Sketching-Based Least Squares Approaches for Random Fourier Neural Networks

by

Norman B. Boyd

THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Bachelor of Science

Mathematics

The University of New Mexico

Albuquerque, New Mexico

May, 2026

Acknowledgments

I would like to acknowledge those who have supported and contributed to my work on this thesis.

My advisor, Professor Mohammad Motamed, for his support in writing this thesis and the opportunity to work in his group.

Jose Agudelo for his support in learning about Random Fourier Neural Networks and for sharing the Python code of his work with me.

Owen Davis of Sandia National Laboratories for his advisement on my work and recommendation of literature.

Sketching-Based Least Squares Approaches for Random Fourier Neural Networks

by

Norman B. Boyd

B.S., Mathematics, University of New Mexico, 2026

Abstract

Random Fourier Neural Networks (rFNNs) are trained by repeatedly solving overdetermined least squares problems to find amplitudes corresponding to sampled frequencies. Because these solves comprise the majority of the cost of training, this thesis investigates whether randomized sketching-based methods can reduce runtime while preserving sufficient accuracy for training rFNNs. We study the regularized least squares problem produced from one training iteration of an rFNN with one layer and compare Householder QR against three sketching-based solvers: Sketch-and-Solve, Sketch-and-Precondition, and Iterative Sketching. Using Clarkson-Woodruff sketching for these methods, we summarize the relevant subspace embedding guarantees and experimentally evaluate the methods in terms of conditioning, embedding distortion, and runtime, as well as forward, residual, and backward error. The results show that Sketch-and-Solve can provide substantial speedups but is not sufficiently accurate for this application. In contrast, Sketch-and-Precondition and Iterative Sketching achieve errors consistent with the Tikhonov regularization level across a broad range of sketch dimensions while also outperforming Householder QR in runtime for many problem sizes. Sketch-and-Precondition is the fastest among the tested methods, while Iterative Sketching is marginally slower and more sensitive to poor subspace embeddings.

Contents

List of Figures

Glossary

W Width (number of neurons) of the first layer of a Random Fourier Neural Network, half the width of all later layers. Also an important parameter for the dimension of the least squares problem.

L Depth (number of layers) in a Random Fourier Neural Network.

N Number of samples from the target function for training. The most important parameter determining the number of rows in the least squares problem.

λ Tikhonov regularization parameter. Used in the regularized least squares problem to improve stability.

θ Scalar input variable for the target function. In our example, $\theta \in [-1, 1]$.

$Q(\theta)$ Target function being evaluated at point θ.

A Overdetermined matrix for the least squares problem.

$\widetilde{A}$ Augmented / regularized least squares matrix in $\mathbb{R}^{m \times n}$.

β Least squares unknown / coefficient vector of Fourier amplitudes or weights in $\mathbb{R}^n$.

$\beta^\star$ Exact minimizer of the unsketched least squares problem.

β_0 Initial guess used to start iterative methods. Often $\beta_0 = R^{-1}Q^\top(Sy)$.

Glossary

β_i Iteration i approximation of β in iterative solvers.

y Vector of target function evaluations; right-hand side of the unaugmented least squares problem.

$\widetilde{y}$ Augmented / regularized right-hand side of least squares problem.

$r(\beta)$ Residual as a function of β. Defined as $r(\beta) = y - A\beta$ for a generalized system $A\beta = y$.

S Clarkson–Woodruff sparse sketching matrix. We take $S \in \mathbb{R}^{d \times m}$.

d Number of rows of the sketching matrix S. Also called the sketch dimension or embedding dimension.

m Number of rows of $\widetilde{A}$ and number of columns of S.

n Number of columns of A, and dimension of β. Thus $A \in \mathbb{R}^{m \times n}$ and $\beta \in \mathbb{R}^n$.

ζ Number of nonzeros per column in the sketching matrix. Sketch sparsity parameter.

$\kappa(\cdot)$ Condition number of a matrix.

$\sigma_{\max}(\cdot)$ Largest singular value.

$\sigma_{\min}(\cdot)$ Smallest singular value.

r_i Residual at iteration i in iterative sketching.

c_i Normal-equation residual (gradient-like vector), typically $c_i \leftarrow A^\top r_i$.

p_i Intermediate vector from a triangular solve, typically solving $R^\top p_i = c_i$.

q_i Correction step, typically obtained from $Rq_i = p_i$, then used in $\beta_{i+1} = \beta_i + q_i$.

ϵ Accuracy parameter also called embedding distortion. Distortion is usually bounded by $(1 \pm \epsilon)$.

δ_{fail} Probability sketching fails to reach an embedding distortion. An embedding holds with probability at least $1 - \delta_{\text{fail}}$.

g_{IS} Convergence factor for iterative sketching. Typically one requires $g_{\text{IS}} < 1$ for convergence.

τ Target relative residual accuracy bound, for Sketch-and-Solve $\|r(\beta_i)\| \leq (1 + \tau)\|r(\beta^\star)\|$.

Chapter 1

Introduction

Random Fourier Neural Networks (rFNNs) are neural network architectures designed to approximate target functions using complex exponential activation functions. In this setting, a target function is represented using sampled Fourier modes, and the contribution of each sampled mode is determined by an amplitude parameter. This gives rFNNs a natural connection to Fourier analysis and makes them well suited for representing oscillatory, high-frequency, and multiscale structure [2].

The central computational problem considered in this thesis is the least squares problem that arises when computing these amplitude parameters. Once a set of frequencies has been sampled, the remaining task is to determine the amplitudes that best fit the target function values at a given set of sample points. This leads to an overdetermined least squares problem, typically with Tikhonov regularization. Since this type of least squares problem can be large, ill-conditioned, and repeatedly encountered in rFNN computations, solving it efficiently is an important numerical task.

The goal of this thesis is to study randomized sketching methods for this least squares problem. In particular, we compare Sketch-and-Solve, Sketch-and-Precondition, and Iterative Sketching against Householder QR. We summarize the relevant subspace embedding theory for Clarkson-

Woodruff sketching and evaluate these methods experimentally in terms of conditioning, embedding distortion, runtime, forward error, residual error, and backward error. For controlled analysis, we restrict attention to the least squares problem arising from one approximation step of a one-layer rFNN. The same least squares structure, however, appears more broadly in rFNN computations.

The main contribution of this thesis is a focused analysis of sketching methods for the specific least squares problems that arise in rFNNs. We formulate the rFNN amplitude-fitting problem in a way that allows direct comparison of the solvers, and we evaluate these methods both theoretically and experimentally in the context of randomized Fourier feature approximation. This provides application-specific evidence for when sketching is effective and when it is not.

Section 1.1 introduces the regularized least squares problem and augmented system studied throughout the thesis. Section 1.2 derives the connection between this problem and rFNNs. The remainder of the thesis studies randomized least squares solvers for that system.

1.1 Least Squares Problem

The least squares problem that we are studying can be expressed as the following unconstrained convex optimization problem:

$$\min_{\beta \in \mathbb{R}^n} \ \|y - A\beta\|_2^2 + \lambda \|\beta\|_2^2.$$

Here, $A \in \mathbb{R}^{(m-n)\times n}$ is a given design matrix, $y \in \mathbb{R}^{m-n}$ is a given right-hand-side vector, $\beta \in \mathbb{R}^n$ is the unknown vector of parameters, and $\lambda > 0$ is a (small) Tikhonov regularization parameter, with $m > 2n$. This can be equivalently expressed as

$$\min_{\beta \in \mathbb{R}^n} \ \|\widetilde{y} - \widetilde{A}\beta\|_2^2, \qquad \widetilde{A} := \begin{bmatrix} A \\ \sqrt{\lambda}\, I_n \end{bmatrix} \in \mathbb{R}^{m\times n}, \qquad \widetilde{y} := \begin{bmatrix} y \\ 0_n \end{bmatrix} \in \mathbb{R}^m. \tag{1.1.1}$$

In practice, this least squares problem is highly overdetermined, meaning that the number of rows m is much greater than the number of columns n (i.e., $m \gg n$).

Challenges. There are three main numerical challenges associated with problems of the form (1.1.1). First, the problems we are interested in are of large scale, where m may be on the order of millions, while n may be on the order of tens or hundreds. Second, although the augmented design matrix $\tilde{A} \in \mathbb{R}^{m \times n}$ is assumed to be full rank, it may have a large condition number

$$\kappa\left(\widetilde{A}\right) = \frac{\sigma_{max}(\widetilde{A})}{\sigma_{min}(\widetilde{A})} \gg 1,$$

where $\sigma_{max}(\widetilde{A})$ and $\sigma_{min}(\widetilde{A})$ are the largest and smallest singular values of $\widetilde{A}$, respectively [12]. Generally, Tikhonov regularization is used to improve the conditioning of this least squares problem [1], but it also introduces a small amount of bias depending on the value of the regularization constant. This means that the least squares problem needs to be solved within an error tolerance on the order of the chosen $\lambda > 0$. Third, in some randomized machine learning applications such as rFNNs (discussed in section 1.2), we may need to repeatedly solve least squares problems of the form (1.1.1) with different constructions of the design matrix.

Goal. The goal of this thesis is to solve least squares problems of the form (1.1.1) efficiently and accurately. Standard direct methods can compute accurate solutions, but they become prohibitively expensive for solving many very tall systems. The methods studied in this thesis seek to reduce this cost by replacing the original problem with smaller or better-conditioned randomized least squares problems.

1.2 Motivation from rFNNs

Random Fourier Neural Networks are designed to approximate a target function $Q = Q(\theta)$, for $\theta \in \mathbb{R}$, from a given training data set $\{\theta_i, Q(\theta_i)\}_{i=1}^{N}$ using complex exponential activation functions. One way to motivate this choice of activation function is through the Fourier integral representation of a function. Precisely, we assume that the target function Q is continuous and

has the integral representation

$$Q(\theta) = \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} \widehat{Q}(\omega) e^{i\omega\theta} \, d\omega,$$

where $\widehat{Q}$ is the Fourier transform of Q [4]. By Euler's formula,

$$e^{i\omega\cdot\theta} = \cos(\omega \cdot \theta) + i \sin(\omega \cdot \theta),$$

which expresses the target function as a continuous superposition of sinusoidal modes. In this interpretation, ω is a frequency variable and $\widehat{Q}(\omega)$ describes the contribution of that frequency to the target function.

