Neumann Networks for Linear Inverse Problems in Imaging

Davis Gilton, Greg Ongie, Rebecca Willett*

June 5, 2019

Abstract

Many challenging image processing tasks can be described by an ill-posed linear inverse problem: deblurring, deconvolution, inpainting, compressed sensing, and superresolution all lie in this framework. Traditional inverse problem solvers minimize a cost function consisting of a data-fit term, which measures how well an image matches the observations, and a regularizer, which reflects prior knowledge and promotes images with desirable properties like smoothness. Recent advances in machine learning and image processing have illustrated that it is often possible to *learn* a regularizer from training data that can outperform more traditional regularizers. We present an end-to-end, data-driven method of solving inverse problems inspired by the Neumann series, which we call a Neumann network. Rather than unroll an iterative optimization algorithm, we truncate a Neumann series which directly solves the linear inverse problem with a data-driven nonlinear regularizer. The Neumann network architecture outperforms traditional inverse problem solution methods, model-free deep learning approaches, and state-of-the-art unrolled iterative methods on standard datasets. Finally, when the images belong to a union of subspaces and under appropriate assumptions on the forward model, we prove there exists a Neumann network configuration that well-approximates the optimal oracle estimator for the inverse problem and demonstrate empirically that the trained Neumann network has the form predicted by theory.

1 Learning to Regularize

In this paper we consider solving linear inverse problems in imaging in which a p-pixel image, $\boldsymbol{\beta}^\star \in \mathbb{R}^p$ (in vectorized form), is observed via m noisy linear projections as $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^\star + \boldsymbol{\epsilon}$, where $\boldsymbol{y}, \boldsymbol{\epsilon} \in \mathbb{R}^m$ and $\boldsymbol{X} \in \mathbb{R}^{m\times p}$. This general model is used throughout computational imaging, from basic image restoration tasks like deblurring, super-resolution, and image inpainting [1], to a wide variety of tomographic imaging applications, including common types of magnetic resonance imaging [2], X-ray computed tomography [3], radar imaging [4], among others [5]. The task of estimating

*D. Gilton is with the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI, 53706 USA (e-mail: gilton@wisc.edu). G. Ongie is with the Department of Statistics, University of Chicago, Chicago, IL, 60637 USA (e-mail: gongie@uchicago.edu). R. Willett is with the Department of Statistics and Computer Science, University of Chicago, Chicago, IL, 60637 USA (e-mail: willett@uchicago.edu).

$\boldsymbol{\beta}^\star$ from $\boldsymbol{y}$ is often referred to as *image reconstruction*. Classical image reconstruction methods assume some prior knowledge about $\boldsymbol{\beta}^\star$ such as smoothness [6], sparsity in some dictionary or basis [7–9], or other geometric properties [10–13], and attempt to estimate a $\widehat{\boldsymbol{\beta}}$ that is both a good fit to the observation $\boldsymbol{y}$ and that also conforms to this prior knowledge. In general, a *regularization function* $r(\boldsymbol{\beta})$ measures the lack of conformity of $\boldsymbol{\beta}$ to this prior knowledge and $\widehat{\boldsymbol{\beta}}$ is selected so that $r(\widehat{\boldsymbol{\beta}})$ is as small as possible while still providing a good fit to the data.

However, recent work in computer vision using deep neural networks has leveraged large collections of "training" images to yield unprecedented image recognition performance [14–16], and an emerging body of research is exploring whether this training data can also be used to improve the quality of image reconstruction. In other words, *can training data be used to learn how to regularize inverse problems?* As we detail below, existing methods include using training images to learn a low-dimensional image manifold and constraining $\widehat{\boldsymbol{\beta}}$ to lie on this manifold [17] or learning a denoising autoencoder that can be treated as a regularization step (*i.e.,* proximal operator) within an iterative reconstruction scheme [18].

In this paper, we propose a novel neural network architecture based on the Neumann series expansion [19, 20] that we call a *Neumann network*, describe several of its key theoretical properties, and empirically illustrate its superior performance on a variety of reconstruction tasks. In particular,

- Neumann networks, which directly incorporate the forward operator $\boldsymbol{X}$ into the network architecture, can have dramatically lower sample complexity than *model-agnostic* networks that attempt to learn the entire image space. As a result, they are much more amenable to applications such as medical imaging or scientific domains where datasets may be smaller.

- Neumann networks naturally yield a block-wise structure with *skip connections* [14] emanating from each block. These skip connections appear to yield a smoother optimization landscape that is easier to train than related network architectures.

