of the query point. Row $n+1$ is all zeros

---

FFN Layer

Input to $\mathrm{FFN}^{(2)}$ is $Z^{(2)}_{\mathrm{attn}} := \mathrm{Attn}^{(2)}(Z^{(1)})$:

$$Z^{(2)}_{\text{attn}} = \begin{pmatrix} X & Y & \mathbf{1}_n X_{n+1}^T & 0_{n\times(d_e-2d-3)} & \mathbf{1}_n & 0_n \\ X_{n+1}^T & 0 & X_{n+1}^T & 0_{d_e-2d-3}^T & 1 & 1 \end{pmatrix} \in \mathbb{R}^{(n+1)\times d_e}.$$

Goal of this FFN: now that every row has both X_i (columns $1..d$) and a broadcast copy of X_{n+1} (columns $d+2..2d+1$), compute the signed difference $(X_i - X_{n+1})/h$ per coordinate, and simultaneously build a running total that will become the kernel weight $\sqrt{K_h(X_i - X_{n+1})} \approx 1 - \|X_i - X_{n+1}\|_1/h$.

First Linear Layer: $W_1^{(2)}$, $b_1^{(2)}$

$$W_1^{(2)} = \begin{pmatrix} I_{d\times d}/h & 0_d & -I_{d\times d}/h & 0_{d\times(d_e-2d-1)} \\ -I_{d\times d}/h & 0_d & I_{d\times d}/h & 0_{d\times(d_e-2d-1)} \\ 0_{(d_{\text{ffn}}-2d)\times d} & 0_{d_{\text{ffn}}-2d} & 0_{(d_{\text{ffn}}-2d)\times d} & 0_{(d_{\text{ffn}}-2d)\times(d_e-2d-1)} \end{pmatrix} \in \mathbb{R}^{d_{\text{ffn}}\times d_e}, \quad b_1^{(2)} = 0.$$

Rows $1..d$ read column-block $1..d$ (coefficient $+I/h$)
column-block $d+2..2d+1$ (coefficient $-I/h$), ignoring everything else.
Rows $d+1..2d$ do the same with flipped sign.
So for any row $z = (x^T, y, x_{n+1}^T, \dots)$ of $Z^{(2)}_{\text{attn}}$:

$$W_1^{(2)} z = \begin{pmatrix} (x - x_{n+1})/h \\ (x_{n+1} - x)/h \\ 0_{d_{\text{ffn}}-2d} \end{pmatrix}.$$

Applying to every token and taking ReLU:

- **Context columns** $i \le n$**:** $x = X_i$, $x_{n+1} = X_{n+1}$, so top d rows become $\mathrm{ReLU}((X_i - X_{n+1})/h)$ and
- next d rows become $\mathrm{ReLU}((X_{n+1} - X_i)/h)$ t
  - Why this: The positive and negative parts of the same difference, split apart (the standard $\mathrm{ReLU}(a), \mathrm{ReLU}(-a)$ trick for recovering $|a|$ or a later).
- **Query column** $n+1$**:** $x = x_{n+1} = X_{n+1}$ in both blocks, so the difference is exactly 0 — giving 0_d in both blocks.

$$R_2 := \mathrm{ReLU}\big(W_1^{(2)} (Z^{(2)}_{\text{attn}})^T\big) = \begin{pmatrix} \mathrm{ReLU}(X^T - X_{n+1}\mathbf{1}_n^T)/h & 0_d \\ \mathrm{ReLU}(X_{n+1}\mathbf{1}_n^T - X^T)/h & 0_d \\ 0_{(d_{\text{ffn}}-2d)\times n} & 0_{d_{\text{ffn}}-2d} \end{pmatrix} \in \mathbb{R}^{d_{\text{ffn}}\times(n+1)}.$$

Second linear:

$$W_2^{(2)} = \begin{pmatrix} 0_{(2d+1)\times d} & 0_{(2d+1)\times d} & 0_{(2d+1)\times(d_{\text{ffn}}-2d)} \\ -\mathbf{1}_d^T & -\mathbf{1}_d^T & 0_{d_{\text{ffn}}-2d}^T \\ 0_{(d_e-2d-2)\times d} & 0 & 0 \end{pmatrix}, \qquad b_2^{(2)} = (0_{2d+1}^T,\ 1,\ 0_{d_e-2d-2}^T)^T$$

Row $2d+2$ of $W_2^{(2)} R_2$ is:
$-\mathbf{1}_d^T \cdot \mathrm{ReLU}((X^T - X_{n+1}\mathbf{1}_n^T)/h) - \mathbf{1}_d^T \cdot \mathrm{ReLU}((X_{n+1}\mathbf{1}_n^T - X^T)/h)$
For training column i: this is

$$-\sum_{j=1}^{d} \mathrm{ReLU}\left(\frac{X_{i,j} - X_{n+1,j}}{h}\right) - \sum_{j=1}^{d} \mathrm{ReLU}\left(\frac{X_{n+1,j} - X_{i,j}}{h}\right) = -\left\|\frac{X_i - X_{n+1}}{h}\right\|_1$$

Adding bias $b_2^{(2)}$ contributes $+1$ to row $2d+2$, giving:

$$\tilde{K}_i := 1 - \left\|\frac{X_i - X_{n+1}}{h}\right\|_1$$

For the query column: everything is zero, bias gives $+1$, so $\tilde{K}_{n+1} = 1$.
Adding the FFN update to the attention output:

$$\boxed{Z^{(2)} = \begin{pmatrix} X & Y & \mathbf{1}_n X_{n+1}^T & \tilde{K} & 0_{n\times(d_e-2d-4)} & \mathbf{1}_n & 0_n \\ X_{n+1}^T & 0 & X_{n+1}^T & 1 & 0_{d_e-2d-4}^T & 1 & 1 \end{pmatrix}}$$

where $\tilde{K} \in \mathbb{R}^n$ has i-th entry $\tilde{K}_i = 1 - \|(X_i - X_{n+1})/h\|_1$.
Note: $\tilde{K}_i$ is not yet the kernel weight.
The actual kernel is $K(x) = (1 - \|x\|_1)_+^2$,
so $\sqrt{K_h(X_i, X_{n+1})} = \sqrt{(1 - \|(X_i - X_{n+1})/h\|_1)_+^2} = (1 - \|(X_i - X_{n+1})/h\|_1)_+ = \mathrm{ReLU}(\tilde{K}_i)$. Block 3 will apply the ReLU.

▼ Block 3

Attention Layer:

identity ($Q^{(3)} = K^{(3)} = V^{(3)} = 0$), so the input is $Z^{(2)}$ unchanged

FFN layer

First linear, $W_1^{(3)} Z^{(2)T} + b_1^{(3)} \mathbf{1}_{n+1}^T$

$$W_1^{(3)} = \begin{pmatrix} 0_{2\times d} & 0_2 & 0_{2\times d} & \mathbf{1}_2 & 0_{2\times(d_e-2d-2)} \\ I_{d\times d}/h & 0_d & -I_{d\times d}/h & 0_d & 0_{d\times(d_e-2d-2)} \\ 0_{d\times d} & 0_d & I_{d\times d} & 0_d & 0_{d\times(d_e-2d-2)} \\ 0_{(d_{\text{ffn}}-2d-2)\times d} & 0_{d_{\text{ffn}}-2d-2} & 0_{(d_{\text{ffn}}-2d-2)\times d} & 0_{d_{\text{ffn}}-2d-2} & 0_{(d_{\text{ffn}}-2d-2)\times(d_e-2d-2)} \end{pmatrix}, \qquad b_1^{(3)} = \begin{pmatrix} 0 \\ d/h \\ \mathbf{1}_d/h \\ 0_d \\ 0_{d_{\text{ffn}}-2d-2} \end{pmatrix}$$

For **training column** i:

$$W_1^{(3)} z_i^T + b_1^{(3)} = \begin{pmatrix} \tilde{K}_i \\ \tilde{K}_i + d/h \\ (X_i - X_{n+1} + \mathbf{1}_d)/h \\ X_{n+1} \\ 0_{d_{\text{ffn}}-2d-2} \end{pmatrix}$$

For the **query column** $n+1$:

$$W_1^{(3)} z_{n+1}^T + b_1^{(3)} = \begin{pmatrix} 1 \\ 1 + d/h \\ \mathbf{1}_d/h \\ X_{n+1} \\ 0_{d_{\text{ffn}}-2d-2} \end{pmatrix}$$