This integral representation also motivates the sampling viewpoint used in randomized Fourier methods. In such methods, we view the frequency ω as a random variable following a probability distribution $p(\omega)$. By importance Monte Carlo sampling [9], and assuming $p(\omega)$ is a probability density that is positive wherever $\widehat{Q}(\omega)$ is nonzero, the Fourier integral can be rewritten as an expectation:

$$Q(\theta) = \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} \widehat{Q}(\omega) e^{i\omega\theta} \frac{p(\omega)}{p(\omega)} \, d\omega = \mathbb{E}_{\omega\sim p} \left[\frac{\widehat{Q}(\omega)}{\sqrt{2\pi} p(\omega)} e^{i\omega\theta} \right].$$

By [8], a Monte Carlo approximation of this expectation using $W \geq 1$ independent sampled frequencies $\omega_1, \ldots, \omega_W \sim p(\omega)$ gives

$$Q(\theta) \approx \frac{1}{W} \sum_{j=1}^{W} \frac{\widehat{Q}(\omega_j)}{\sqrt{2\pi} p(\omega_j)} e^{i\omega_j\theta}.$$

Thus, a finite set of sampled frequencies gives a finite Fourier feature approximation of the target function. In practice, the Fourier transform $\widehat{Q}$ and an optimal sampling density p are generally not known exactly. The rFNN training procedure therefore samples frequencies and then solves for amplitudes that best fit the target data. This leads to the finite model

$$Q(\theta) \approx \Re \left[\sum_{j=1}^{W} \beta_j e^{i\omega_j\theta} \right],$$

where the sampled frequencies ω_j determine the Fourier modes available to the model, and the amplitudes β_j determine how strongly those modes contribute. To visualize the architecture of these networks and how they use this finite model, a diagram representing a depth-1 rFNN with W neurons is provided in figure 1.1.

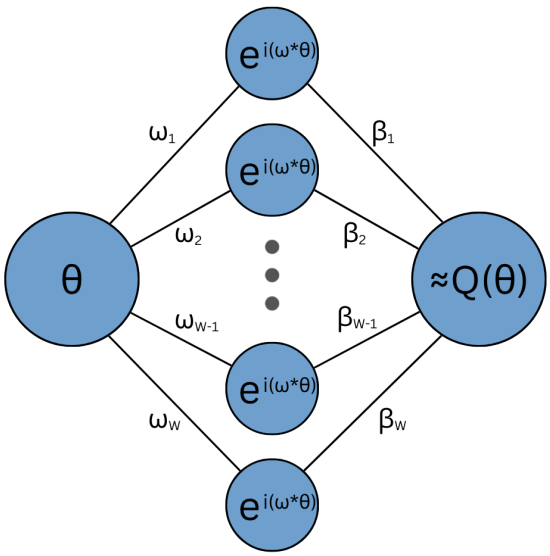

Figure 1.1: Depth-1 rFNN with W neurons.

For each sampled frequency ω_j, the corresponding complex exponential feature is

$$e^{i\omega_j \cdot \theta}.$$

Write the corresponding complex amplitude as

$$\beta_j = a_j + ic_j,$$

where $a_j, c_j \in \mathbb{R}$. Since the target function is real-valued, the contribution of this feature is taken to be the real part:

$$\Re\big(\beta_j e^{i\omega_j \cdot \theta}\big) = \Re((a_j + ic_j)(\cos(\omega_j \cdot \theta) + i\sin(\omega_j \cdot \theta))) = a_j \cos(\omega_j \cdot \theta) - c_j \sin(\omega_j \cdot \theta).$$

Thus, each frequency contributes two real-valued features: a cosine feature and a negative sine feature. The corresponding unknowns are the real and imaginary parts of the complex amplitude.

Evaluating the model at the sampled points $\theta_1, \ldots, \theta_N$ gives a linear least squares problem for the unknown amplitudes. Specifically, the design matrix contains the cosine and negative sine features evaluated at the training points, the right-hand side contains the target values $Q(\theta_i)$, and the unknown vector contains the real and imaginary amplitude components. Figure 1.2 gives a visual representation of what the unregularized problem looks like.

$$\begin{bmatrix} \cos(\omega_1\theta_1) & -\sin(\omega_1\theta_1) & \cdots & \cos(\omega_W\theta_1) & -\sin(\omega_W\theta_1) \\ \cos(\omega_1\theta_2) & -\sin(\omega_1\theta_2) & \cdots & \cos(\omega_W\theta_2) & -\sin(\omega_W\theta_2) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \cos(\omega_1\theta_N) & -\sin(\omega_1\theta_N) & \cdots & \cos(\omega_W\theta_N) & -\sin(\omega_W\theta_N) \end{bmatrix} \begin{bmatrix} a_1 \\ c_1 \\ \vdots \\ a_W \\ c_W \end{bmatrix} \approx \begin{bmatrix} Q(\theta_1) \\ Q(\theta_2) \\ \vdots \\ Q(\theta_N) \end{bmatrix}$$

Figure 1.2: Least squares problem arising from a depth-1 rFNN. The matrix A contains cosine and negative sine Fourier features evaluated at the training points, y contains the sampled target function values, and β contains the unknown amplitudes. Adding Tikhonov regularization to this amplitude-fitting problem gives the regularized least squares problem studied in this thesis.

This sampling-and-amplitude-fitting framework is developed in [2]. This thesis does not study the frequency sampling procedure itself. Instead, it studies numerically solving the least squares problem that appears once a set of frequencies has been sampled.

Chapter 2

Sketching-Based Least Squares Solvers for Overdetermined Systems

In a highly overdetermined system, the number of equations is far larger than the number of unknowns, but the solution depends primarily on the geometry of a low-dimensional subspace determined by the columns of the system matrix. Randomized sketching methods exploit this fact by compressing the problem to a smaller number of rows while approximately preserving the system's column space [13].

Sketching methods typically do this by constructing a sketching matrix S that when multiplied to the left of some system $A\beta = y$, result in a smaller system $SA\beta = Sy$ where SA has the same number of columns as A, but many fewer rows. The number of rows in SA depends on the number of rows in S, sometimes called the sketch dimension, which is taken as a parameter we will call d.

Clarkson-Woodruff sketching, described in depth in [13], is one sketching method and it constructs the sketching matrix S with d rows and the same number of columns as A has rows so that the two matrices can be multiplied. S is taken to have all entries as zero except for a given number per column ζ which are taken to be either $\frac{1}{\sqrt{\zeta}}$ or $-\frac{1}{\sqrt{\zeta}}$ with equal probability. This

parameter ζ is also known as sketch sparsity.

Clarkson-Woodruff sketching is an especially attractive sketching method because it utilizes sparse matrix multiplication to efficiently compute a smaller problem that retains most of the essential information of the original problem with high probability. For this reason, we will be using Clarkson-Woodruff sketching as our sketching method of choice, although it is not the only sketching technique.

Sketching can be used in a variety of methods, including least squares solvers for similar problems to the one we are studying. The sketching-based methods we study in this thesis, which we describe through the rest of this section, are Sketch-and-Solve, Sketch-and-Precondition, and Iterative Sketching.

2.1 Sketch-and-Solve

The Sketch-and-Solve algorithm is the simplest approach to reduce computational cost of overdetermined least squares problems using sketching. This method uses sketching to reduce the size of the least squares problem then solves this smaller system with a standard least squares solver such as Householder QR.

The performance of this method is heavily dependent on parameter selection. While we discuss the theoretical guarantees in relation to parameter selection later in section 3.1, it is common to pick $\zeta = 8$ for sketch sparsity because of empirical success according to [3]. The sketch dimension d can vary substantially depending on the specific problem, but it has to be some number significantly smaller than the number of rows in A so that the sketched problem is still quicker to solve than the original when accounting for the time it takes to do the sketching.

2.2 Sketch-and-Precondition

The second sketching-based method we discuss in this thesis is the sketch-and-precondition algorithm which utilizes sketching to improve the LSQR iterative least squares solver. LSQR is based on Golub-Kahan bidiagonalization and given a generalized least squares problem $A\beta \approx y$, it constructs approximate solutions using only matrix-vector products with A and A^T. This makes it well suited for large problems where forming a full matrix factorization is computationally expensive. A detailed derivation of the algorithm is beyond the scope of this thesis; see [7] for the original development of LSQR.

The convergence of iterative least squares methods is closely tied to the conditioning of the matrix. Once again, for a full column rank matrix A, the condition number is defined as

$$\kappa(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)},$$

where $\sigma_{\max}(A)$ and $\sigma_{\min}(A)$ are the largest and smallest singular values of A respectively [12]. A large condition number means that the columns of A are nearly linearly dependent, or equivalently that the least squares problem is sensitive to perturbations in the data. In the rFNN setting, this can occur when sampled frequencies are close together, producing nearly dependent columns in the feature matrix.

Conditioning is important for LSQR because poorly conditioned systems generally require more iterations to reach a given accuracy. Although LSQR avoids explicitly forming the normal equations A^TA, and therefore avoids directly squaring the condition number, its convergence behavior is still strongly affected by the singular value distribution of A. As a result, LSQR can be efficient for well-conditioned least squares problems but slow for ill-conditioned ones. This motivates the use of sketch-and-precondition, where a randomized sketch is used to construct a preconditioner that improves the conditioning of the system before applying LSQR.

To construct this preconditioner for some system $A\beta \approx y$, we take the reduced QR factorization of A and form AR^{-1}. For $A \in \mathbb{R}^{m\times n}$, the QR factorization gives $R \in \mathbb{R}^{n\times n}$ which is

upper triangular and $Q \in \mathbb{R}^{m \times n}$ which has orthonormal columns. For an overdetermined system where $m \gg n$, computing AR^{-1} is very efficient because R is an $n \times n$ matrix so it is small, and it is upper triangular so computing its inverse is cheap. The resulting matrix AR^{-1} then has a condition number of 1 because of the following.

$$A = QR$$

$$A^T A = (QR)^T QR = R^T Q^T QR$$

and since Q has orthonormal columns, $Q^T Q = I$

$$A^T A = R^T R$$

now take $\beta = R^{-1} z$ and our system becomes $AR^{-1} z = y$

$$\begin{aligned}(AR^{-1})^T (AR^{-1}) &= R^{-T} A^T A R^{-1} \\ &= R^{-T} (R^T R) R^{-1} \\ &= I\end{aligned}$$

This shows that $(AR^{-1})^T (AR^{-1}) = I$ thus AR^{-1} has orthonormal columns so it has a condition number of 1. This means that if you substitute $\beta = R^{-1} z$ into the system to get $(AR^{-1}) z \approx y$, LSQR will converge quickly to give us the solution to the preconditioned system z. Then we can solve the triangular system $\beta = R^{-1} z$ to get our final solution β. The problem with forming this preconditioner is that forming QR is very expensive, so the runtime of this algorithm ends up being slower than directly solving the problem with QR factorization. This is where sketching comes in. If you use sketching to get $SA = QR$, you are factoring a much smaller system and you still maintain the size of R. Since this factorization is substituted in for preconditioning and then substituted back out for the final solution, using this sketched factorization means that the columns of AR^{-1} are approximately orthonormal and AR^{-1} has a condition number near 1, while the final solution is not significantly impacted. This algorithm is succinctly expressed in algorithm 1.