- When the images of interest lie on a union of subspaces, and when the trainable nonlinear components of the network have sufficient expressiveness/capacity, there exists a Neumann network estimator that approximates the optimal oracle estimator arbitrarily well. Furthermore, after training the Neumann network on simulated data drawn from a union of subspaces, we show the learned nonlinear components in the trained Neumann network have the form predicted by theory.

- A simple preconditioning step combined with the Neumann network further improves empirical performance.

- The empirical performance of the Neumann network on superresolution, deblurring, compressed sensing, and inpainting problems exceeds that of competing methods.

2 Previous Work

There are several general categories of methods used to learn to solve inverse problems, which are reviewed below. Throughout, we assume we have training samples of the form $(\boldsymbol{\beta}_i, \boldsymbol{y}_i)$ for $i = 1, \ldots, N$, where $\boldsymbol{y}_i = \boldsymbol{X}\boldsymbol{\beta}_i + \boldsymbol{\epsilon}_i$, $\boldsymbol{X} \in \mathbb{R}^{m \times p}$ is known and the noise $\boldsymbol{\epsilon}_i$ is treated as unknown.

2.1 Agnostic

An agnostic learner uses the training data to learn a mapping from $\boldsymbol{y}$ to $\boldsymbol{\beta}$ without any knowledge of $\boldsymbol{X}$ at any point in the training or testing process [21]. The general principle is that, given enough training data, we should be able to learn everything we need to know about $\boldsymbol{X}$ to successfully estimate $\boldsymbol{\beta}$. Empirically, the success of this approach appears to be highly dependent on the forward operator $\boldsymbol{X}$. This straightforward approach has been demonstrated on superresolution [22, 23], blind deconvolution [21], and motion deblurring [24], among others. In general, this approach requires large quantities of training data because it is required to not only learn the geometry of the image space containing the $\boldsymbol{\beta}$'s, but also aspects of $\boldsymbol{X}$. A particularly successful approach to solving inverse problems with neural networks has been through residual learning [14].

2.2 Decoupled

A decoupled approach operates in two stages. In the first stage, a collection of training images $\boldsymbol{\beta}_i$ is used to learn a representation of the image space of interest. In the second stage, this learned representation is incorporated into a mapping from $(\boldsymbol{y}, \boldsymbol{X})$ to $\widehat{\boldsymbol{\beta}}$. That is, the learning takes place in a manner that is decoupled from the inverse problem at hand. Here we present two examples of this.

First, we might learn a generative model G for $\boldsymbol{\beta}$'s that takes as input a low-dimensional vector $\boldsymbol{z} \in \mathbb{R}^d$ for $d < p$ and outputs $\boldsymbol{\beta} = G(\boldsymbol{z})$. The basic idea is that the images of interest lie on a low-dimensional submanifold that can be indexed by $\boldsymbol{z}$. Given the learned G, we can compute $\widehat{\boldsymbol{\beta}}$ from $\boldsymbol{y}$ via

$$\widehat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}=G(\boldsymbol{z}),\boldsymbol{z}\in\mathbb{R}^d}{\arg\min} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2. \tag{1}$$

This approach was described in [17] with compelling empirical performance for compressed sensing.

Alternatively, we might learn a denoising autoencoder that could be used as a proximal operator in an iterative reconstruction method. Specifically, imagine we had a fixed regularizer $r(\cdot)$ and want to set

$$\widehat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\arg\min} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + r(\boldsymbol{\beta}). \tag{2}$$

A proximal gradient algorithm [25, 26] starts with an initial estimate $\boldsymbol{\beta}^{(0)}$ and step size $\eta > 0$ and then iterates between computing a gradient descent step that pushes the current estimate $\boldsymbol{\beta}^{(k)}$ to be a better fit to the data, followed by a *proximal operator* that finds an estimate in the proximity of the resulting iterate that is well-regularized (as measured by $r(\cdot)$). This second step is often thought of as a denoising step. One approach to learning to solve inverse problems is to implicitly learn $r(\cdot)$ by explicitly learning a proximal operator in the form of a denoising autoencoder [18, 27, 28].

The key feature in both of these approaches is that all training takes place independently of $\boldsymbol{X}$ – *i.e.,* we either learn a generative model or a proximal operator using the training data, neither of which require knowledge of $\boldsymbol{X}$. The advantage of this approach is that once training has taken place, the learned generative model or proximal operator can be used for *any* linear inverse

problem, so we do not need to re-train a system for each new inverse problem. In other words, the learning is *decoupled* from solving the inverse problem.