Apply ReLU

Training column i**:**

- **Row 1** — $\mathrm{ReLU}(\tilde{K}_i)$: since $\tilde{K}_i = 1 - \|(X_i - X_{n+1})/h\|_1$ can be negative, ReLU gives $(\tilde{K}_i)_+$
  - Using the kernel identity $K(x) = (1 - \|x\|_1)_+^2 \Rightarrow \sqrt{K_h(X_i, X_{n+1})} = h^{-d/2}(1 - \|(X_i - X_{n+1})/h\|_1)_+$,
  - this equals $h^{d/2}\sqrt{K_h(X_i, X_{n+1})}$.
- **Row 2** — $\mathrm{ReLU}(\tilde{K}_i + d/h)$: since $\|(X_i - X_{n+1})/h\|_1 \le d/h$ always (coordinates in $[0,1]^d$), this is always $\ge 1 > 0$,
  - so ReLU is the identity: stays $\tilde{K}_i + d/h$.
- **Rows 3..d+2** — $\mathrm{ReLU}((X_i - X_{n+1} + \mathbf{1}_d)/h)$: numerator $\in [0,2]^d$, always non-negative,
  - so ReLU is the identity.
- **Rows d+3..2d+2** — $\mathrm{ReLU}(X_{n+1})$: always non-negative, ReLU is the identity.

$$R_3^{(i)} = \begin{pmatrix} h^{d/2}\sqrt{K_h(X_i, X_{n+1})} \\ \tilde{K}_i + d/h \\ (X_i - X_{n+1} + \mathbf{1}_d)/h \\ X_{n+1} \\ 0_{d_{\text{ffn}}-2d-2} \end{pmatrix}$$

Query column $n+1$**:** every pre-ReLU entry above is already non-negative, so $R_3^{(n+1)}$ is unchanged:

$$R_3^{(n+1)} = \begin{pmatrix} 1 \\ 1 + d/h \\ \mathbf{1}_d/h \\ X_{n+1} \\ 0_{d_{\text{ffn}}-2d-2} \end{pmatrix}$$

Second linear, $W_2^{(3)} R_3 + b_2^{(3)} \mathbf{1}_{n+1}^T$

$$W_2^{(3)} = \begin{pmatrix} 0_{d+1} & 0_{d+1} & 0_{(d+1)\times d} & 0_{(d+1)\times d} & 0_{(d+1)\times(d_{\text{ffn}}-2d-2)} \\ 0_d & 0_d & I_{d\times d} & -I_{d\times d} & 0_{d\times(d_{\text{ffn}}-2d-2)} \\ n^{-1/2}h^{-d/2} & -1 & 0_d^T & 0_d^T & 0_{d_{\text{ffn}}-2d-2}^T \\ 0_{d_e-2d-2} & 0_{d_e-2d-2} & 0_{(d_e-2d-2)\times d} & 0_{(d_e-2d-2)\times d} & 0_{(d_e-2d-2)\times(d_{\text{ffn}}-2d-2)} \end{pmatrix}, \qquad b_2^{(3)} = \begin{pmatrix} 0_{d+1} \\ -\mathbf{1}_d/h \\ d/h \\ 0_{d_e-2d-2} \end{pmatrix}$$

Training column i:

Rows $1..d+1$ are zero.
Rows $d+2.....2d+1$ (the $+I, -I$ block):

$$I_{d\times d}\cdot\frac{X_i - X_{n+1} + \mathbf{1}_d}{h} - I_{d\times d}\cdot X_{n+1} - \frac{\mathbf{1}_d}{h} = \frac{X_i - X_{n+1}}{h} - X_{n+1}$$

Row $2d+2$ (scalar row):

$$n^{-1/2}h^{-d/2}\cdot h^{d/2}\sqrt{K_h(X_i, X_{n+1})} - (\tilde{K}_i + d/h) + d/h = n^{-1/2}\sqrt{K_h(X_i, X_{n+1})} - \tilde{K}_i$$

Hence, we get

$$W_2^{(3)}R_3^{(i)} + b_2^{(3)} = \begin{pmatrix} 0_{d+1} \\ \dfrac{X_i - X_{n+1}}{h} - X_{n+1} \\ n^{-1/2}\sqrt{K_h(X_i, X_{n+1})} - \tilde{K}_i \\ 0_{d_e-2d-2} \end{pmatrix}$$

Query column $n+1$:

Rows $d+2.....2d+1$:
$I\cdot\mathbf{1}_d/h - I\cdot X_{n+1} - \mathbf{1}_d/h = -X_{n+1}.$
Row $2d+2$:
$n^{-1/2}h^{-d/2}\cdot 1 - (1 + d/h) + d/h = n^{-1/2}h^{-d/2} - 1.$
Hence, we get

$$W_2^{(3)}R_3^{(n+1)} + b_2^{(3)} = \begin{pmatrix} 0_{d+1} \\ -X_{n+1} \\ n^{-1/2}h^{-d/2} - 1 \\ 0_{d_e-2d-2} \end{pmatrix}$$

Add residual connection

$Z^{(3)} = Z^{(2)} + \left(W_2^{(3)}R_3 + b_2^{(3)}\mathbf{1}_{n+1}^T\right)^T$. Adding each correction to the corresponding column-block of $Z^{(2)}$ (the $\mathbf{1}_n X_{n+1}^T$ block becomes $(X - \mathbf{1}_n X_{n+1}^T)/h$, and the $\tilde{K}$ column becomes $n^{-1/2}\sqrt{K_h}$):

$$\boxed{Z^{(3)} = \begin{pmatrix} X & Y & (X - \mathbf{1}_n X_{n+1}^T)/h & n^{-1/2}\sqrt{K_h(X, X_{n+1})} & 0_{n\times(d_e-2d-4)} & \mathbf{1}_n & 0_n \\ X_{n+1}^T & 0 & 0_d^T & n^{-1/2}h^{-d/2} & 0_{d_e-2d-4}^T & 1 & 1 \end{pmatrix}}$$

This is the input to Block 4 (or the read-out), where $n^{-1/2}\sqrt{K_h(X, X_{n+1})}$ is exactly the kernel-weighted quantity $\tilde{X}$ needs, and the query row carries $n^{-1/2}h^{-d/2}$ ready to be used as its own weight.

▼ Why we need B1-B6 to go from B7 → B8

In B.8 we need to compute, inside a transformer FFN:

$$\tilde{X}_{i,\nu} = \frac{\sqrt{K_h(X_i, X_{n+1})}}{\sqrt{n}}\cdot\frac{(X_i - X_{n+1})^\nu}{\nu!\, h^{|\nu|}}$$

This is a product of several numbers that are sitting in the token matrix. FFN layers can only do:
$x \mapsto W_2\,\mathrm{ReLU}(W_1 x + b_1) + b_2$
which is a piecewise linear map.
Multiplication is not piecewise linear. S
o the entire B.1–B.6 Lemmas exists to answer one question:
Can a piecewise linear function (ReLU network) approximate multiplication?
the path is:

1. approximate x^2 first,

2. then use $xy = \frac{(x+y)^2-(x-y)^2}{4}$ to get pairwise products,
3. then chain pairwise products to get k-fold products,
4. then handle general monomials

▼ Definition B.1

Statement. A function $f : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ is a ReLU network with width N, depth L, parameters bounded by B if:
$f = A_{L+1} \circ \mathrm{ReLU} \circ A_L \circ \mathrm{ReLU} \circ \cdots \circ \mathrm{ReLU} \circ A_1$
where each $A_\ell(z) = W_\ell z + b_\ell$, with $W_\ell \in [-B, B]^{d_\ell \times d_{\ell-1}}$, $b_\ell \in [-B, B]^{d_\ell}$, and $d_\ell \leq N$ for $\ell \in [L]$

▼ Lemma B.2

▼ B.2 Statement

If $f^{(1)} : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^m$ has width $N^{(1)}$, depth $L^{(1)}$, parameters bounded by $B^{(1)}$, and
$f^{(2)} : \mathbb{R}^m \to \mathbb{R}^{d_{\text{out}}}$ has width $N^{(2)}$, depth $L^{(2)}$, parameters bounded by $B^{(2)}$,
then $f^{(2)} \circ f^{(1)}$ is a network with:

- Width $\max(N^{(1)}, N^{(2)})$
- Depth $L^{(1)} + L^{(2)}$
- Parameters bounded by $\left(m \|W_1^{(2)}\|_{\max} \|W_{L^{(1)}+1}^{(1)}\|_{\max} \vee \ldots\right) \vee B^{(1)} \vee B^{(2)}$