Algorithm 1 Sketch-and-Precondition

1: $S \leftarrow d \times m$ (sketching matrix)
2: $(Q, R) \leftarrow \mathrm{QR}(SA)$ (economy QR)
3: $z_0 \leftarrow Q^{\mathsf{T}}(Sy)$ (LSQR initial condition)
4: $R^{-1}z \leftarrow \beta$
5: solve $(AR^{-1})z = y$ (preconditioned LSQR)
6: $\beta \leftarrow R^{-1}z$

This algorithm iterates until a desired stopping tolerance is met as described in [10]. This algorithm has shown potential in the literature [3, 5] to significantly decrease the runtime of solving least squares problems while maintaining accuracy. With that being said, its numerical stability is still in question. This algorithm is shown to be possibly unstable with an initial condition of zero, but has improved stability with the initial condition $z_0 \leftarrow Q^{\mathsf{T}}(Sy)$.

2.3 Iterative Sketching

Similarly to the sketch-and-precondition algorithm, iterative sketching uses a sketching matrix to economically form a QR factorization and uses this to find an initial condition. From here, the algorithm iteratively approximates β by taking the residual r_i of the current solution to get $c_i \leftarrow A^T r_i$. Then the algorithm does two triangular solves to get q_i such that $\beta_{i+1} \leftarrow \beta_i + q_i$. This is described in algorithm 2.

Algorithm 2 Iterative Sketching

$S \leftarrow d \times m$ (sketching matrix)
$(Q, R) \leftarrow \mathrm{QR}(SA)$ (economy QR)
$\beta_0 \leftarrow R^{-1}\big(Q^{\mathsf{T}}(Sy)\big)$ (initial condition)
for $i = 0, 1, \ldots,$ `iter_lim` $- 1$ **do**
  $r_i \leftarrow y - A\beta_i$
  $c_i \leftarrow A^T r_i$
  if $\|r_i\|_2 \leq$ `atol` $\|A\|_2\|\beta_i\|_2 +$ `btol` $\|y\|_2$ **then**
    break
  end if
  if $\|c_i\|_2 \leq$ `atol` $\|A\|_2\|r_i\|_2$ **then**
    break
  end if
  solve $R^T p_i = c_i$
  solve $Rq_i = p_i$ (two triangular solves)
  $\beta_{i+1} \leftarrow \beta_i + q_i$
end for

We iterate until the same stopping condition as sketch-and-precondition is met. This algorithm has shown similar potential in literature to decrease the runtime of solving least squares problems while maintaining accuracy. Compared to the sketch-and-precondition algorithm, literature suggests slower convergence, but stronger theoretical guarantees [3].

Chapter 3

Theoretical Guarantees

Here we will look at the theoretical guarantees of each of these methods. The core assumption behind all three methods is that S is an l_2 subspace embedding for the column space of A [13]. For a target distortion factor ϵ, we have:

$$(1-\epsilon)\,\|Ax\|_2 \;\leq\; \|SAx\|_2 \;\leq\; (1+\epsilon)\,\|Ax\|_2, \qquad \forall x \in \mathbb{R}^n. \tag{3.0.1}$$

It is typical to assume the same embedding holds for the slightly larger space $\mathrm{range}([A\ y])$ [13, 6].

The embedding given by the inequality in 3.0.1 has a failure probability of δ_{fail}. This can be expressed as

$$\mathbb{P}\,(\text{embedding holds}) = 1 - \delta_{\text{fail}}.$$

According to theorem 9 in [13], with $\zeta = 1$, a given value of n (the number of columns in A), and a desired δ_{fail} and ϵ, you asymptotically choose d such that it satisfies the following:

$$d = O\left(\frac{n^2}{\delta_{\text{fail}}\epsilon^2}\right) \tag{3.0.2}$$

You can also get somewhat sharper conditions for d given in [6] by increasing the number of non-zero rows per column of the sketching matrix ζ. Their Theorem 9 gives the condition

$$d = \Omega\left(\frac{n \times \log^8(n/\delta_{\text{fail}})}{\epsilon^2}\right) \quad \text{with} \quad \zeta = \Theta\left(\frac{\log^3(n/\delta_{\text{fail}})}{\epsilon}\right). \tag{3.0.3}$$

3.1 Sketch-and-Solve Guarantees

The sketch-and-solve algorithm computes

$$\beta_0 \;=\; \arg\min_{\beta\in\mathbb{R}^n} \;\|Sy-(SA)\beta\|_2,$$

typically with Householder QR on SA. Take $\beta^\star$ to be the exact minimizer of $\min_\beta \|A\beta - y\|_2$ and define $r(\beta) = y - A\beta$. If S is a subspace embedding for $\mathrm{range}([A\ y])$ with distortion ϵ, then sketch-and-solve satisfies the following guarantee for its residual [3]:

$$\|r(\beta_0)\|_2 \;\le\; \frac{1+\epsilon}{1-\epsilon}\,\|r(\beta^\star)\|_2.$$

Fact 2.4 of [3] goes on to prove the following bounds for how close the residual and solution are to being optimal:

$$\|r(\beta^\star)-r(\beta_0)\|_2 \;\le\; \frac{2\sqrt{\epsilon}}{1-\epsilon}\,\|r(\beta^\star)\|_2, \qquad \|\beta^\star-\beta_0\|_2 \;\le\; \frac{2\sqrt{\epsilon}}{1-\epsilon}\,\kappa(A)\,\frac{\|r(\beta^\star)\|_2}{\|A\|_2}.$$

So, for a desired relative residual accuracy $\tau > 0$ (i.e., $\|r(\beta_0)\|_2 \le (1+\tau)\|r(\beta^\star)\|_2$), you should choose $\epsilon \le \tau/(2+\tau)$ so that $(1+\epsilon)/(1-\epsilon) \le 1+\tau$.

There are two major components of this runtime: forming the sketched system and solving it. If $\zeta = 1$, forming the sketched system can be done by processing each nonzero entry of the original system once. More generally, for a sparse sign embedding with ζ nonzeros entries per column, and the number of nonzero entries in A, $\mathrm{nnz}(A)$, forming the sketch costs $O(\zeta \times \mathrm{nnz}(A))$ which follows from the embedding structure and sparse matrix multiplication cost [3]. After sketching, one solves the smaller least squares problem. Using Householder QR on a $d \times n$ sketched system costs $O(dn^2)$. Which gives the end-to-end cost,

$$O(\zeta\,\mathrm{nnz}(A)) + O(dn^2).$$

3.2 Sketch-and-Precondition Guarantees

The preconditioned system is $(AR^{-1})z \approx y$, followed by $\beta = R^{-1}z$. With the subspace embedding condition in 3.0.1, the singular values and consequently condition number of AR^{-1} are bounded by Fact 2.3 of [3] as

$$\sigma_{\max}(AR^{-1}) \leq \frac{1}{1-\epsilon}, \qquad \sigma_{\min}(AR^{-1}) \geq \frac{1}{1+\epsilon}, \qquad \kappa(AR^{-1}) = \frac{\sigma_{\max}(AR^{-1})}{\sigma_{\min}(AR^{-1})} \leq \frac{1+\epsilon}{1-\epsilon}.$$

Since the number of iterations LSQR takes to converge is governed by the condition number of the problem, a small tolerance of ϵ implies rapid convergence to a requested tolerance. In practice, this method becomes practical for our least squares problem when $m \gg n$, since forming SA and factoring it can be much cheaper than factoring A directly, while still giving a high-quality preconditioner. In exact arithmetic, the preconditioner primarily changes the convergence rate; in floating point arithmetic, however, standard sketch-and-precondition may still exhibit numerical instability for ill-conditioned problems [5].

3.3 Iterative Sketching Guarantees

Iterative sketching uses the same sketched QR factorization $SA = QR$ and starts from the initialization $\beta_0 = R^{-1}Q^\top(Sy)$ [3]. Then it does iterative updates which are described by the following:

$$(SA)^\top(SA)\, d_i = A^\top(y - A\beta_i), \qquad \beta_{i+1} = \beta_i + d_i.$$

In theorem 3.1 of [3], if S is a subspace embedding with distortion $0 < \epsilon < 1 - 1/\sqrt{2}$, then iterative sketching converges geometrically and the following is true:

$$\|\beta^\star - \beta_i\|_2 \;<\; (8-2\sqrt{2})\sqrt{\epsilon}\,\kappa(A)\, g_{\mathrm{IS}}^i\, \frac{\|r(\beta^\star)\|_2}{\|A\|_2}, \quad \|r(\beta^\star) - r(\beta_i)\|_2 \;<\; (8-2\sqrt{2})\sqrt{\epsilon}\, g_{\mathrm{IS}}^i\, \|r(\beta^\star)\|_2,$$

where the convergence factor is

$$g_{\mathrm{IS}} \;=\; \frac{(2-\epsilon)\epsilon}{(1-\epsilon)^2} \;\leq\; (2+\sqrt{2})\,\epsilon \;<\; 1.$$

Thus,

$$\|r(\beta_i)\|_2 \;\leq\; \|r(\beta_\star)\|_2 + \|r(\beta_\star) - r(\beta_i)\|_2 \;<\; \|r(\beta_\star)\|_2\Big(1 + (8 - 2\sqrt{2})\sqrt{\epsilon}\, g_{\mathrm{IS}}^i\Big).$$

So to achieve a target relative residual $\|r(\beta_i)\|_2 \leq (1+\tau)\|r(\beta^\star)\|_2$, you should choose a number of iterations i, where

$$i \;\geq\; \frac{\log((8 - 2\sqrt{2})\sqrt{\varepsilon}/\tau)}{\log(1/g_{\mathrm{IS}})}.$$

This superior theoretical guarantee over the sketch-and-precondition algorithm comes at the cost of requiring a stronger embedding where ϵ must be small enough such that $g_{\mathrm{IS}} < 1$. This means that $\epsilon < 1/(2+\sqrt{2}) \approx 0.293$, to ensure good convergence [3]. This means that d has a higher lower bound according to equation 3.0.2, which if not met, $g_{\mathrm{IS}} \geq 1$ so the term g_{IS}^i grows exponentially with the number of iteration i. This also means that the forward and residual error bounds no longer guarantee convergence.

Chapter 4

Numerical Studies

This section starts by explaining the test design we use and how we measure performance. Next, we will experimentally study the suitability of the least squares solvers we have mentioned for our specific problem. Firstly, this includes studying Sketch-and-Solve's behavior by itself across different embedding dimensions to motivate the use of the other sketching algorithms that build off of it. Secondly, we compare Sketch-and-Precondition and Iterative Sketching to Householder QR across these same embedding dimensions. Lastly, we study how Sketch-and-Precondition and Iterative Sketching scale in number of input samples N compared to Householder QR.