However, the flexibility of the decoupled approach comes with a high price in terms of sample complexity. To see why, note that learning a generative model or a denoising autoencoder fundamentally amounts to estimating a probability distribution and its support over the space of images; let us denote this distribution as $\phi(\boldsymbol{\beta})$. Thoroughly understanding the space of images of interest is important if our learned regularizer is to be used for linear inverse problems of which we are unaware during training.

On the other hand, when we know $\boldsymbol{X}$ at training time, then we only need to learn the *conditional* distribution $\phi(\boldsymbol{\beta}|\boldsymbol{X}\boldsymbol{\beta})$ or $\phi(\boldsymbol{\beta}|\boldsymbol{y})$ [29].

For example, imagine an image inpainting scenario in which we only observe a subset of pixels in the image $\boldsymbol{\beta}$. Rather than learn the distribution over the space of all possible images, we only need to learn the distribution over the space of missing pixels conditioned on the observed pixels, $\phi(\boldsymbol{\beta}|\boldsymbol{X}\boldsymbol{\beta})$. Of course, $\phi(\boldsymbol{\beta}|\boldsymbol{X}\boldsymbol{\beta})$ can be calculated from $\phi(\boldsymbol{\beta})$ and $\boldsymbol{X}$ using Bayes' law, but the latter distribution may lie in a much lower-dimensional space, making it easier to learn with limited data.

It is well-known that the accuracy of any estimate of $\phi(\boldsymbol{\beta})$ has a minimax rate that scales as $\mathcal{O}(N^{-\frac{\alpha}{2\alpha+p}})$ with N the number of training samples, p the dimension of $\phi(\boldsymbol{\beta})$, and α a smoothness term [30–32]. This scaling is quite restrictive, but if the conditional density only depends on a subset of size p' of the original p coordinates, the rate for estimating the conditional density function is $\mathcal{O}(N^{-\frac{\alpha}{2\alpha+p'}})$ [29].

The key point is that decoupled approaches (implicitly) require learning the full density $\phi(\boldsymbol{\beta})$, whereas a method that incorporates $\boldsymbol{X}$ into the learning process has the potential to simply learn the conditional density $\phi(\boldsymbol{\beta}|\boldsymbol{X}\boldsymbol{\beta})$, which often can be performed accurately with relatively little training data. This observation is supported by our experimental results, which illustrate that decoupled approaches generally require far more training samples than methods that incorporate knowledge of $\boldsymbol{X}$.

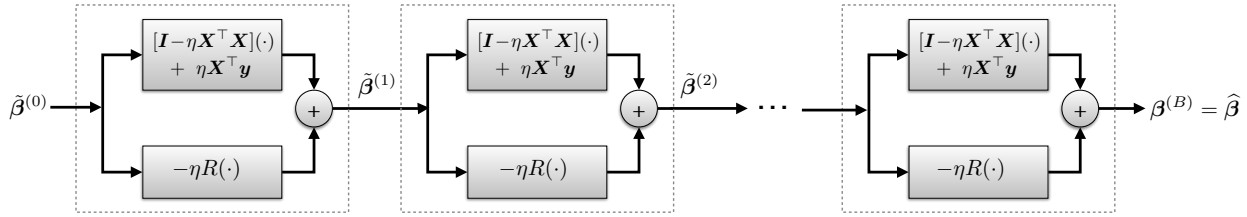


Figure 1: Unrolled gradient descent network. The result of B iterations of gradient descent with a fixed step size η and regularizer with gradient R, as in (3) is equivalent to the output of the above network, with each block corresponding to a single iteration. The network maps a linear function of the measurements, $\boldsymbol{\beta}^{(0)} = \eta\boldsymbol{X}^\top\boldsymbol{y}$, to a reconstruction $\widehat{\boldsymbol{\beta}}$ by successive application of an operator of the form $[\boldsymbol{I} - \eta\boldsymbol{X}^\top\boldsymbol{X}](\cdot) - \eta R(\cdot)$ and addition of $\eta\boldsymbol{X}^\top\boldsymbol{y}$. Here R is a trained neural network, and the scale parameter η is also trained.

2.3 Unrolled Optimization

Another approach treats a learned component of a network as the gradient of a prior over the data or a proximal operator for a regularizer [33]. Suppose the desired optimal point $\boldsymbol{\beta}^\star$ satisfies the

optimality condition in (2) Now we assume $r(\cdot)$ is differentiable and let $R(\boldsymbol{\beta}) := \nabla r(\boldsymbol{\beta})$ denote the gradient of the regularizer. Then solving (2) can be accomplished using iterative optimization; for instance, gradient descent would result in the iterates

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \eta \left[\boldsymbol{X}^\top (\boldsymbol{X}\boldsymbol{\beta}^{(k)} - \boldsymbol{y}) + R(\boldsymbol{\beta}^{(k)}) \right] \tag{3}$$

for a step size $\eta > 0$. Imagine computing these iterates for a fixed number of iterations, which we will denote B (for Blocks, as will become clear shortly).