→ When they stack two networks, the depths add and the widths take the max
→ The only subtlety is the parameter bound at the interface
→ the last layer of $f^{(1)}$ feeds into the first layer of $f^{(2)}$, and we have to multiply those weights together, which can increase the bound

▼ B.2 Proof

The interface is:
$A_1^{(2)} \circ A_{L^{(1)}+1}^{(1)}(z) = W_1^{(2)} \left(W_{L^{(1)}+1}^{(1)} z + b_{L^{(1)}+1}^{(1)}\right) + b_1^{(2)} = \underbrace{W_1^{(2)} W_{L^{(1)}+1}^{(1)}}_{\tilde{W}} z + \underbrace{W_1^{(2)} b_{L^{(1)}+1}^{(1)} + b_1^{(2)}}_{\tilde{b}}$
So define a single merged affine map $\tilde{A}(z) = \tilde{W} z + \tilde{b}$. The composed network becomes:
$f^{(2)} \circ f^{(1)} = A_{L^{(2)}+1}^{(2)} \circ \mathrm{ReLU} \circ \cdots \circ \mathrm{ReLU} \circ \tilde{A} \circ \mathrm{ReLU} \circ \cdots \circ \mathrm{ReLU} \circ A_1^{(1)}$

▼ Width

The hidden layer widths:
$d_\ell^{(1)} \leq N^{(1)}$ for $\ell \in [L^{(1)}]$ from $f^{(1)}$
$d_\ell^{(2)} \leq N^{(2)}$ for $\ell \in [L^{(2)}]$ from $f^{(2)}$
The merged layer $\tilde{A}$ maps from $d_{L^{(1)}}^{(1)}$ to $d_1^{(2)}$, so its hidden dimension is not a new hidden layer
So the maximum hidden width across the composed network is:
$\max\left(\max_{\ell \in [L^{(1)}]} d_\ell^{(1)}, \max_{\ell \in [L^{(2)}]} d_\ell^{(2)}\right) \leq \max(N^{(1)}, N^{(2)})$

▼ Depth

$f^{(1)}$ contributes $L^{(1)}$ ReLUs
$f^{(2)}$ contributes $L^{(2)}$ ReLUs
and the merge eliminated zero ReLUs
the interface had none to begin with.
Total depth $= L^{(1)} + L^{(2)}$

▼ Parameter Bound

Every parameter in the composed network comes from one of three places:

1. parameters from $f^{(1)}$ layers (except the last), bounded by $B^{(1)}$
2. parameters from $f^{(2)}$ layers (except the first), bounded by $B^{(2)}$
3. the merged layer $\tilde{A}$

Recall, The interface is:
$A_1^{(2)} \circ A_{L^{(1)}+1}^{(1)}(z) = W_1^{(2)}\left(W_{L^{(1)}+1}^{(1)} z + b_{L^{(1)}+1}^{(1)}\right) + b_1^{(2)} = \underbrace{W_1^{(2)} W_{L^{(1)}+1}^{(1)}}_{\tilde{W}} z + \underbrace{W_1^{(2)} b_{L^{(1)}+1}^{(1)} + b_1^{(2)}}_{\tilde{b}}$

$\tilde{W} = W_1^{(2)} W_{L^{(1)}+1}^{(1)}, \qquad \tilde{b} = W_1^{(2)} b_{L^{(1)}+1}^{(1)} + b_1^{(2)}$
For $\tilde{W}$: it is an $m \times m$ product of two matrices. Entry (i,j) of $\tilde{W}$ is:
$\tilde{W}_{ij} = \sum_{k=1}^{m} W_{1,ik}^{(2)} W_{L^{(1)}+1,kj}^{(1)}$
Taking absolute values:
$|\tilde{W}_{ij}| \le \sum_{k=1}^{m} |W_{1,ik}^{(2)}| \cdot |W_{L^{(1)}+1,kj}^{(1)}| \le m \cdot \|W_1^{(2)}\|_{\max} \cdot \|W_{L^{(1)}+1}^{(1)}\|_{\max}$
For $\tilde{b}$: entry i is:
$\tilde{b}_i = \sum_{k=1}^{m} W_{1,ik}^{(2)} b_{L^{(1)}+1,k}^{(1)} + b_{1,i}^{(2)}, \qquad |\tilde{b}_i| \le m \|W_1^{(2)}\|_{\max} \|b_{L^{(1)}+1}^{(1)}\|_\infty + \|b_1^{(2)}\|_\infty$
Combining both into a single bound for the merged layer:
$\|\tilde{W}\|_{\max} \vee \|\tilde{b}\|_\infty \le m \|W_1^{(2)}\|_{\max} \left(\|W_{L^{(1)}+1}^{(1)}\|_{\max} \vee \|b_{L^{(1)}+1}^{(1)}\|_\infty\right) + \|b_1^{(2)}\|_\infty$
The overall parameter bound for the composed network is therefore:
$\left(m \|W_1^{(2)}\|_{\max} \left(\|W_{L^{(1)}+1}^{(1)}\|_{\max} \vee \|b_{L^{(1)}+1}^{(1)}\|_\infty\right) + \|b_1^{(2)}\|_\infty\right) \vee B^{(1)} \vee B^{(2)}$

▼ Lemma B.3

▼ The problem it solves

The transformer FFN (Definition 2.2) always has this form:
$z \mapsto z + W_2 \,\mathrm{ReLU}(W_1 z + b_1) + b_2$
The $+z$ residual is always there & It cannot be removed
But when Lemma B.6 gives us a network approximating x^ν, that network computes:
$z \mapsto W_2 \,\mathrm{ReLU}(W_1 z + b_1) + b_2$
without the residual.
We cannot directly plug this into a transformer FFN layer, because the transformer would add z to it and we'd have to make adjustments
NOTE: Then this can also be applied in the transformer blocks construction part! can it be? how?

▼ Statement

Given any $W_1, W_2 \in [-B, B]^{d' \times d'}$ and
$b_1, b_2 \in [-B, B]^{d'}$,
there exist W_1', W_2', b_1', b_2' (with parameters still bounded by B, hidden width $3d'$) such that:
$z + W_2' \,\mathrm{ReLU}(W_1' z + b_1') + b_2' = W_2 \,\mathrm{ReLU}(W_1 z + b_1) + b_2$

▼ Proof

Construction

$$W_1' = \begin{pmatrix} W_1 \\ I \\ -I \end{pmatrix}, \quad b_1' = \begin{pmatrix} b_1 \\ 0 \\ 0 \end{pmatrix}, \quad W_2' = \begin{pmatrix} W_2 & -I & I \end{pmatrix}, \quad b_2' = b_2$$

Verification

$W_2' \,\mathrm{ReLU}(W_1' z + b_1') + b_2' = W_2 \,\mathrm{ReLU}(W_1 z + b_1) - \mathrm{ReLU}(z) + \mathrm{ReLU}(-z) + b_2 = W_2 \,\mathrm{ReLU}(W_1 z + b_1) - z + b_2$
using $\mathrm{ReLU}(-z) - \mathrm{ReLU}(z) = -z$ entrywise. Adding residual $+z$:
$z + W_2' \,\mathrm{ReLU}(W_1' z + b_1') + b_2' = W_2 \,\mathrm{ReLU}(W_1 z + b_1) + b_2$

Lemma B.4

Statement.