4.1 Test Design

In order to study the suitability of each of these least squares solvers for training rFNNs, we will be looking at the least squares problem that arises from a single training iteration of a network with a single layer. We will be testing this with the target function

$$Q(\theta) = \cos(4\theta) + 0.3\cos(70\theta) + 0.05\cos(150\theta), \qquad \theta \in [-1, 1] \tag{4.1.1}$$

because it is a simple example of a multiscale sinusoidal function. All tests were also performed on

$$Q(\theta) = \mathrm{Si}\left(\frac{\theta}{a}\right) e^{-\theta^2/2}, \qquad \mathrm{Si}(x) := \int_0^x \frac{\sin t}{t}\, dt, \qquad a = 10^{-3}, \quad \theta \in [-1, 1] \tag{4.1.2}$$

as the target function because it has a slowly decaying Fourier transform which means the amplitudes corresponding to many frequencies are large [4]. Those results are included in appendix A but not here because they were trivially different.

Our initial studies in sections 4.2 and 4.3 sweep sketch dimensions from $d = 1,000$ to $d = 50,000$ with intervals of $1,000$. Later on in section 4.4, we fix a sketch dimension and sweep problem sizes by increasing the number of input samples from $N = 50,000$ to $N = 10^6$ with intervals of $50,000$. For our initial studies, we take the number of samples to be $N = 50,000$ and the width of the network to be $W = 50$, so the matrix $\widetilde{A}$ has $50,100$ rows and 100 columns. The metrics that we are primarily looking at are computational efficiency and error. Further discussion of these metrics is done later in sections 4.1.1 and 4.1.2.

We also take our Tikhonov regularization parameter to be $\lambda = 10^{-6}$. It is important to note that once the least squares solve is accurate below the effect introduced by λ, further numerical accuracy is unlikely to change the rFNN training behavior meaningfully. This means the convergence we are looking for is to this constant, not machine precision. We take the number of non-zero entries per column in our sketching matrices to be $\zeta = 8$ by the recommendation and experimental results of [3].

To account for randomness from frequency sampling, we form five different least squares problems from each given problem and test on each of them. The most significant way randomness can skew the results in practice is by giving a problem with an outlier condition number so we discuss problem conditioning for each of our studies. Additionally, to account for randomness in the Sketch-and-Precondition and Iterative Sketching algorithms, we run five trials on each problem. The most significant way randomness can skew the results here is by giving better or worse embeddings from sketching which is one of the most important factors for the accuracy of

Sketch-and-Solve as well as convergence rates of Sketch-and-Precondition and Iterative Sketching. We measure this by the embedding distortion of each sketch which we discuss for each study. We accumulate all these results to get median results along with upper and lower bounds.

Since both Sketch-and-Precondition and Iterative Sketching use iterative methods, they require stopping conditions. For the LSQR solve in Sketch-and-Precondition, we use SciPy's implementation which takes two tolerances: $atol$ and $btol$, as well as an iteration cap $iter\_lim$. These conditions terminate when either the residual is sufficiently small such that

$$\|r\|_2 \leq atol\|A\|_2\|\beta\|_2 + btol\|y\|_2,$$

or when the normal-equations residual is sufficiently small such that

$$\|A^\top r\|_2 \leq atol\|A\|_2\|r\|_2,$$

where $r = y - A\beta$. For Iterative Sketching, we use the same two stopping conditions to maintain consistency between the methods. SciPy states in their documentation that these tolerances are best interpreted as estimates of the relative error in the entries of A and y (using our notation) [10]. For example, if the entries of A have 7 correct digits, set the tolerances to 10^{-7}. For all our tests, we take both these tolerances to be 10^{-9} to allow some leeway for all our error metrics to get sufficiently small.

All of these tests are done in Python version 3.13.9 with an Intel(R) Core(TM) i9-14900HX processor and 31.7GB of memory. Since some Python libraries such as NumPy use implicit parallelization, which can potentially skew results, we explicitly use single-threading for all our studies.

4.1.1 Error Analysis

To study the accuracy of different algorithms, we will be looking at forward, residual, and backward errors which are taken from [3]. Here, forward error is measuring the normalized difference in the

solution β, residual error is measuring the normalized difference in the residual ($r(\beta) = \widetilde{y} - \widetilde{A}\beta$), and backward error is measuring how close our problem is to the problem that our solution is exact for. These errors are respectively given by

$$\mathrm{FE}(\beta) := \frac{\|\beta^\star - \beta\|_2}{\|\beta^\star\|_2}, \tag{4.1.3}$$

$$\mathrm{RE}(\beta) := \frac{\|r(\beta^\star) - r(\beta)\|_2}{\|r(\beta^\star)\|_2}, \tag{4.1.4}$$

$$\mathrm{BE}(\beta) := \min\left\{ \frac{\|\Delta\widetilde{A}\|_F}{\|\widetilde{A}\|_F} : \beta = \arg\min_v \ \|\widetilde{y} - (\widetilde{A} + \Delta\widetilde{A})v\|_2 \right\}. \tag{4.1.5}$$

An important note is forward and residual errors measure distance from the optimal solution which in our tests is taken to be the solution using QR and thus it doesn't make sense to track the forward or residual errors of QR since they are defined as zero.

The forward error is most directly studying the difference in the amplitude between an optimal least squares solution and the one we get. This solution β is what changes the contribution of each frequency. Despite this, backward error is typically considered the "gold standard" because it bounds the forward and residual errors as discussed in [3].

4.1.2 Computational Efficiency

To study the computational efficiency of different algorithms, we will look at their runtimes on the same sets of problems. We accumulate the results from all these problems to get median runtimes as well as upper and lower bounds to show variance. Each method is timed from its start to the finish as it would run in practice, including the algorithms that have sketching as well as other operations. For each of the sketching methods, we independently create a random number generator and use it to create the sketching matrix, then complete the rest of each method. Because each of these sketching matrices are formed separately but with identically formed random

number generators and parameters, the resulting sketches and embedding distortion factors are identical. This is not because of reuse of the sketching matrices throughout the methods, rather because of the operations being standardized across methods.

Our implementation of Householder QR is given by NumPy which has been heavily optimized over many years to have close to the lowest computational cost that is possible with the method. Our high-level implementations of Sketch-and-Solve, Sketch-and-Precondition, and Iterative Sketching are all done by us and utilize NumPy and SciPy functions for the required computations. Because of this, Householder QR likely has a more efficient implementation which gives it an advantage in numerical experiments. To make our implementations as efficient as possible, we take the following steps.

- For sketching to be efficient, the sketching matrix S must be stored and applied as a sparse matrix. We first construct S in coordinate format (COO), which stores a sparse matrix as a list of $(\text{row}, \text{column}, \text{value})$ triples. This is convenient because the Clarkson-Woodruff sketch is generated by choosing the nonzero entries of each column. That being said, COO is mainly useful for constructing sparse matrices, not for multiplying them. Before applying the sketch, we convert S to compressed sparse row format (CSR), which stores the nonzero entries row-by-row and is more efficient for sparse matrix multiplication. COO is used to build the sketching matrix, while CSR is used to compute SA and Sy since converting between these formats is very inexpensive [11].
- For our Sketch-and-Precondition algorithm, we do not explicitly form AR^{-1}. Instead, we solve the triangular system with R^{-1} then multiply by A to reduce work with dense matrices.
- When computing QR factorizations, we used NumPy's reduced QR factorization instead of a full one because it saves computation and memory.

4.2 Sketch-and-Solve

The most simple algorithm we are studying and the one that the others rely on is the Sketch-and-Solve algorithm. By itself, this algorithm does not have the strong enough theoretical or experimental results to be used for precise tasks, however it is still important to establish its efficacy for use in the Sketch-and-Precondition and Iterative Sketching algorithms.

4.2.1 Problem Conditioning and Embedding Distortion

As mentioned previously, we create five different least squares problems given the same inputs to account for randomness in the frequency sampling algorithm. As conditioning is the most significant factor we are controlling for here, we have the condition numbers of each $\widetilde{A}$ used for this study plotted in figure 4.1. All of these problems are very similarly conditioned with condition numbers between 1.37×10^6 and 1.44×10^6. This suggests that these problems give a representative pool of the behavior we would expect to see out of our specific problem.

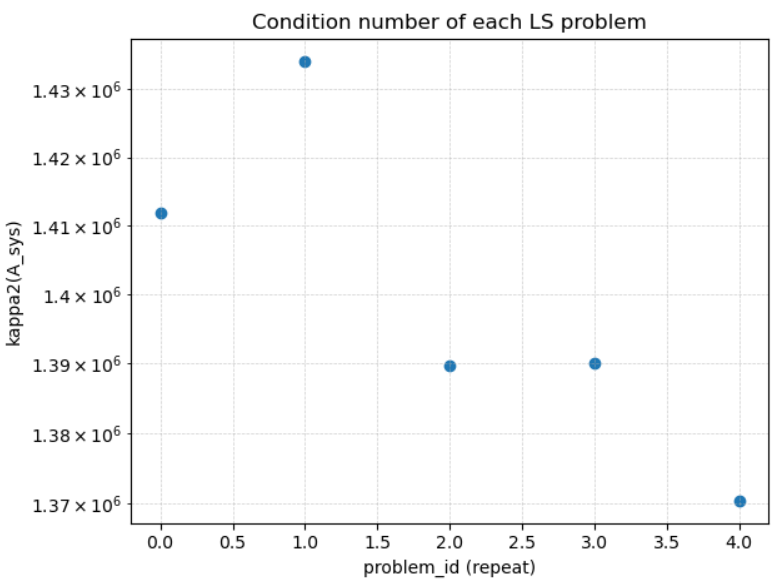

Figure 4.1: Condition numbers of sample least squares problems arising from one iteration of network training.

We do five trials for each problem and do this for each embedding dimension (d) we study. This means there are twenty-five sketches for each embedding dimension. To look at how effective sketching is for this problem, we have plotted the embedding distortion of each sketch in figure 4.2. This shows that for very small embedding dimensions, the distortion can be excessive, but quickly decreases below 0.3 and 0.2 in many cases. While this range is far too much for direct use as done in the Sketch-and-Solve algorithm, this is theoretically adequate for the other algorithms as discussed in sections 3.2 and 3.3.