"Unrolling" an optimization method refers to taking an iterative optimization method and, instead of iterating until convergence, thinking of a series of B iterates as a single operation to be applied to an input. This idea as applied to gradient descent is represented pictorially in Figure 1. We can now represent the gradient of the regularizer, $R(\cdot)$, with a trainable neural network. In contrast to the decoupled approach described above, unrolled optimization methods learn the regularizer (or its gradient) in the context of the forward model $\boldsymbol{X}$ and training observations $\boldsymbol{y}_i$ by minimizing the disparity between the true $\boldsymbol{\beta}_i$ and $\widehat{\boldsymbol{\beta}}(\boldsymbol{y}_i)$, the output of the full network (see Figure 1). This end-to-end training sidesteps the sample complexity challenges described in Section 2.2.

The unrolling approach can be applied to a variety of optimization algorithms beyond gradient descent. The earliest proposed unrolled inverse problem solver was [34], in which the authors proposed unrolling the Iterative Shrinkage and Thresholding Algorithm (ISTA) [35] and the coordinate descent algorithm; further refinements of this approach were proposed in [36, 37]. More recent work has illustrated the efficacy of unrolled optimization as applied to (proximal) gradient descent [33, 38, 39], alternating directions method of multipliers [40], primal-dual methods [41], half-quadratic splitting [42, 43], block coordinate descent [44–46], alternating minimization [47], iterative reweighted least squares [48, 49], and approximate message passing [50]. In proximal gradient settings, the learned neural network is interpreted as a learned proximal operator, whereas in the gradient descent network, the learned neural network is interpreted as the gradient of the regularizer at the input. In other words, for different unrolled optimization methods the learned neural network can play different roles.

While for practical reasons the number of blocks B must be kept small in end-to-end training, empirically this does not appear to be an obstacle to good performance. For example, [34] notes that end-to-end training reduces the iterations of the ISTA algorithm required to achieve a fixed error rate by a factor of 20, and [38] achieve promising performance with $B = 8$ proximal gradient descent iterations. Another strategy to enable deeper unrollings is to perform block-by-block training as in [45]; however, this approach is unsuitable when the neural network weights are shared between blocks, which is the case in our setting. It is possible to relax the shared-weights assumption, but [47] has illustrated that in the low-sample setting, different learned weights in each block can be suboptimal.

3 Neumann Networks

Below, we adopt the following strategy. First, we consider the setting in which the gradient of the regularizer is a linear operator and derive a simple Neumann series approximation to an optimal solution of (2). We then consider the overall *Neumann network* formed if R is represented by a (potentially nonlinear) neural network. In this section, we treat a nonlinear network operation as a

heuristic that we justify theoretically in Section 4. This section also describes a simple preconditioning step that can improve the accuracy of our approach and an explicit comparison between the proposed Neumann network and the unrolled gradient descent network described in Section 2.3.

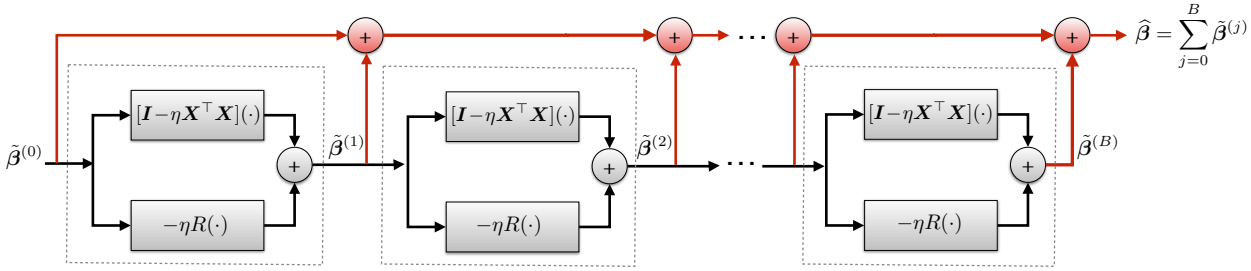


Figure 2: Proposed Neumann network architecture. Inspired by the Neumann series expansion for computing the inverse of an operator, a Neumann network maps a linear function of the measurements, $\widetilde{\boldsymbol{\beta}}^{(0)} = \eta\boldsymbol{X}^\top\boldsymbol{y}$ to a reconstruction $\widehat{\boldsymbol{\beta}}$ by successive application of an operator the form $[\boldsymbol{I} - \eta\boldsymbol{X}^\top\boldsymbol{X}](\cdot) - \eta R(\cdot)$ while summing the intermediate outputs of each block. Here R is a trained neural network, and the scale parameter η is also trained. Unlike other networks based on unrolling of iterative optimization algorithms, the series structure of Neumann networks lead naturally to skip connections [14] (highlighted in red) that route the output of each dashed block to directly to the output layer.