For $C \geq 1$ and $N, L \in \mathbb{N}$, there exists a ReLU network $\phi : \mathbb{R}^2 \to \mathbb{R}$ with:

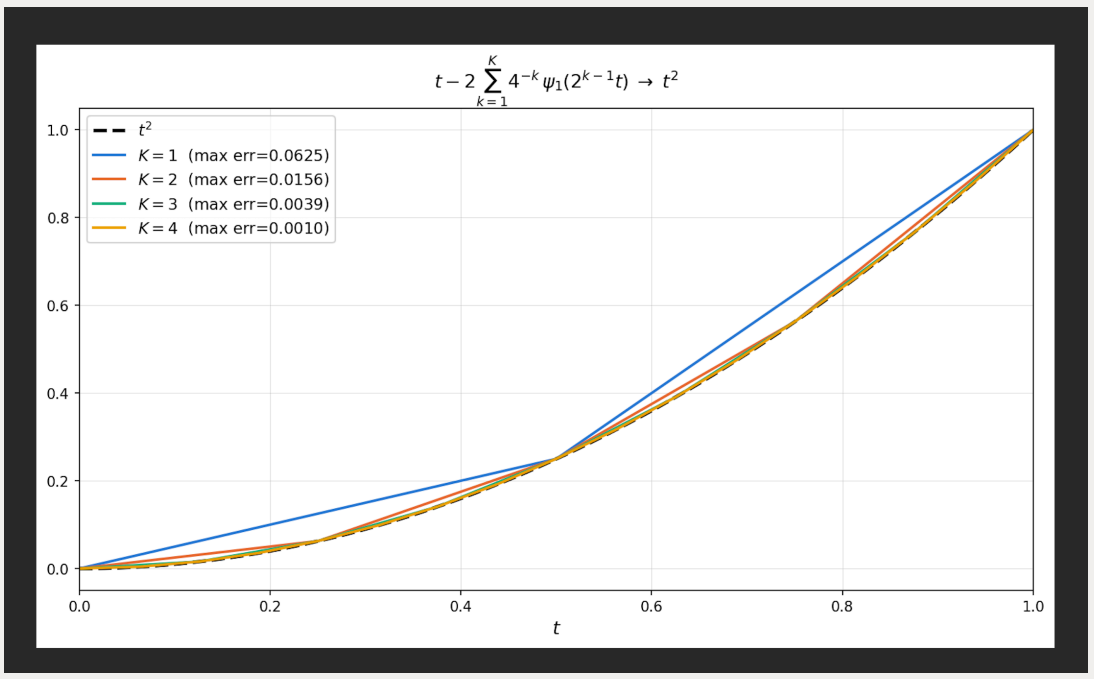
- Width $9N + 1$
- Depth L
- Parameters bounded by $32C^2N$

such that for all $(x, y) \in [-C, C]^2$:
$|\phi(x, y) - xy| \leq 24C^2N^{-L}$

The IMP step

$xy = \frac{(x+y)^2-(x-y)^2}{4}$
So if you can approximate t^2, you can approximate xy.
And ReLU networks can approximate t^2 because of the recursive identity:
$t^2 = 4\left(\frac{t}{2}\right)^2 = 4\left(\mathrm{ReLU}\left(\frac{t}{2}\right) - \mathrm{ReLU}\left(-\frac{t}{2}\right)\right)^2$
Applying this recursively L times — squeezing the input by half each time, squaring, then rescaling — gives a piecewise linear approximation to t^2 with error $\sim C^2 \cdot 2^{-L}$. With N breakpoints per layer, the error is C^2N^{-L}

Plot



The error $24C^2N^{-L}$ decreases exponentially in depth L. This is the key fact that makes the whole construction logarithmic in n — to get error $\leq n^{-k}$, you only need $L = k \log n / \log N = \Theta(\log n)$ layers.

Proof

We want a ReLU network $\phi : \mathbb{R}^2 \to \mathbb{R}$ such that $|\phi(x, y) - xy| \leq 24C^2N^{-L}$ for all $(x, y) \in [-C, C]^2$.
The proof has two stages:

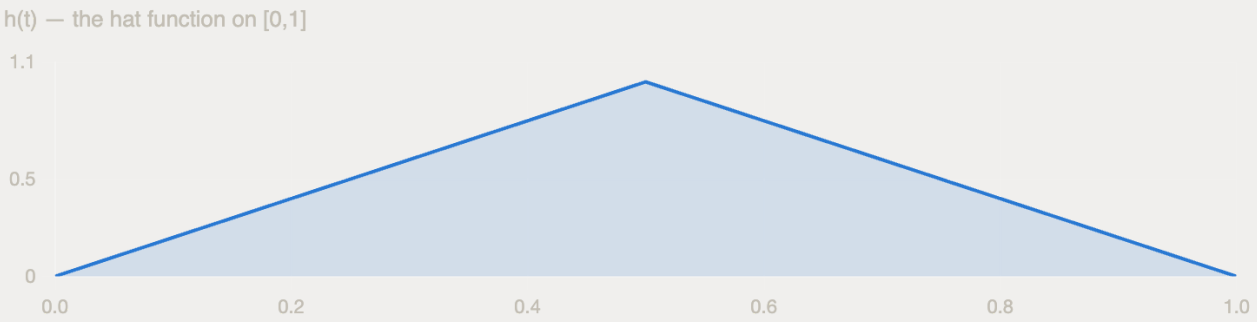
1. Build a ReLU network that approximates t^2 on $[-C, C]$
2. Use the polarization identity $xy = \frac{(x+y)^2-(x-y)^2}{4}$ to get xy

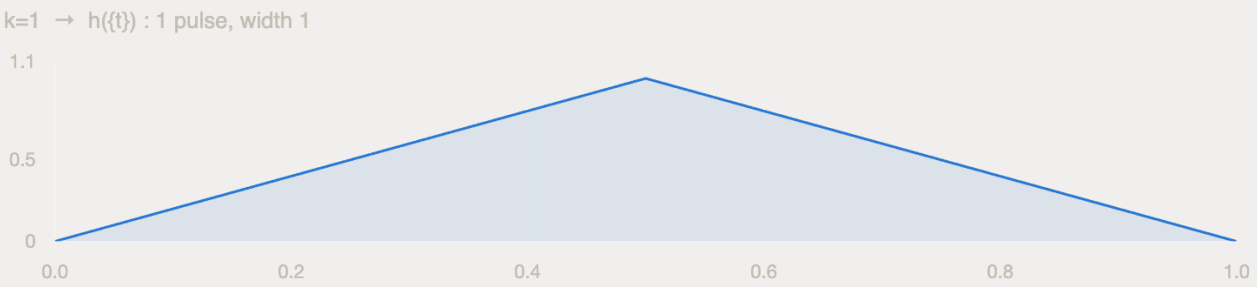
1: Approximating t^2 on $[0, 1]$

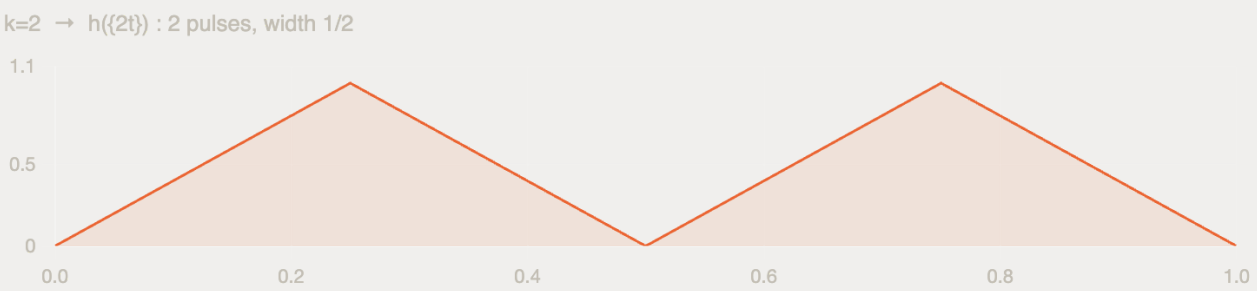
The hat function

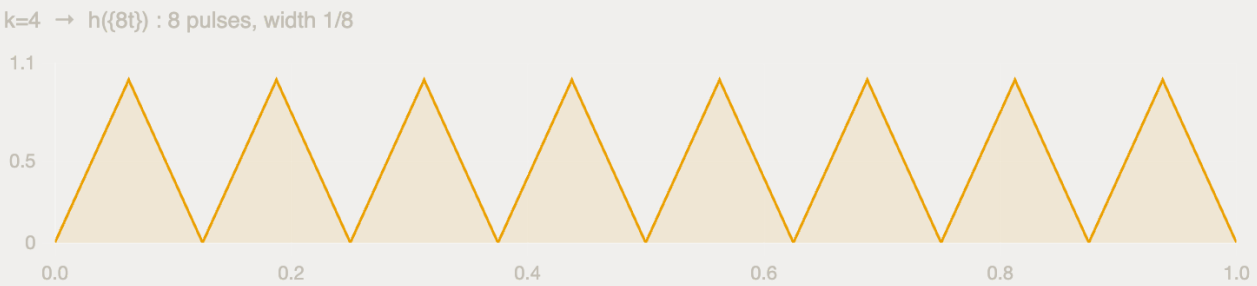
Define:
$\psi_1(t) = \mathrm{ReLU}(t) - 2\,\mathrm{ReLU}(t - \frac{1}{2}) + \mathrm{ReLU}(t - 1)$
On $[0, 1]$ this gives:
$\psi_1(t) = \begin{cases} t & t \in [0, \frac{1}{2}] \\ 1 - t & t \in [\frac{1}{2}, 1] \end{cases}$
which is a hat with peak $\frac{1}{2}$ at $t = \frac{1}{2}$. This is a depth-1 ReLU network with 3 neurons and parameters bounded by 2