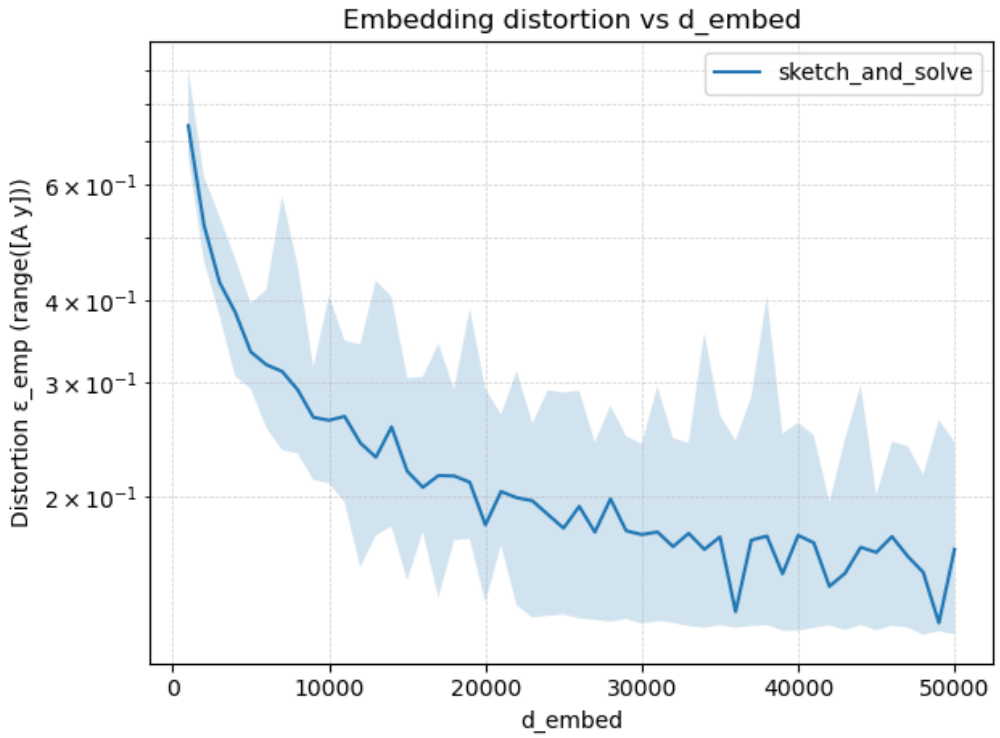

Figure 4.2: Empirical embedding distortion from Sketch-and-Solve least squares solutions compared to Householder QR across different sketching dimensions.

4.2.2 Error Analysis

Next, we look at the accuracy of the Sketch-and-Solve algorithm. The forward, residual, and backward errors we observed from our tests are plotted in figure 4.3. Considering the error tolerance we are looking for given by our Tikhonov regularization constant is 10^{-6}, it is evident that this method is not sufficiently accurate. While the backward error stays around 10^{-2}, the forward error stays above 1 for the most part, and in the worst cases, gets well into the order of 10^{1}. Error this high, especially in the forward error, would greatly change the contribution of each frequency in an rFNN, thus the Sketch-and-Solve algorithm is not suitable for our application by itself.

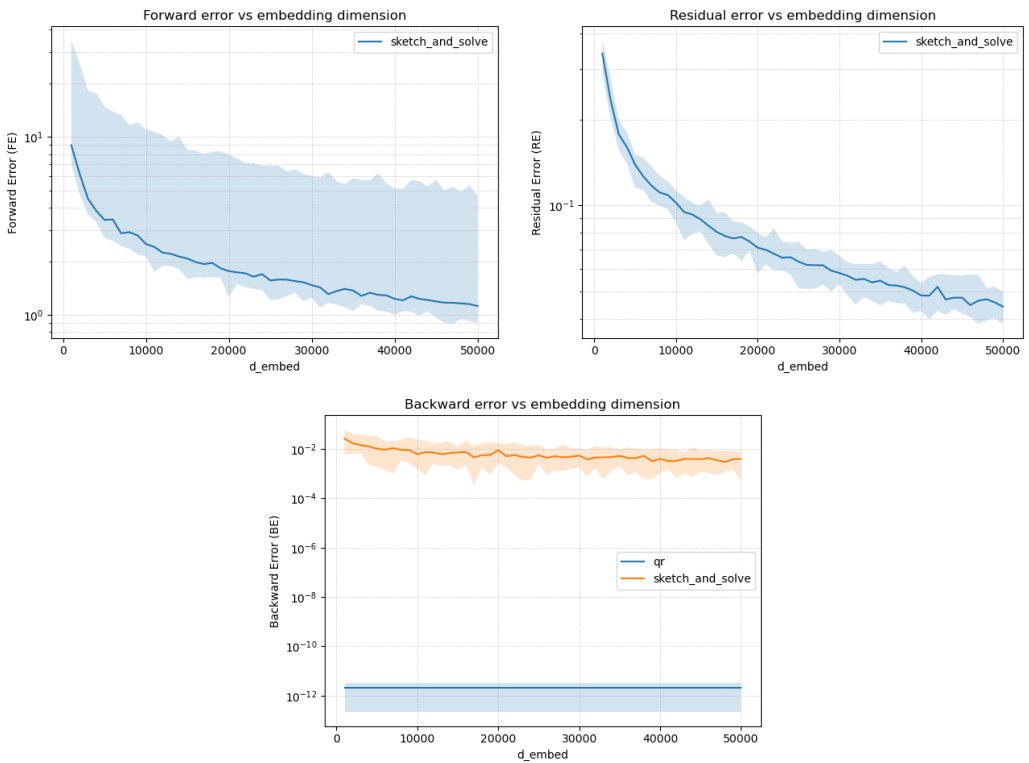

Figure 4.3: Forward, residual, and backward errors from Sketch-and-Solve least squares solutions compared to Householder QR across different sketching dimensions. The QR forward and residual errors aren't graphed because they are defined to be zero under this comparison.

4.2.3 Solve Time Analysis

Even though our previous results suggest that the Sketch-and-Solve algorithm is not accurate enough to be suitable for our application by itself, we still present its results for computational efficiency in figure 4.4 because of their relevance to the other methods. These results show that sketching can significantly outperform Householder QR in solve time across a wide range of embedding dimensions. Between $d = 1,000$ and $d \approx 20,000$ Sketch-and-Solve takes under half the time as QR and only becomes slower around $d \approx 35,000$.

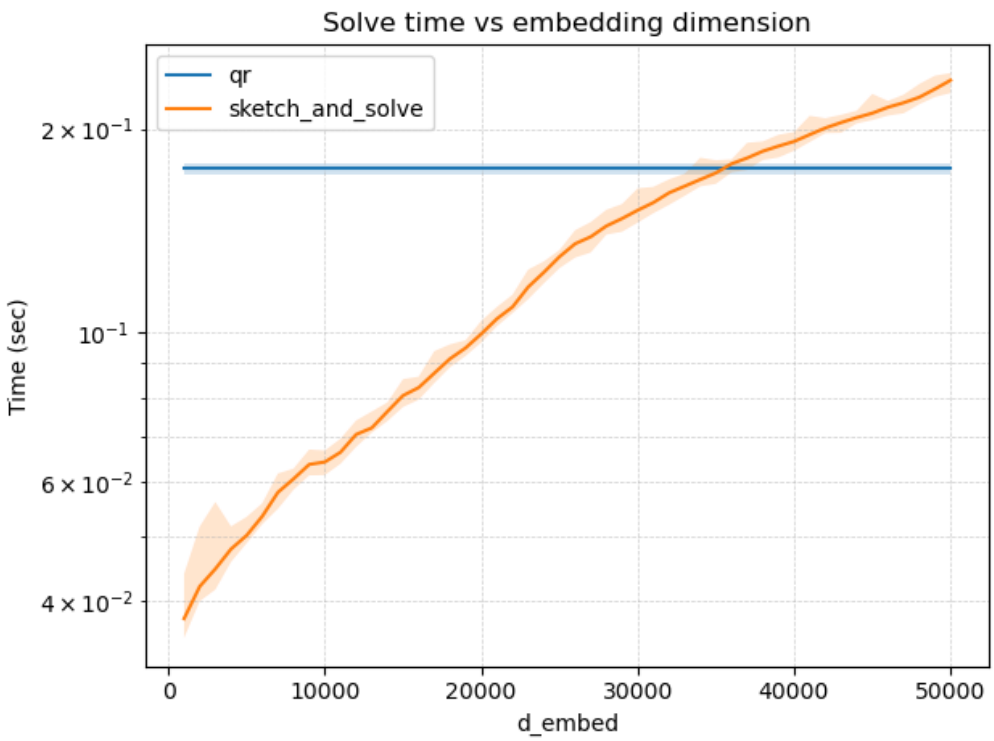

Figure 4.4: Solve time from Sketch-and-Solve least squares solutions compared to Householder QR across different sketching dimensions.

4.3 Sketch-and-Precondition & Iterative Sketching Algorithms

While the results in section 4.2 don't reflect positively on the Sketch-and-Solve algorithm as a standalone method, they show potential for algorithms that utilize sketching as an intermediate step such as the Sketch-and-Precondition and Iterative Sketching Algorithms. The theoretical guarantees we discussed in sections 3.2 and 3.3 of both of these methods are also more robust and have had more experimental success in [5, 3, 13]. For this section, we use the same test design as the previous section where $N = 50,000$ and $W = 50$ so the least squares problem has dimension $50,100 \times 100$.

4.3.1 Problem Conditioning and Embedding Distortion

While this study is conducted separately, the five least squares problems we solve are identical because the same code for creating the problems and random seed are used. This means that these studies are directly comparable. The graph corresponding to the condition number of these problems is not repeated here as it is the same as 4.1. Once again, all of these problems are very similarly conditioned with condition numbers between 1.37×10^6 and 1.44×10^6.

In addition to the conditioning of the problems being the same, the sketches created from all our sketching related methods are identical and have the same embedding distortion as discussed in section 4.1.2. We include the graph of the embedding distortion for Sketch-and-Precondition and Iterative Sketching in figure 4.5 even though both methods perfectly overlap to demonstrate this phenomenon.

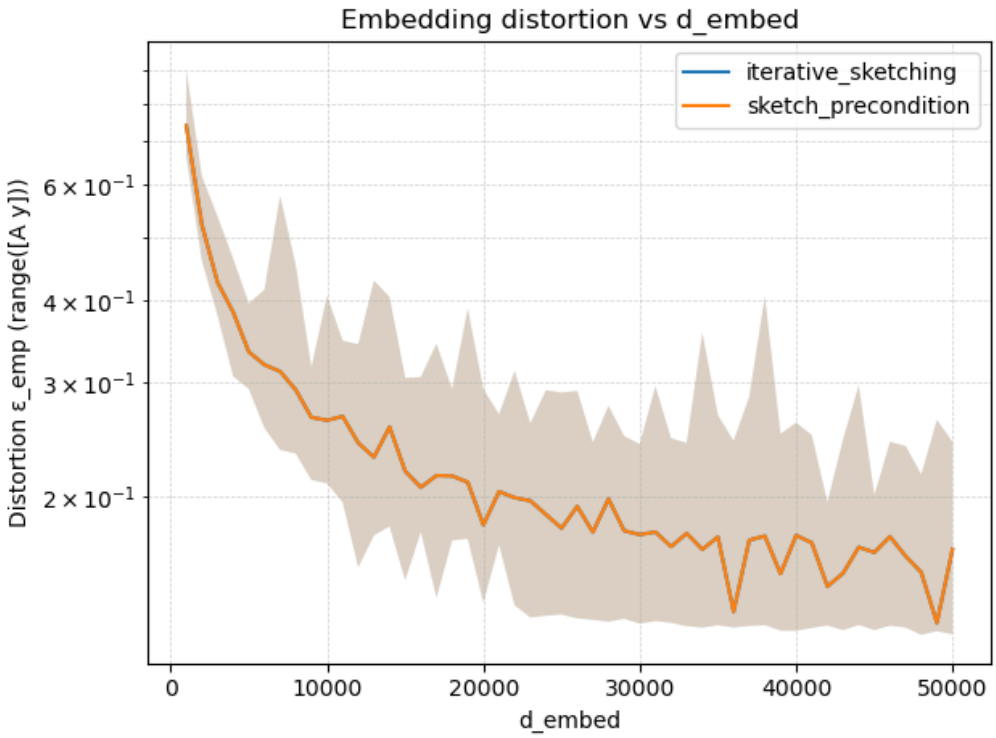

Figure 4.5: Embedding distortion from Sketch-and-Precondition and Iterative Sketching least squares solutions across different sketching dimensions.