3.1 Proposed Network Architecture

Our proposed network architecture is motivated by the regularized least squares optimization problem (2) in the special case where the regularizer r is quadratic. In particular, assume $r(\boldsymbol{\beta}) = \frac{1}{2}\boldsymbol{\beta}^\top\boldsymbol{R}\boldsymbol{\beta}$ so that $\nabla r(\boldsymbol{\beta}) = \boldsymbol{R}\boldsymbol{\beta}$ for some matrix $\boldsymbol{R} \in \mathbb{R}^{p\times p}$. Then a necessary condition for $\boldsymbol{\beta}^\star$ to be a minimizer of (2) in this case is

$$(\boldsymbol{X}^\top\boldsymbol{X} + \boldsymbol{R})\boldsymbol{\beta}^\star = \boldsymbol{X}^\top\boldsymbol{y}. \tag{4}$$

Assuming the matrix on the left-hand side is invertible, the solution is given by

$$\boldsymbol{\beta}^\star = (\boldsymbol{X}^\top\boldsymbol{X} + \boldsymbol{R})^{-1}\boldsymbol{X}^\top\boldsymbol{y}. \tag{5}$$

In order to approximate the matrix inverse in (5) we consider a Neumann series expansion of linear operators [19, 20], which we now recall. Let $\boldsymbol{A}$ be any $p \times p$ matrix and let $\boldsymbol{I}$ denote the $p \times p$ identity matrix. If the *Neumann series* $\sum_{k=0}^{\infty}\boldsymbol{A}^k$ converges then $\boldsymbol{I} - \boldsymbol{A}$ is invertible and we have

$$(\boldsymbol{I} - \boldsymbol{A})^{-1} = \sum_{k=0}^{\infty}\boldsymbol{A}^k = \boldsymbol{I} + \boldsymbol{A} + \boldsymbol{A}^2 + \boldsymbol{A}^3\cdots \tag{6}$$

In particular, a sufficient condition for the convergence of the Neumann series is $\|\boldsymbol{A}\| < 1$ where $\|\cdot\|$ is the operator norm. We will make use of an alternative form of the same identity:

$$\boldsymbol{B}^{-1} = \eta\sum_{k=0}^{\infty}(\boldsymbol{I} - \eta\boldsymbol{B})^k, \tag{7}$$

which is obtained through a change of variables.

Applying the Neumann series expansion (7) to the matrix inverse appearing in (5), we have[1]

$$\boldsymbol{\beta}^\star = \sum_{j=0}^{\infty} (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta \boldsymbol{R})^j (\eta \boldsymbol{X}^\top \boldsymbol{y}). \tag{8}$$

Truncating the series in (8) to $B+1$ terms, and replacing multiplication by the matrix $\boldsymbol{R}$ with a general mapping $R : \mathbb{R}^p \to \mathbb{R}^p$, motivates an estimator $\widehat{\boldsymbol{\beta}}$ of the form

$$\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) := \sum_{j=0}^{B} ([\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X}](\cdot) - \eta R(\cdot))^j (\eta \boldsymbol{X}^\top \boldsymbol{y}). \tag{9}$$

We turn (9) into a trainable estimator by letting $R = R_{\boldsymbol{\theta}}$ be a trainable mapping depending on a vector of parameters $\boldsymbol{\theta} \in \mathbb{R}^q$ to be learned from training data. Specifically, in this work we assume $R_{\boldsymbol{\theta}}$ is a neural network, where $\boldsymbol{\theta}$ is a vectorized set of weights and biases that define the network. We also treat the step-size choice η as a trainable parameter. The class of estimators $\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \widehat{\boldsymbol{\beta}}(\boldsymbol{y}; \boldsymbol{\theta}, \eta)$ specified (9) with trainable network $R = R_{\boldsymbol{\theta}}$ we call *Neumann networks*.