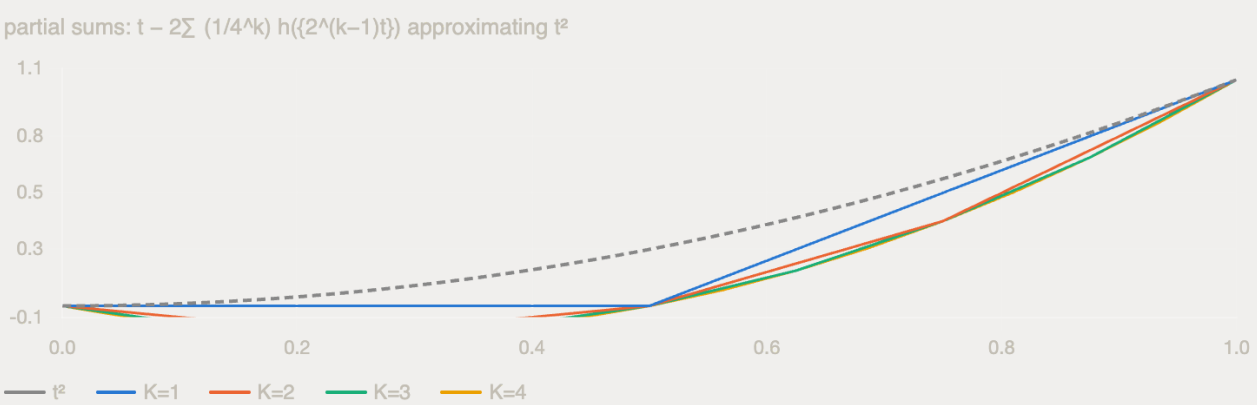
▼ Plot











The identity

For $t \in [0, 1]$:
$t^2 = t - 2\sum_{k=1}^{\infty} \frac{1}{4^k}\, \psi_1(2^{k-1}t)$

▼ Proof of identity:
Define $e(t) := t - t^2$. We claim:
$e(t) = 2\sum_{k=1}^{\infty} \frac{1}{4^k} \psi_1(2^{k-1}t)$

Truncating after L terms

Define the partial sum:

$S_L(t) := t - 2\sum_{k=1}^{L} \frac{1}{4^k}\psi_1(2^{k-1}t)$

The truncation error is:

$|S_L(t) - t^2| = 2\sum_{k=L+1}^{\infty} \frac{1}{4^k}\psi_1(2^{k-1}t) \le 2\sum_{k=L+1}^{\infty} \frac{1}{4^k}\cdot\frac{1}{2} = \sum_{k=L+1}^{\infty}\frac{1}{4^k} = \frac{1}{3\cdot 4^L}$ [using $\psi_1(\cdot) \le \frac{1}{2}$ everywhere]

(Have to check)

▼ Explicit Layer-by-Layer Computation

The network carries the pair (r_k, s_k) where $s_k = \{2^{k-1}t\}$ (fractional part).

Initialization (before any layer):

$r_0 = t, \qquad s_0 = t \quad (\text{since } t \in [0,1])$

Layer k: given (r_{k-1}, s_{k-1}), compute:

$\psi_1(s_{k-1}) = \mathrm{ReLU}(s_{k-1}) - 2\,\mathrm{ReLU}(s_{k-1} - \tfrac{1}{2}) + \mathrm{ReLU}(s_{k-1} - 1)$

This uses 3 ReLU neurons.

Then update:

$r_k = r_{k-1} - \frac{2}{4^k}\psi_1(s_{k-1})$

$s_k = 2s_{k-1} - \lfloor 2s_{k-1} \rfloor = \{2s_{k-1}\}$

The floor is implemented via: $\{2s\} = 2s - \mathrm{ReLU}(2s - 1)$ for $s \in [0,1]$ (since $\lfloor 2s \rfloor = 0$ if $s < \frac{1}{2}$ and 1 if $s \ge \frac{1}{2}$, so $\lfloor 2s \rfloor = \mathrm{ReLU}(2s-1)$ — one more ReLU neuron... but this is absorbed into the 3 neurons of ψ_1 at the next step).

The Network Written Out for $L = 3$

Let $c_k = \frac{2}{4^k}$ be the coefficients: $c_1 = \frac{1}{2}$, $c_2 = \frac{1}{8}$, $c_3 = \frac{1}{32}$.

$r_0 = t$

After layer 1:

$r_1 = t - \frac{1}{2}\underbrace{\psi_1(t)}_{3\ \mathrm{ReLUs}}$

At $t = 0.3$: $\psi_1(0.3) = 0.3$, so $r_1 = 0.3 - 0.15 = 0.15$. True $t^2 = 0.09$. Error $= 0.06$.

After layer 2:

$r_2 = r_1 - \frac{1}{8}\underbrace{\psi_1(2t)}_{3\ \mathrm{ReLUs}}$

At $t = 0.3$: $\psi_1(0.6) = 1 - 0.6 = 0.4$, so $r_2 = 0.15 - 0.05 = 0.10$. Error $= 0.01$.

After layer 3:

$r_3 = r_2 - \frac{1}{32}\underbrace{\psi_1(4t)}_{3\ \mathrm{ReLUs}}$

At $t = 0.3$: $4t = 1.2$, $\{1.2\} = 0.2$, $\psi_1(0.2) = 0.2$, so $r_3 = 0.10 - \frac{0.2}{32} = 0.10 - 0.00625 = 0.09375$. Error $= 0.00375$.

Compare error bound $\frac{1}{3\cdot 4^3} = \frac{1}{192} \approx 0.0052$. ✓

Numerical check

K	r_K at $t = 0.3$	error vs 0.09	bound $\frac{1}{3\cdot 4^K}$
0	0.300	0.210	—
1	0.150	0.060	0.0833
2	0.100	0.010	0.0208
3	0.094	0.004	0.0052
4	0.091	0.001	0.0013

Each layer roughly quarters the error — geometric convergence, which is why you only need $L = \Theta(\log n)$ layers for error $< n^{-c}$.

▼ 2: From Squaring to Multiplication

Rescaling to $[-C, C]$

For $t \in [-2C, 2C]$ (we need this range for $x + y$ and $x - y$), define:

$\phi_{\mathrm{sq}}^{2C}(t) := (2C)^2 \cdot \phi_{\mathrm{sq}}\left(\frac{|t|}{2C}\right)$

using $|t| = \mathrm{ReLU}(t) + \mathrm{ReLU}(-t)$, which adds one layer. This gives for all $t \in [-2C, 2C]$:

$|\phi_{\mathrm{sq}}^{2C}(t) - t^2| = (2C)^2\left|\phi_{\mathrm{sq}}\left(\frac{|t|}{2C}\right) - \left(\frac{|t|}{2C}\right)^2\right| \le \frac{(2C)^2}{3\cdot 4^L} = \frac{4C^2}{3\cdot 4^L}$

The polarization identity

For any $x, y \in \mathbb{R}$:

$xy = \frac{(x+y)^2 - (x-y)^2}{4}$

Proof: $(x+y)^2 - (x-y)^2 = x^2 + 2xy + y^2 - x^2 + 2xy - y^2 = 4xy$. ✓

Define the approximating network

$\phi(x,y) := \frac{\phi_{\text{sq}}^{2C}(x+y) - \phi_{\text{sq}}^{2C}(x-y)}{4}$
For $(x,y) \in [-C,C]^2$: $x+y \in [-2C,2C]$ and $x-y \in [-2C,2C]$, so both inputs are in the valid domain.