4.3.2 Error Analysis

Now, we will discuss the accuracy of the Sketch-and-Precondition and Iterative Sketching algorithms. Once again, the error tolerance we are looking for is given by our Tikhonov regularization constant of 10^{-6}. The results for the forward, residual, and backward errors for both of these methods are plotted in figure 4.6.

The Sketch-and-Precondition algorithm exhibits all three error metrics below the 10^{-6} threshold across all embedding dimensions. This makes sense because sketching is only used here for preconditioning, the final solution is solved for exactly through the substitution of $\beta = R^{-1}z$ that was used in reverse to create the preconditioned system.

The Iterative Sketching algorithm exhibits very similar behavior to Sketch-and-Precondition in all three error metrics, however it does have some peaks just above 10^{-6} in forward error and a spike in all errors that go well above the plotted region for very small sketch dimensions. We set the maximum error value on all three of these graphs to be 10^{1}, because that is well above any reasonable accuracy threshold for our problem, but the spikes at the beginning reach well above 10^{50} in some cases for forward and residual errors. These spikes in error are very likely explained by the embedding distortion shown in figure 4.5. As discussed in section 3.3, Iterative Sketching has a stricter requirement for embedding distortion that if not met, leads to divergent error metrics.

Aside from the initial error, both of these methods converge below the threshold of 10^{-6} given by our Tikhonov regularization constant. Sketch-and-Precondition achieves a slightly lower error than Iterative Sketching, but since they are both below 10^{-6}, this does not necessarily mean any less error for our specific problem. This is similar behavior to that seen in the contexts of [5, 3, 13]. This suggests that both these methods are suitably accurate for application in the training process of rFNNs.

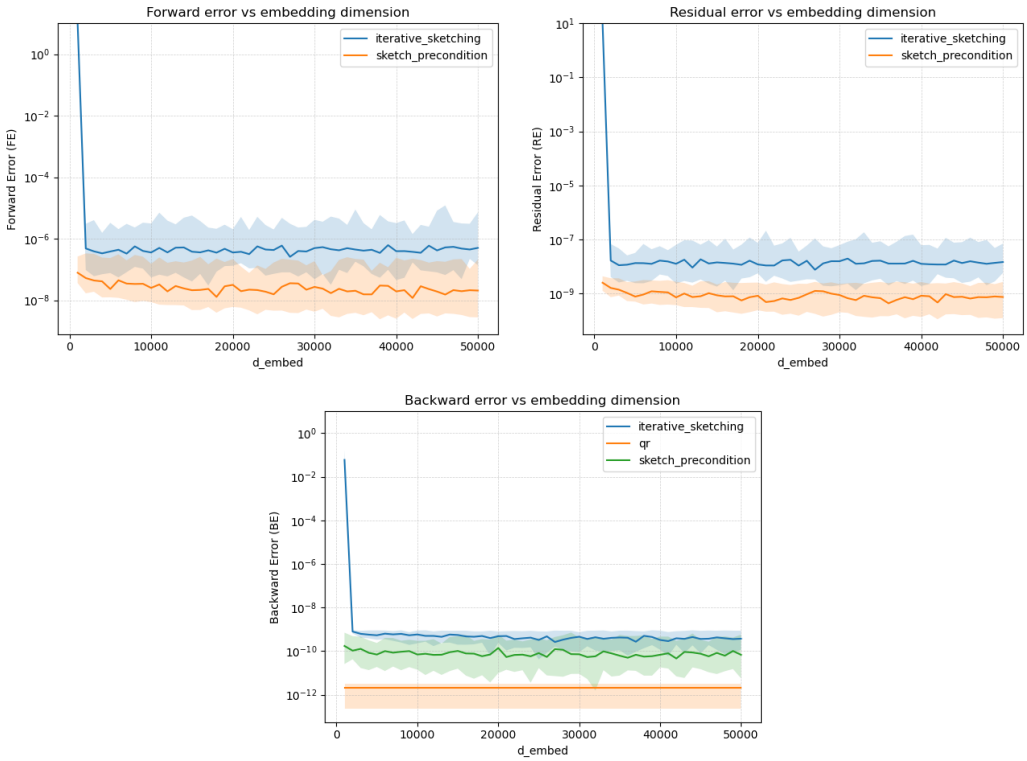

Figure 4.6: Forward, residual, and backward errors from Sketch-and-Precondition and Iterative Sketching least squares solutions compared to Householder QR across different sketching dimensions. The QR forward and residual errors aren't graphed because they are defined to be zero under this comparison.

4.3.3 Solve Time Analysis

Now that we have established accuracy for the Sketch-and-Precondition and Iterative Sketching algorithms, we will look at the computational efficiency of these algorithms compared to Householder QR. We plot the results of the solve time against embedding dimension in figure 4.7.

Here, the Sketch-and-Precondition algorithm exhibits quicker solve time than Householder QR across a wide range of embedding dimensions from the smallest we tested $d = 1,000$, all the way up to around $d \approx 30,000$. At the very low end of this range at around $d = 1,000$, we see a slight decrease in solve time with increasing embedding dimension, it is likely that this is because of the poor embedding distortion at very small embedding dimensions as seen in figure 4.5. While this had a very disruptive impact on Iterative Sketching's ability to converge accurately as discussed in section 3.3, Sketch-and-Precondition is only affected in how effective the preconditioner is. If this preconditioner isn't effective, it means that the system is more poorly conditioned so it converges slower. The slight increase in solve time with excessively low embedding dimension is exactly what we expect here because while the computational cost of the sketching scales down linearly, the embedding distortion can grow much more sharply as seen in figure 4.5.

When it comes to solve time of Iterative Sketching, it also exhibits quicker solve time than Householder QR up to around the same $d \approx 30,000$ mark. At the very low end of this range at around $d = 1,000$, the solve time well exceeds Householder QR. This matches up with the large error we see at this dimension which likely reflects the same inability of Iterative Sketching to converge without a sufficiently good subspace embedding from sketching. From a sketching dimension of about $d \approx 10,000$ all the way to the highest sketching dimension $d = 50,000$, Iterative Sketching and Sketch-and-Precondition perform nearly identically in solve time. This suggests that these solvers have very similar behavior under ideal conditions, but Iterative Sketching is much more sensitive to poor subspace embeddings.

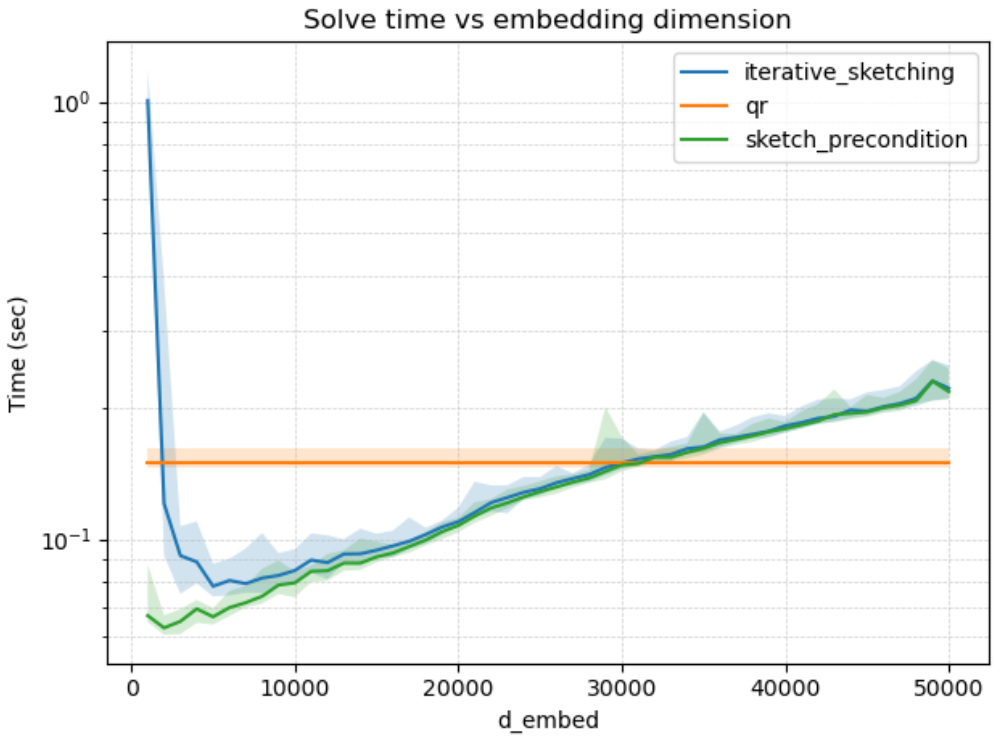

Figure 4.7: Solve time from Sketch-and-Precondition and Iterative Sketching least squares solutions compared to Householder QR across different sketching dimensions.

4.4 Scaling in N

The final study we do looks at how the Sketch-and-Precondition and Iterative Sketching algorithms scale with problem size. Previously in this section, we established suitable accuracy for both algorithms, along with promising solve times given proper sketch dimension. This section serves to study how these methods can hold up with larger problem sizes. More specifically, we sweep values of N (number of input samples) from $N = 50,000$ to $N = 10^6$ with intervals of $50,000$. These resulting least squares problems have sizes $50,100 \times 100$ up to $1,000,100 \times 100$. Since N is the main factor in the number of rows in A, this study looks at how least squares solvers scale in relation to the number of rows in the problem.

The previous section (4.3) shows that both of these methods tend to show sufficient accuracy across sketch dimensions. In terms of solve time, they tend to show better solve times with smaller sketch dimensions. That being said, both methods show some instability with very small sketch dimensions around $d = 1,000$ to $d = 2,000$. It then follows that you want to choose the smallest sketch dimension that does not negatively impact accuracy or solve time. For this study, we choose $d = 5,000$ because it is roughly the minimum for solve time for both methods and it gives a little leeway from the very small sketch dimensions that are highly unstable. When choosing a sketch dimension, it appears to be a good idea to be on the high side since the runtimes are much more sensitive to sketch dimensions being too small than too large.