Observe that Neumann networks are motivated by an application of the Neumann series identity in the case where the gradient of the regularizer is a linear operator (or, equivalently, the regularizer is quadratic). However, for a general regularizer r such that $R = \nabla r$ is nonlinear, the Neumann network estimator $\widehat{\boldsymbol{\beta}}(\boldsymbol{y})$ in (9) may not be a good solution to the optimization problem (2). This is because the Neumann series identity (7) only holds for linear operators. Despite this fact, we show in the next section that a Neumann network estimator is still mathematically justified under certain model assumptions on the data distribution for which the ideal R is *piecewise linear*. For now, we simply treat the Neumann network estimator as a heuristic motivated by case where R is linear.

To see how (9) can be formulated as a network, observe that the terms in (9) have the following recursive form: let the input to the network be $\tilde{\boldsymbol{\beta}}^{(0)} := \eta \boldsymbol{X}^\top \boldsymbol{y}$ and define

$$\tilde{\boldsymbol{\beta}}^{(j)} := (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X}) \tilde{\boldsymbol{\beta}}^{(j-1)} - \eta R(\tilde{\boldsymbol{\beta}}^{(j-1)}) \tag{10}$$

for all $j = 1, ..., B$. Then we have $\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \sum_{j=0}^{B} \tilde{\boldsymbol{\beta}}^{(j)}$.

Figure 2 shows a block diagram for implementing the Neumann network using the recursion (10). Each block with a dashed boundary in Figure 2 represents an application of the operator $[\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X}](\cdot) - \eta R(\cdot)$. Due to its underlying series structure, the Neumann network has several *skip connections* (highlighted in red) that route the output of each dashed block (*i.e.,* the $\tilde{\boldsymbol{\beta}}^{(j)}$'s) to the output layer, similar to those found in residual networks [14] and related architectures [15]. These skip connections are a distinguishing feature of Neumann networks compared to networks derived from unrolled optimization approaches, such as unrolled gradient descent (see Figure 1). We hypothesize these additional skip connections result in a smoother optimization landscape relative to other unrolling approaches, which allows for easier training via stochastic gradient descent. See Section 5.7 for empirical evidence and more discussion on this point.

[1]The series in (7) is guaranteed to converge if $\|\boldsymbol{I} - \eta \boldsymbol{B}\| < 1$. Hence, the expansion in (8) is valid provided $\|\boldsymbol{I} - \eta(\boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{R})\| < 1$, which holds if and only if $\boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{R}$ is positive definite and $\eta < \|\boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{R}\|^{-1}$.

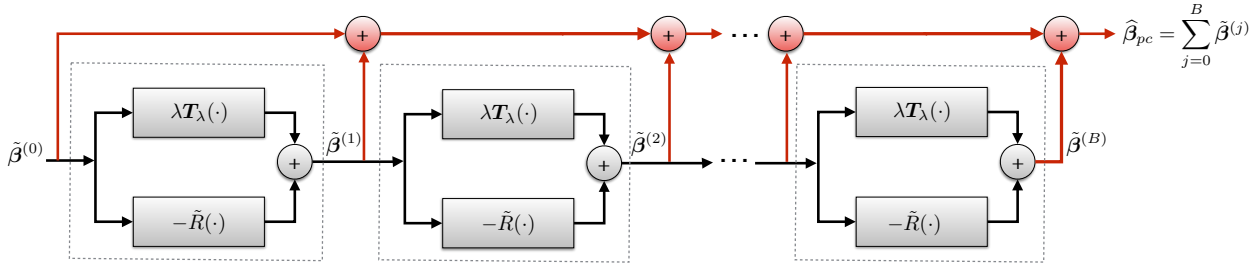


Figure 3: Proposed preconditioned Neumann network architecture. The network has the same basic architecture as a Neumann network, but uses a different linear component given by $\boldsymbol{T}_\lambda = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1}$ where $\lambda > 0$ and a different initialization $\tilde{\boldsymbol{\beta}}^{(0)} = \boldsymbol{T}_\lambda \boldsymbol{X}^\top \boldsymbol{y}$. When the matrix inverse in $\boldsymbol{T}_\lambda$ is computationally prohibitive to apply, we replace all instances of $\boldsymbol{T}_\lambda$ with an unrolling of a fixed number of iterations of the conjugate gradient algorithm, similar to [47]. Here $\tilde{R}$ is a trained neural network, and the scale parameter λ is also trained when feasible.

3.2 Preconditioning

Efficiently finding a solution to the linear system (4) using an iterative method is challenging when the matrix $\boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{R}$ is ill-conditioned. This suggests that our Neumann network approach, which is derived from a Neumann series expansion of the system in (4), may benefit from preconditioning. Here we derive a variant of Neumann networks inspired by a preconditioning of (4).