Error bound

$|\phi(x,y) - xy| = \left| \frac{\phi_{\text{sq}}^{2C}(x+y) - \phi_{\text{sq}}^{2C}(x-y)}{4} - \frac{(x+y)^2 - (x-y)^2}{4} \right|$
$\leq \frac{1}{4} \left| \phi_{\text{sq}}^{2C}(x+y) - (x+y)^2 \right| + \frac{1}{4} \left| \phi_{\text{sq}}^{2C}(x-y) - (x-y)^2 \right|$
$\leq \frac{1}{4} \cdot \frac{4C^2}{3 \cdot 4^L} + \frac{1}{4} \cdot \frac{4C^2}{3 \cdot 4^L} = \frac{2C^2}{3 \cdot 4^L}$
Since $4^L \geq N^L$ for $N \leq 4$, and absorbing the constant $2/3 \leq 24$:
$|\phi(x,y) - xy| \leq 24C^2 N^{-L}$

▼ Lemma B.5

▼ Statement

For $C \geq 1$, $k \geq 2$, there exists a ReLU network $\phi : \mathbb{R}^k \to \mathbb{R}$ with:

- Width $9(N+1) + 2k - 1$
- Depth $7kL(k-1)$
- Parameters bounded by $3C^k(40N+40)^2$

such that for all $(x_1, \ldots, x_k) \in [-C,C]^k$:
$|\phi(x_1, \ldots, x_k) - x_1 \cdots x_k| \leq 30C^k(k-1)(N+1)^{-7kL}$

▼ Why we need B5

B6 Statement (monomials). For $\nu \in \mathbb{N}_0^d$ with $|\nu| \leq k$, there exists a ReLU network $\psi : \mathbb{R}^d \to \mathbb{R}$ with:

- Width $9(N+1) + 2k - 1$
- Depth $7kL(k-1) + 1$
- Parameters bounded by $3(k+1)C^k(40N+40)^2$

such that for all $x \in [-C,C]^d$:
$|\psi(x) - x^\nu| \leq 30C^k(k-1)(N+1)^{-7kL}$

How it reduces to B.5

$x^\nu = x_1^{\nu_1} \cdots x_d^{\nu_d}$ is a product of $|\nu|$ scalars — just with repetitions. Specifically it is the product of the list:
$(\underbrace{x_1, \ldots, x_1}_{\nu_1}, \underbrace{x_2, \ldots, x_2}_{\nu_2}, \ldots, \underbrace{x_d, \ldots, x_d}_{\nu_d})$
which has $|\nu| \leq k$ elements.

▼ Proof

Proof strategy:

We want a ReLU network $\phi_k : \mathbb{R}^k \to \mathbb{R}$ such that for all $(x_1, \ldots, x_k) \in [-C,C]^k$:
$|\phi_k(x_1, \ldots, x_k) - x_1 \cdots x_k| \leq 30C^k(k-1)(N+1)^{-7kL}$
We prove this by induction on k

▼ Base Case $k = 2$

▼ By Lemma B.4
For $C \geq 1$ and $N, L \in \mathbb{N}$, there exists a ReLU network $\phi : \mathbb{R}^2 \to \mathbb{R}$ with:

- Width $9N + 1$
- Depth L
- Parameters bounded by $32C^2N$

such that for all $(x,y) \in [-C,C]^2$:
$|\phi(x,y) - xy| \leq 24C^2N^{-L}$

with depth $7 \cdot 2 \cdot L = 14L$, there exists $\phi_2 : \mathbb{R}^2 \to \mathbb{R}$ with width $9(N+1)+1$, such that for all $(x_1, x_2) \in [-C, C]^2$:
$|\phi_2(x_1, x_2) - x_1 x_2| \leq 24C^2(N+1)^{-14L} \leq 30C^2 \cdot 1 \cdot (N+1)^{-7 \cdot 2 \cdot L}$ [since $24 \leq 30$ and $k - 1 = 1$.]

▼ Inductive Step: Assume True for m, Prove for $m+1$

Inductive hypothesis: there exists $\phi_m : \mathbb{R}^m \to \mathbb{R}$ with depth $7kL(m-1)$, width $9(N+1)+2m-1$, such that for all $(x_1, \ldots, x_m) \in [-C, C]^m$:
$|\phi_m(x_1, \ldots, x_m) - x_1 \cdots x_m| \leq 30C^m(m-1)(N+1)^{-7kL} \qquad (*)$
Define $\phi_{m+1} : \mathbb{R}^{m+1} \to \mathbb{R}$ by:
$\phi_{m+1}(x_1, \ldots, x_{m+1}) := \phi_2(\phi_m(x_1, \ldots, x_m),\ x_{m+1})$
where ϕ_2 is the B.4 network with domain $[-C^{m+1}, C^{m+1}]$ (justified below).

Now

We need to bound:
$|\phi_2(\phi_m(x_1, \ldots, x_m), x_{m+1}) - x_1 \cdots x_m \cdot x_{m+1}|$
Add and subtract $\phi_m(x_1, \ldots, x_m) \cdot x_{m+1}$:
$\leq \underbrace{|\phi_2(\phi_m(x_1, \ldots, x_m), x_{m+1}) - \phi_m(x_1, \ldots, x_m) \cdot x_{m+1}|}_{\text{Term A: error of this multiplication step}} + \underbrace{|\phi_m(x_1, \ldots, x_m) \cdot x_{m+1} - x_1 \cdots x_m \cdot x_{m+1}|}_{\text{Term B: propagated error from previous steps}}$

▼ Bounding Term A

This is the error of ϕ_2 when applied to inputs $(\phi_m(x_1, \ldots, x_m), x_{m+1})$.
To apply B.4, we need to know the domain.
We claim $|\phi_m(x_1, \ldots, x_m)| \leq C^{m+1}$

▼ Justifying $|\phi_m(x_1, \ldots, x_m)| \leq C^{m+1}$

We need $|\phi_m(x_1, \ldots, x_m)| \leq C^{m+1}$ to apply B.4.
From the inductive hypothesis $(*)$:
$|\phi_m(x_1, \ldots, x_m)| \leq |x_1 \cdots x_m| + |\phi_m - x_1 \cdots x_m|$
$\leq C^m + 30C^m(m-1)(N+1)^{-7kL}$
For large enough N or L, the second term is $\leq C^m$, giving $|\phi_m| \leq 2C^m \leq C^{m+1}$ (since $C \geq 2$ in all applications, as $C = 2(M+1)n^{1/(2\alpha+d)} \geq 2$).

Combined with $|x_{m+1}| \leq C$, the inputs lie in $[-C^{m+1}, C^{m+1}]^2$ for large enough C.
B.4 with domain bound C^{m+1} and depth $7kL$ gives:
$\text{Term A} \leq 24(C^{m+1})^2(N+1)^{-7kL} \leq 30C^{m+1}(N+1)^{-7kL}$
where the last step absorbs C^{m+1} and 24 into the constant 30 (valid since $C \geq 1$).

▼ Bounding Term B

$\text{Term B} = |\phi_m(x_1, \ldots, x_m) - x_1 \cdots x_m| \cdot |x_{m+1}| \leq 30C^m(m-1)(N+1)^{-7kL} \cdot C$ by $(*)$ and $|x_{m+1}| \leq C$
$= 30C^{m+1}(m-1)(N+1)^{-7kL}$

▼ Combining

$\text{Term A} + \text{Term B} \leq 30C^{m+1}(N+1)^{-7kL} + 30C^{m+1}(m-1)(N+1)^{-7kL}$
$= 30C^{m+1} \cdot m \cdot (N+1)^{-7kL} = 30C^{m+1} \cdot ((m+1)-1) \cdot (N+1)^{-7kL}$
This is exactly the bound for index $m+1$.

▼ Architecture via B.2

Composing $\phi_{m+1} = \phi_2(\phi_m(x_1, \ldots, x_m),\ x_{m+1})$:

Depth

B.2 says depths add:
$\text{depth}(\phi_{m+1}) = \underbrace{7kL(m-1)}_{\phi_m} + \underbrace{7kL}_{\phi_2} = 7kL \cdot m$
After $k-1$ total steps, depth $= 7kL(k-1)$

Width

B.2 says widths take the max, plus one extra channel per remaining input passed alongside:
$\text{width} = 9(N+1) + 2k - 1$

Parameters

B.2 says the interface between ϕ_m's output and ϕ_2's input multiplies two weight matrices, contributing a factor of C per step.