4.4.1 Problem Conditioning and Embedding Distortion

For each problem size, we create five least squares problems. As problem sizes grow, it is expected that the condition numbers grow in unison. The condition numbers of the problems are plotted in figure 4.8. Similar to the previous studies, each set of problems made with the same parameters have very similar condition numbers. The smallest problems where $N = 50,000$ have very similar condition numbers around 1.4×10^6 which makes sense since these problems are made with the same parameters as those previous studies. As N grows, the condition number of the least squares problems steadily grows to around 6×10^6. These condition numbers are fairly closely spaced together which shows that the Tikhonov regularization is effectively stabilizing the least squares problems. The consistency between these least squares problems also serves as evidence they are representative of the problem we are studying and not statistical anomalies.

The embedding distortion in these tests, which is plotted in figure 4.9, remains roughly constant throughout all problem sizes which is the expected behavior because the theory discussed in section 3 says that the embedding distortion is independent of the number of rows in the least squares problem. Once again, because of the way the test is designed, the embedding distortions of sketch-and-precondition and iterative sketching are independent but identical so they perfectly

overlap in this figure. The embedding distortion also matches up with the embedding distortion seen for a sketching dimension of $d = 5,000$ in previous problem given by figure 4.2 and 4.5. While this embedding distortion seems high given the theory, especially for Iterative Sketching, our previous studies suggest that our parameter choices are close to optimal for both methods in this case.

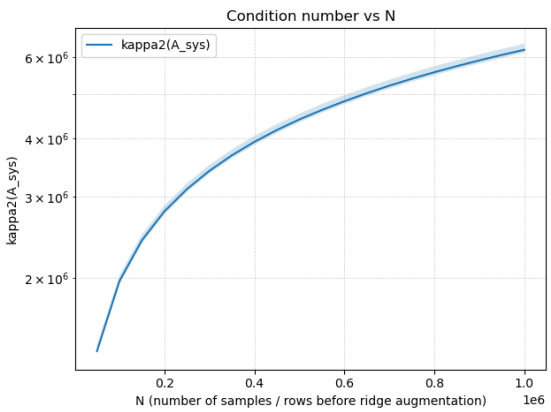

Figure 4.8: Condition number of least squares problems arising from one iteration of network training across different numbers of input samples N.

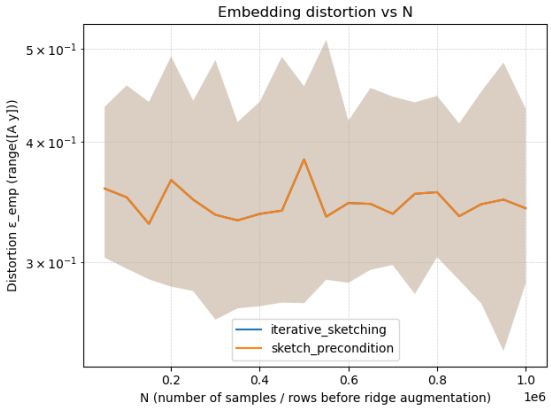

Figure 4.9: Embedding distortion from Sketch-and-Precondition and Iterative Sketching least squares solutions compared to Householder QR across different numbers of input samples, with fixed $d = 5,000$.

4.4.2 Error Analysis

In figure 4.6, we saw that both Sketch-and-Precondition and Iterative Sketching reach around the 10^{-6} error tolerance for most embedding dimensions, including $d = 5,000$ as we use here. The forward, residual, and backward errors for the scaling of these methods are plotted in figure 4.10. The behavior shown here is what we would expect, convergence of both methods to below 10^{-6}, except for some peaks of forward error for Iterative Sketching just as we saw in figure 4.6. This demonstrates that the accuracy of these methods is maintained for larger condition numbers and problem sizes.

These results also show that our embedding dimension of $d = 5,000$ is reasonable, and although the embedding distortion appears to be large, especially for Iterative Sketching, it is still effective in this context. While the forward error for Iterative Sketching peaks just above 10^{-6}, it stays close to this tolerance and this is more indicative of the stopping conditions than the ability of the method to converge. The stopping conditions of both methods can be tightened at the cost of needing more iterations, but we did not deem this necessary for these studies since the errors are largely below the tolerance of yielding additional accuracy.

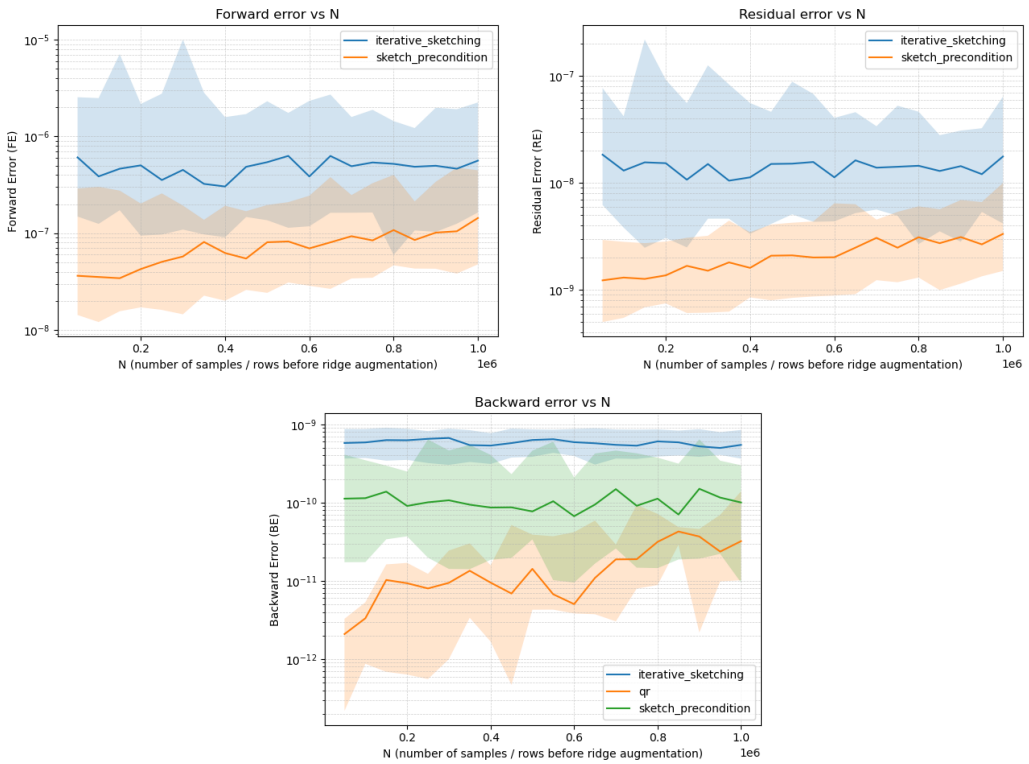

Figure 4.10: Forward, residual, and backward errors from Sketch-and-Precondition and Iterative Sketching least squares solutions compared to Householder QR across different numbers of input samples. The QR forward and residual errors aren't graphed because they are defined to be zero under this comparison.

4.4.3 Solve Time Analysis

The last result we will share is the solve time analysis of our scaling study which we plot in figure 4.11. Across all studied values of N, both Iterative Sketching and Sketch-and-Precondition have quicker solve times than Householder QR, except for one peak of Iterative Sketching that marginally exceeds QR at $N = 200,000$.

As N gets larger, the time savings of Iterative Sketching grow from taking around 70% as long as Householder QR at $N = 50,000$, to closer to half at $N = 10^6$. We also see this for Sketch-and-Precondition, but at $N = 50,000$, it already takes around half the time as Householder QR, and by $N = 10^6$, it takes around 30% as long. Both Sketch-and-Precondition and Iterative Sketching scale at a similar rate which is superior to that of Householder QR, although the runtime of Iterative Sketching here is much more variable than that of Sketch-and-Precondition.

This shows that both of these methods give real time savings over Householder QR across a variety of problems. It is also worth remembering that these tests are slightly skewed in the favor of Householder QR as discussed in section 4.1.2 because of its heavily optimized implementation. Because of the differences in implementation, the scaling properties demonstrated here are especially indicative of success for these methods. This is especially true since applications of these methods are for very large problems since that is where Householder QR becomes impractical.

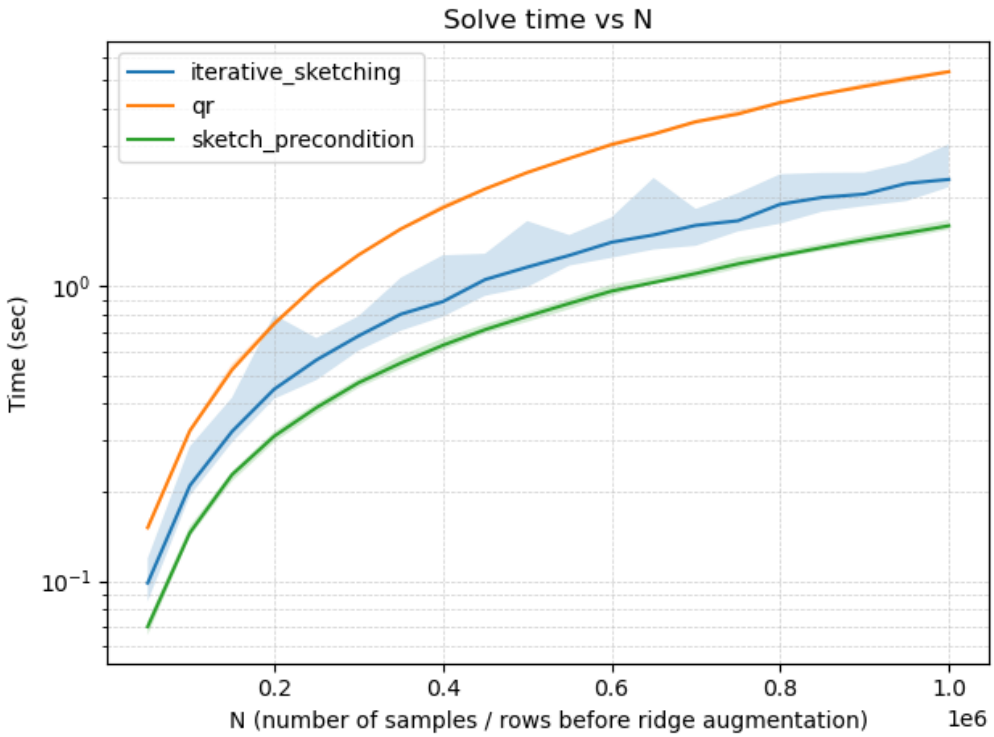

Figure 4.11: Solve time from Sketch-and-Precondition and Iterative Sketching least squares solutions compared to Householder QR across different numbers of input samples.