Starting from (4), for any $\lambda > 0$ we have

$$(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})\boldsymbol{\beta}^\star + (\boldsymbol{R} - \lambda \boldsymbol{I})\boldsymbol{\beta}^\star = \boldsymbol{X}^\top \boldsymbol{y}. \tag{11}$$

Applying $\boldsymbol{T}_\lambda := (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1}$ to both sides and rearranging terms gives

$$(\boldsymbol{I} - \lambda \boldsymbol{T}_\lambda + \tilde{\boldsymbol{R}})\boldsymbol{\beta}^\star = \boldsymbol{T}_\lambda \boldsymbol{X}^\top \boldsymbol{y}. \tag{12}$$

where we have set $\tilde{\boldsymbol{R}} = \boldsymbol{T}_\lambda \boldsymbol{R}$. Following the same steps used to derive the Neumann network, we arrive at the modified estimator

$$\widehat{\boldsymbol{\beta}}_{pc}(\boldsymbol{y}) = \sum_{j=0}^{B} (\lambda \boldsymbol{T}_\lambda(\cdot) - \tilde{R}(\cdot))^j \boldsymbol{T}_\lambda \boldsymbol{X}^\top \boldsymbol{y} \tag{13}$$

which we call a *preconditioned Neumann network*. Here $\tilde{R} = \tilde{R}_{\boldsymbol{\theta}}$ is a trainable mapping depending on parameters $\boldsymbol{\theta}$. We also treat $\lambda > 0$ as a trainable parameter when gradients with respect to λ are easily calculated (more on this below).

As shown in Figure 3, a preconditioned Neumann network has the same basic network architecture as the standard Neumann network, except the linear component $[\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X}](\cdot)$ is replaced with $[\lambda \boldsymbol{T}_\lambda](\cdot)$ and the learned component $[-\eta R](\cdot)$ is replaced with $[-\tilde{R}](\cdot)$. The preconditioned Neumann network also has a different initialization, $\tilde{\boldsymbol{\beta}}^{(0)} = \boldsymbol{T}_\lambda \boldsymbol{X}^\top \boldsymbol{y}$, which is the solution to the Tikhonov regularized least squares problem $\min_{\boldsymbol{\beta}} \|\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \lambda \|\boldsymbol{\beta}\|^2$. For many inverse problems in imaging, such as deblurring, this is much more accurate approximation to the ideal solution than the matrix transpose initialization $\tilde{\boldsymbol{\beta}}^{(0)} = \eta \boldsymbol{X}^\top \boldsymbol{y}$ of the standard Neumann network. Hence, we might expect that a preconditioned Neumann network could achieve higher quality solutions

with fewer blocks B. Our experiments on deblurring of natural images (see Figure 9b) support this observation.

Applying $\boldsymbol{T}_\lambda(\cdot)$ may be computationally prohibitive for certain large-scale inverse problems in imaging, such as those arising in CT and MRI reconstruction. To address this issue, we adapt the approach of [47] and replace all instances of $\boldsymbol{T}_\lambda(\cdot)$ in the preconditioned Neumann network by an unrolling of a fixed number of iterations of the conjugate gradient (CG) algorithm [51], which approximates the application of $\boldsymbol{T}_\lambda(\cdot)$. Unrolling CG does not require any additional trainable parameters, and backpropagation through the CG layers can be performed via automatic differentiation. This strategy has been shown to be effective for various large-scale MRI reconstruction problems [48, 49]. Incorporating a trainable λ parameter into this approach is simple since the derivatives of the end-to-end network $\widehat{\boldsymbol{\beta}}_{pc}$ with respect to λ are also easily computed by automatic differentiation. In particular, we do not need $\boldsymbol{T}_\lambda$ to have an analytic expression in terms of λ in order to compute derivatives.

Finally, we note other preconditioned Neumann networks could be derived by replacing $\boldsymbol{I}$ with a general matrix $\boldsymbol{S}$ such that $\boldsymbol{X}^\top\boldsymbol{X} + \lambda\boldsymbol{S}$ is positive definite, e.g., $\boldsymbol{S} = \boldsymbol{D}^\top\boldsymbol{D}$ where $\boldsymbol{D}$ is a discrete approximation of the image gradient. For simplicity, we restrict ourselves to the choice $\boldsymbol{S} = \boldsymbol{I}$ in this work.