After $k-1$ steps: $3C^k(40N+40)^2$

Final Result

By induction, for all $k \geq 2$ and $(x_1, \ldots, x_k) \in [-C, C]^k$:
$|\phi_k(x_1, \ldots, x_k) - x_1 \cdots x_k| \leq 30C^k(k-1)(N+1)^{-7kL}$

▼ Lemma B.6

▼ Lemma B.8

▼ Lemma B.9

▼ The Objective

$$f(w) = \|\tilde{Y} - \tilde{X}w\|_2^2$$

$$\nabla f(w) = 2\tilde{X}^T\tilde{X}w - 2\tilde{X}^T\tilde{Y}$$

$$\nabla^2 f(w) = 2\tilde{X}^T\tilde{X} \quad (\text{constant} — f \text{ is quadratic})$$

The Algorithm

$w_0 = 0_D, \qquad w_{t+1} = w_t - \frac{g_t}{C_2}$
where $\|g_t - \nabla f(w_t)\|_2 \leq \varepsilon$ for all t.

▼ We cannot compute $\nabla f(w_t)$ exactly, only g_t with $\|g_t - \nabla f(w_t)\|_2 \leq \varepsilon$ ⇒ We need to show w_T is still close to w^*.
Why we can not compute $\nabla f(w_t)$ exactly?
Because the transformer in B.10 does not have the true $\tilde{X}$ and $\tilde{Y}$ — it only has $\hat{X}$ and $\hat{Y}$, the approximations from B.8.
now, The true gradient is:
$\nabla f(w_t) = 2\tilde{X}^T\tilde{X}w_t - 2\tilde{X}^T\tilde{Y}$
But the attention layer in B.10 computes using $\hat{X}$ and $\hat{Y}$ instead, giving:
$g_t = 2\hat{X}^T\hat{X}w_t - 2\hat{X}^T\hat{Y}$
The difference is:
$g_t - \nabla f(w_t) = 2(\hat{X}^T\hat{X} - \tilde{X}^T\tilde{X})w_t - 2(\hat{X}^T\hat{Y} - \tilde{X}^T\tilde{Y})$
Since $\|\hat{X} - \tilde{X}\|_{\max} \leq \xi$ and $\|\hat{Y} - \tilde{Y}\|_\infty \leq \xi$ from B.8, this difference is bounded by $\varepsilon = O(n\xi)$. That is exactly the noise ε in B.9.

▼ Proof

▼ Proof Strategy

Split into two questions answered independently, then combine:

- Part 2: How fast does exact (noiseless) gradient descent converge to w^*?
- Part 3: How far do the noisy iterates w_t drift from the noiseless iterates?

▼ Part 1

Strong convexity constant C_1

Definition. f is C_1-strongly convex if for all $w, w' \in \mathbb{R}^D$:
$f(w') \geq f(w) + \nabla f(w)^T(w' - w) + \frac{C_1}{2}\|w' - w\|_2^2$
Why equivalent to $\nabla^2 f \succeq C_1 I$. Since f is quadratic, its Taylor expansion is exact:
$f(w') = f(w) + \nabla f(w)^T(w' - w) + \frac{1}{2}(w' - w)^T\nabla^2 f(w)(w' - w)$
For strong convexity to hold for all w', setting $v = w' - w$:
$\frac{1}{2}v^T\nabla^2 f(w)v \geq \frac{C_1}{2}\|v\|_2^2 \quad \forall v \in \mathbb{R}^D$
$\iff v^T\nabla^2 f(w)v \geq C_1\|v\|_2^2 \quad \forall v$
$\iff \nabla^2 f(w) \succeq C_1 I$
For our f. Substituting $\nabla^2 f = 2\tilde{X}^T\tilde{X}$:
$v^T(2\tilde{X}^T\tilde{X})v \geq C_1\|v\|_2^2 \quad \forall v$
The left side equals $2v^T\tilde{X}^T\tilde{X}v$. By definition of minimum eigenvalue:
$v^T\tilde{X}^T\tilde{X}v \geq \lambda_{\min}(\tilde{X}^T\tilde{X}) \cdot \|v\|_2^2 \quad \forall v$
So we take $C_1 = 2\lambda_{\min}(\tilde{X}^T\tilde{X})$.
From event (6) of Theorem 2.5: $\lambda_{\min}(\tilde{X}^T\tilde{X}) \geq 1/C_0$, giving:
$C_1 = \frac{2}{C_0}$

Lipschitz gradient constant C_2

Definition. f has C_2-Lipschitz gradient if:
$\|\nabla f(w) - \nabla f(w')\|_2 \leq C_2\|w - w'\|_2 \quad \forall w, w' \in \mathbb{R}^D$
For our f:
$\nabla f(w) - \nabla f(w') = 2\tilde{X}^T\tilde{X}(w - w')$
$\|\nabla f(w) - \nabla f(w')\|_2 = \|2\tilde{X}^T\tilde{X}(w - w')\|_2 \leq 2\lambda_{\max}(\tilde{X}^T\tilde{X})\|w - w'\|_2$
From event (6): $\lambda_{\max}(\tilde{X}^T\tilde{X}) \leq C_0$, giving:
$C_2 = 2C_0$

▼ Part 2: Exact Gradient Descent Converges Exponentially

Define clean iterates with exact gradient:

$$v_0 = 0_D, \qquad v_{t+1} = v_t - \frac{\nabla f(v_t)}{C_2}$$

Claim:

$$\|v_T - w^*\|_2 \leq \exp\left(-\frac{C_1 T}{2C_2}\right)\|w^*\|_2$$

Proof. To show:

$$\|v_{t+1} - w^*\|_2^2 \leq \left(1 - \frac{C_1}{C_2}\right)\|v_t - w^*\|_2^2$$

Expand $\|v_{t+1} - w^*\|_2^2$:

$$\|v_{t+1} - w^*\|_2^2 = \left(v_t - \frac{\nabla f(v_t)}{C_2} - w^*\right)^T\left(v_t - \frac{\nabla f(v_t)}{C_2} - w^*\right)$$

$$= \|v_t - w^*\|_2^2 - \frac{2}{C_2}\nabla f(v_t)^T(v_t - w^*) + \frac{\|\nabla f(v_t)\|_2^2}{C_2^2} \text{—}(A)$$

We need a lower bound on $\nabla f(v_t)^T(v_t - w^*)$ and an upper bound on $\|\nabla f(v_t)\|_2^2$.

▼ Lower bound on $\nabla f(v_t)^T(v_t - w^*)$ via strong convexity.

Apply the strong convexity definition at $(w, w') = (w^*, v_t)$. Since w^* is the minimizer, $\nabla f(w^*) = 0$:
$f(v_t) \geq f(w^*) + \frac{C_1}{2}\|v_t - w^*\|_2^2$ — call this (I)
Apply at $(w, w') = (v_t, w^*)$:
$f(w^*) \geq f(v_t) + \nabla f(v_t)^T(w^* - v_t) + \frac{C_1}{2}\|w^* - v_t\|_2^2$ — call this (II)
Add (I) and (II). Left side is 0:
$0 \geq C_1\|v_t - w^*\|_2^2 + \nabla f(v_t)^T(w^* - v_t)$
$\implies \nabla f(v_t)^T(v_t - w^*) \geq C_1\|v_t - w^*\|_2^2$ — this is (III)

What We Want to Prove

$\nabla f(v_t)^T(v_t - w^*) \geq \frac{1}{C_2}\|\nabla f(v_t)\|_2^2$ — call this (IV)

▼ The Descent Lemma

▼ Statement

If f is C_2-smooth (i.e. $\|\nabla f(w) - \nabla f(w')\|_2 \leq C_2\|w - w'\|_2$), then for all w, w':
$f(w') \leq f(w) + \nabla f(w)^T(w' - w) + \frac{C_2}{2}\|w' - w\|_2^2$

▼ **Proof of descent lemma**

By the fundamental theorem of calculus
$f(w') - f(w) = \int_0^1 \nabla f(w + t(w' - w))^T(w' - w)\, dt$
$= \nabla f(w)^T(w' - w) + \int_0^1 [\nabla f(w + t(w' - w)) - \nabla f(w)]^T(w' - w)\, dt$
Bound the integral using C_2-smoothness:
$\left|\int_0^1 [\nabla f(w + t(w' - w)) - \nabla f(w)]^T(w' - w)\, dt\right| \leq \int_0^1 \|\nabla f(w + t(w' - w)) - \nabla f(w)\|_2\|w' - w\|_2\, dt$
$\leq \int_0^1 C_2 t\|w' - w\|_2^2\, dt = \frac{C_2}{2}\|w' - w\|_2^2$
Therefore:
$f(w') \leq f(w) + \nabla f(w)^T(w' - w) + \frac{C_2}{2}\|w' - w\|_2^2$