Chapter 5

Conclusion

This thesis studies alternative least squares solvers for use in training Random Fourier Neural Networks (rFNNs). These networks are trained using a frequency sampling algorithm that solves an overdetermined least squares problem for their corresponding magnitudes in the target function. Solving this least squares problem takes up the majority of the runtime for training rFNNs. This motivates us studying whether sketching-based solvers could be practical alternatives to standard solvers like Householder QR by reducing cost while maintaining sufficient accuracy.

To answer this question, we compared Sketch-and-Solve, Sketch-and-Precondition, and Iterative Sketching against Householder QR on least squares problems arising from one training iteration of an rFNN with one layer. The numerical results showed that Sketch-and-Solve algorithm by itself achieves significant speedups over Householder QR but is not accurate enough to be suitable for this application. In contrast, the Sketch-and-Precondition and Iterative Sketching algorithms achieved sufficient accuracy across many embedding dimensions, excluding very small ones. Both of these methods gave significant speedups over QR as well as exhibited superior scaling behavior in number of samples.

These results show that sketching is most effective here not as a final approximation, but as a tool for constructing efficient subspace embeddings that support iterative least squares methods.

For the class of rFNN least squares problems studied in this thesis, both Sketch-and-Precondition and Iterative Sketching produced suitably low error while taking less time than Householder QR. Among these two methods, Sketch-and-Precondition performed better empirically in our experiments, giving slightly faster and more consistent results in our implementation.

There are still several limitations to this work. This study only looked at least squares problems arising from a depth one rFNN which is not identical to the least squares problems from deeper networks or later training iterations. These computational results are also based on high-level Python implementations of each of the sketching methods which likely understates their performance relative to NumPy's highly optimized Householder QR solver. Future work should study least squares solver performance with scaling the width of rFNNs (number of columns in the least squares problem), deeper networks and their full training behavior, the relationship between these solvers and Tikhonov regularization, and optimization of solver implementations.

Appendix A

Alternative Target Function Graphs

Here we include the results from the tests in section 4.3 done with the target function in equation 4.1.2. These were moved to the appendix because they are trivially different from the results with the target function in equation 4.1.1 and give no further insight. The graphs of the conditioning, embedding distortion, and solve time, as well as forward, residual, and backward errors are below.

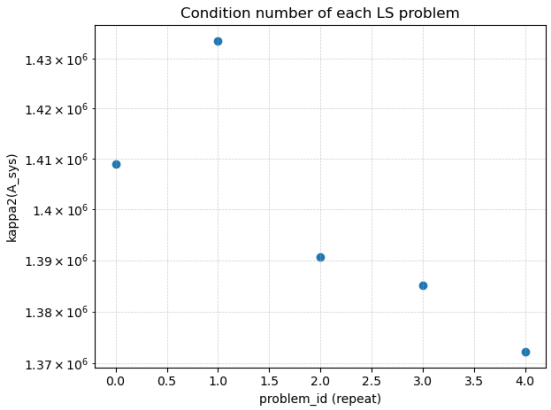

Figure A.1: Alternate target function: Condition numbers of sample least squares problems arising from one iteration of network training.

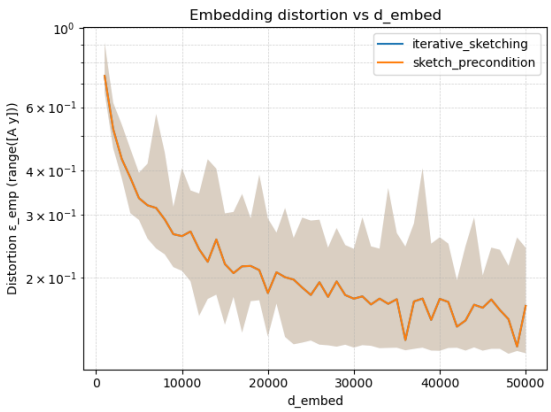

Figure A.2: Alternate target function: Embedding distortion from Sketch-and-Precondition and Iterative Sketching least squares solutions across different sketching dimensions.

Appendix A. Alternative Target Function Graphs

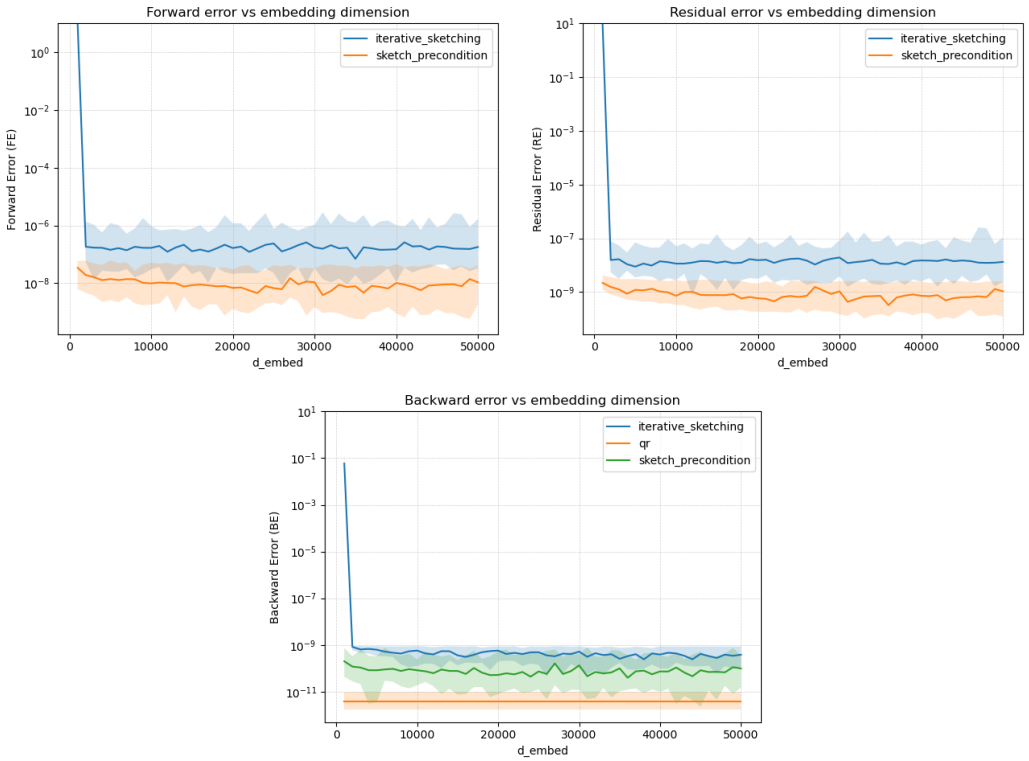

Figure A.3: Alternate target function: Forward, residual, and backward errors from Sketch-and-Precondition and Iterative Sketching least squares solutions compared to Householder QR across different sketching dimensions. The QR forward and residual errors aren't graphed because they are defined to be zero under this comparison.

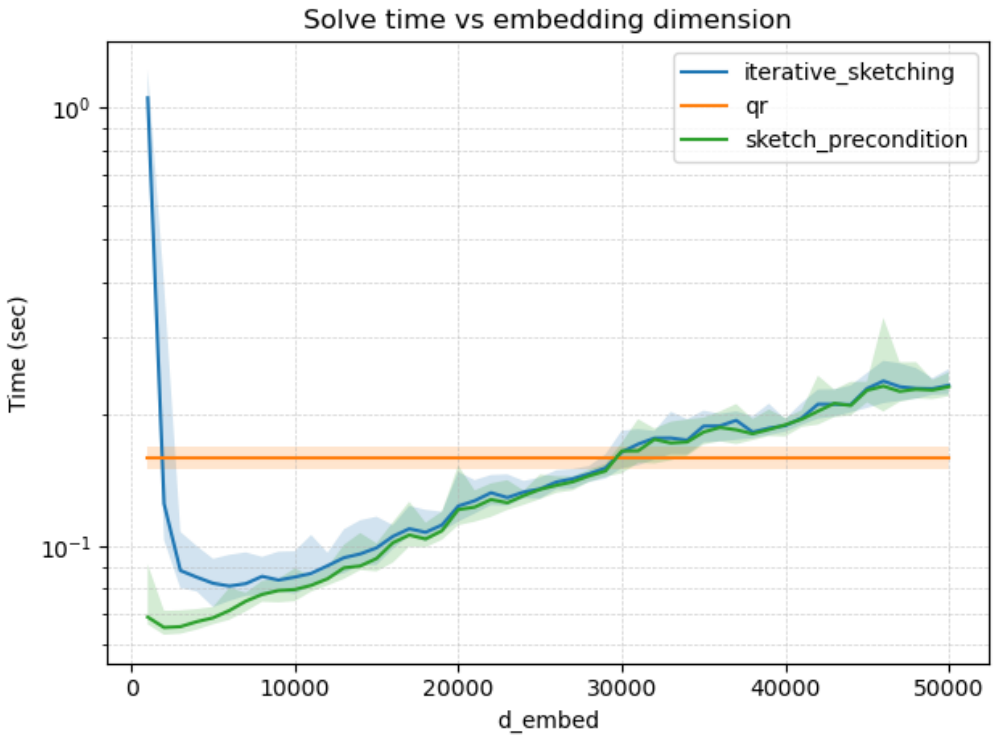

Figure A.4: Alternate target function: Solve time from Sketch-and-Precondition and Iterative Sketching least squares solutions compared to Householder QR across different sketching dimensions.

References

[1] Åke Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[2] O. Davis, G. Geraci, and M. Motamed. Deep learning without global optimization by random Fourier neural networks. *SIAM Journal on Scientific Computing*, 47(2):C265–C290, 2025.

[3] E. N. Epperly. Fast and forward stable randomized algorithms for linear least-squares problems. *SIAM Journal on Matrix Analysis and Applications*, 45:1782–1804, 2024.

[4] Loukas Grafakos. *Classical Fourier Analysis*. Springer, New York, 2 edition, 2008.

[5] M. Meier, Y. Nakatsukasa, A. Townsend, and M. Webb. Are sketch-and-precondition least squares solvers numerically stable? *SIAM Journal on Matrix Analysis and Applications*, 45:905–929, 2024.

[6] J. Nelson and H. L. Nguyen. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *Proceedings of the 2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 117–126, 2013.

[7] Christopher C. Paige and Michael A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1):43–71, 1982.

[8] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, volume 20, 2007.

[9] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*. Springer, New York, 2 edition, 2004.

[10] SciPy Community. scipy.sparse.linalg.lsqr — SciPy v1.17.0 Manual. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.lsqr.html`, 2025. Accessed 2026-04-04.

[11] SciPy Developers. SciPy Sparse Matrix Documentation. `https://docs.scipy.org/doc/scipy/reference/sparse.html`. Accessed: 2026-04-28.

[12] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 2022.

[13] D. P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1–2):1–157, 2014.