▼ Now Prove Co-coercivity

We want to prove (IV): $\nabla f(v_t)^T(v_t - w^*) \geq \frac{1}{C_2}\|\nabla f(v_t)\|_2^2$.
Apply the descent lemma at $w = v_t$, $w' = v_t - \frac{\nabla f(v_t)}{C_2}$:

$$f\left(v_t - \frac{\nabla f(v_t)}{C_2}\right) \leq f(v_t) + \nabla f(v_t)^T\left(-\frac{\nabla f(v_t)}{C_2}\right) + \frac{C_2}{2}\|\frac{\nabla f(v_t)}{C_2}\|_2^2$$

$$= f(v_t) - \frac{\|\nabla f(v_t)\|_2^2}{C_2} + \frac{\|\nabla f(v_t)\|_2^2}{2C_2}$$

$$= f(v_t) - \frac{\|\nabla f(v_t)\|_2^2}{2C_2} - (*)$$

Since w^* is the minimizer of f:
$f(w^*) \leq f\left(v_t - \frac{\nabla f(v_t)}{C_2}\right) \leq f(v_t) - \frac{\|\nabla f(v_t)\|_2^2}{2C_2}$ — call this ()
Now apply convexity of f at $w = v_t$, $w' = w^*$:
$f(w^*) \geq f(v_t) + \nabla f(v_t)^T(w^* - v_t)$
$\implies f(v_t) - f(w^*) \leq \nabla f(v_t)^T(v_t - w^*)$ — call this (*)
From (): $f(v_t) - f(w^*) \geq \frac{\|\nabla f(v_t)\|_2^2}{2C_2}$.
Substitute into (*):
$\nabla f(v_t)^T(v_t - w^*) \geq f(v_t) - f(w^*) \geq \frac{\|\nabla f(v_t)\|_2^2}{2C_2}$
$\implies \frac{\|\nabla f(v_t)\|_2^2}{C_2^2} \leq \frac{2\nabla f(v_t)^T(v_t - w^*)}{C_2}$ — call this (IV')

▼ Upper bound on $\|\nabla f(v_t)\|_2^2$ via co-coercivity

Substitute (III) and (IV) into (A):
$\|v_{t+1} - w^*\|_2^2 \leq \|v_t - w^*\|_2^2 - \frac{2}{C_2}\nabla f(v_t)^T(v_t - w^*) + \frac{\nabla f(v_t)^T(v_t - w^*)}{C_2}$
$= \|v_t - w^*\|_2^2 - \frac{1}{C_2}\nabla f(v_t)^T(v_t - w^*)$
$\leq \|v_t - w^*\|_2^2 - \frac{C_1}{C_2}\|v_t - w^*\|_2^2 = \left(1 - \frac{C_1}{C_2}\right)\|v_t - w^*\|_2^2$
The last step uses (III) with a negative sign: since the term appears as $-\frac{1}{C_2}(\cdot)$, substituting the lower bound $C_1\|v_t - w^*\|_2^2$ gives an upper bound.
Apply recursively over T steps. Taking square roots:
$\|v_{t+1} - w^*\|_2 \leq \left(1 - \frac{C_1}{C_2}\right)^{1/2}\|v_t - w^*\|_2$
At $t = 0$: $\|v_1 - w^*\|_2 \leq \left(1 - \frac{C_1}{C_2}\right)^{1/2}\|v_0 - w^*\|_2$
At $t = 1$: $\|v_2 - w^*\|_2 \leq \left(1 - \frac{C_1}{C_2}\right)^{2/2}\|v_0 - w^*\|_2$
After T steps, with $v_0 = 0_D$ so $\|v_0 - w^*\|_2 = \|w^*\|_2$:
$\|v_T - w^*\|_2 \leq \left(1 - \frac{C_1}{C_2}\right)^{T/2}\|w^*\|_2$
Using $1 - x \leq e^{-x}$ for all x:
$\|v_T - w^*\|_2 \leq \exp\left(-\frac{C_1 T}{2C_2}\right)\|w^*\|_2$ — call this (V)

▼ Part 3: Noisy Iterates Stay Close to Clean Iterates

▼ Claim:

$\|w_T - v_T\|_2 \leq \frac{T\varepsilon}{C_2}$

▼ Proof

The Two Updates Written Out

$$w_{t+1} = w_t - \frac{g_t}{C_2}$$

$$v_{t+1} = v_t - \frac{\nabla f(v_t)}{C_2}$$

Subtract:

$$w_{t+1} - v_{t+1} = (w_t - v_t) - \frac{g_t - \nabla f(v_t)}{C_2} - (*)$$

The Problem With the Naive Bound

Take norms in (*) and apply triangle inequality:
$\|w_{t+1} - v_{t+1}\|_2 \leq \|w_t - v_t\|_2 + \frac{\|g_t - \nabla f(v_t)\|_2}{C_2}$
Now bound $\|g_t - \nabla f(v_t)\|_2$ by adding and subtracting $\nabla f(w_t)$:
$\|g_t - \nabla f(v_t)\|_2 \leq \|g_t - \nabla f(w_t)\|_2 + \|\nabla f(w_t) - \nabla f(v_t)\|_2 \leq \varepsilon + C_2\|w_t - v_t\|_2$
Substituting:
$\|w_{t+1} - v_{t+1}\|_2 \leq 2\|w_t - v_t\|_2 + \frac{\varepsilon}{C_2}$
This has coefficient 2 on $\|w_t - v_t\|_2$. Summing over T steps gives $\sim 2^T$ growth. Useless.

The Fix: Split the Noisy Step Differently

The noisy update $w_{t+1} = w_t - g_t/C_2$ can be rewritten. Add and subtract $\nabla f(w_t)/C_2$:
$w_{t+1} = w_t - \frac{\nabla f(w_t)}{C_2} + \frac{\nabla f(w_t) - g_t}{C_2}$
The first two terms is exactly what the clean step from w_t would give. Call it $\tilde{v}_{t+1} := w_t - \nabla f(w_t)/C_2$. So:
$w_{t+1} = \tilde{v}_{t+1} + \frac{\nabla f(w_t) - g_t}{C_2}$ — call this (A)
$v_{t+1} = v_t - \frac{\nabla f(v_t)}{C_2}$ — call this (B)
Subtract (B) from (A):

$$w_{t+1} - v_{t+1} = \underbrace{(\tilde{v}_{t+1} - v_{t+1})}_{\text{clean step from } w_t \text{ minus clean step from } v_t} + \underbrace{\frac{\nabla f(w_t) - g_t}{C_2}}_{\text{noise term}}$$

Bound Each Part

Noise term:
$\|\frac{\nabla f(w_t) - g_t}{C_2}\|_2 = \frac{\|g_t - \nabla f(w_t)\|_2}{C_2} \leq \frac{\varepsilon}{C_2}$
Clean step difference $\|\tilde{v}_{t+1} - v_{t+1}\|_2$:
$\tilde{v}_{t+1} - v_{t+1} = \left(w_t - \frac{\nabla f(w_t)}{C_2}\right) - \left(v_t - \frac{\nabla f(v_t)}{C_2}\right)$
$= (w_t - v_t) - \frac{\nabla f(w_t) - \nabla f(v_t)}{C_2}$
From Part 2, the clean gradient step is a contraction with factor $(1 - C_1/C_2)^{1/2}$. This contraction applies to any two starting points, not just to the distance from w^*. Specifically, applying the same algebra from Part 2 to the pair (w_t, v_t) instead of (v_t, w^*):
$\|\tilde{v}_{t+1} - v_{t+1}\|_2 \leq \left(1 - \frac{C_1}{C_2}\right)^{1/2} \|w_t - v_t\|_2$

Combine

$\|w_{t+1} - v_{t+1}\|_2 \leq \left(1 - \frac{C_1}{C_2}\right)^{1/2} \|w_t - v_t\|_2 + \frac{\varepsilon}{C_2}$
Since $(1 - C_1/C_2)^{1/2} \leq 1$:
$\|w_{t+1} - v_{t+1}\|_2 \leq \|w_t - v_t\|_2 + \frac{\varepsilon}{C_2}$
Coefficient on $\|w_t - v_t\|_2$ is now ≤ 1, not 2. Sum from $t = 0$ to $T - 1$ with $w_0 = v_0 = 0_D$:
$\|w_T - v_T\|_2 \leq \frac{T\varepsilon}{C_2}$ ✓

▼ **Lemma B.10**