3.3 Equivalence of Unrolled Gradient Descent and Neumann Network for a Linear Learned Component

Suppose the learned component R is linear, *i.e.,* $R(\boldsymbol{\beta}) = \boldsymbol{R}\boldsymbol{\beta}$ for some matrix $\boldsymbol{R} \in \mathbb{R}^{p\times p}$. The Bth iteration $\boldsymbol{\beta}^{(B)}$ of unrolled gradient descent (3) with step size $\eta > 0$ and initialization $\boldsymbol{\beta}^{(0)} = \eta\boldsymbol{X}^\top\boldsymbol{y}$ can be expanded to obtain

$$\boldsymbol{\beta}^{(B)} = \eta\sum_{j=0}^{B}(\boldsymbol{I} - \eta\boldsymbol{X}^\top\boldsymbol{X} - \eta\boldsymbol{R})^j\boldsymbol{X}^\top\boldsymbol{y},$$

which is precisely the form of the Neumann network estimator (9). Therefore, if R is linear the estimator obtained using a unrolling of gradient descent and the Neumann network estimator are the same. When R is nonlinear we no longer have this equivalence.

4 Theory

The Neumann network architecture proposed in the previous section is motivated by the Neumann series expansion of a (potentially nonlinear) operator R representing the gradient of a regularizer. Strictly speaking, this Neumann series expansion is valid (*i.e.,* corresponds to the solution of the equation (2)) only if R is linear. However, restricting R to to be linear severely limits the class of estimators that can be learned in our framework. In particular, if R is linear then the resulting learned estimator has to be linear, which is suboptimal for many data types.

In this section we show that a Neumann series approach is still mathematically justified for data belonging to a union of subspaces (UoS) model. Our reasons for focusing on a UoS model are two-fold: First, UoS models are a natural generalization of linear subspace models and are used widely in many signal and image reconstruction problems, either as a deterministic model

[52–54] or as a statistical model in the form of a Gaussian mixture model with low-rank covariances [55–57]. Second, we believe UoS models represent a reasonable trade-off between model complexity/expressiveness and analytic tractability, and allow us to provide some insight on the expected behavior of Neumann networks beyond the setting where R is linear.

To be precise, here we consider the class of *Neumann network estimators* $\widehat{\boldsymbol{\beta}}$ given in (9) that are specified by a (potentially nonlinear) mapping[2] $R : \mathbb{R}^p \to \mathbb{R}^p$, step size η, and number of blocks B. We study two questions:

1. Can a Neumann network estimator be used to reconstruct images belonging to a UoS, and if so, what is an optimal choice of R?

2. Can this R be learned using standard neural network architectures and training?

Our main result, given in Theorem 1, addresses the first question by showing there exists a Neumann network estimator with a *piecewise linear* R that gives arbitrarily small reconstruction error under mild assumptions on the subspaces and their interaction with the measurement operator. We study the second question empirically, and show that the learned R well-approximates the predicted optimal piecewise linear R in an idealized setting.

4.1 Images Belonging to a Single Subspace

Suppose the ground truth images belong to an r-dimensional subspace $\mathcal{S} \subset \mathbb{R}^p$. Let $\boldsymbol{U} \in \mathbb{R}^{p \times r}$ be a matrix whose columns form an orthonormal basis for $\mathcal{S}$. Assume $m \geq r$ and $\boldsymbol{X}\boldsymbol{U} \in \mathbb{R}^{m \times r}$ is full rank. In other words, we assume there is no image in the subspace also in the nullspace of $\boldsymbol{X}$ besides the zero image[3]. Then given noise-free linear measurements of the form $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^\star$ of any data point $\boldsymbol{\beta}^\star = \boldsymbol{U}\boldsymbol{w}^\star \in \mathcal{S}$ we can always recover $\boldsymbol{\beta}^\star$ by applying the linear estimator

$$\widehat{\boldsymbol{\beta}}_o(\boldsymbol{y}) = \boldsymbol{U}(\boldsymbol{U}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{U})^{-1}\boldsymbol{U}^\top \boldsymbol{X}^\top \boldsymbol{y} \tag{14}$$

since it is easy to check that $\widehat{\boldsymbol{\beta}}_o(\boldsymbol{y}) = \boldsymbol{\beta}^\star$. In other words, there always exists a linear estimator that gives exact recovery of images belonging to the subspace from their noise-free linear measurements.

Our first result shows that there exists a linear Neumann network estimator of the form (9) (*i.e.,* a linear choice of R in (9)) such that for all points in the subspace the reconstruction error can be made arbitrarily small by choosing the step size η and block size B appropriately. For simplicity, we restrict ourselves to the case of noise-free measurements and where $\boldsymbol{X}$ has orthonormal rows.

Lemma 1. *Let $\boldsymbol{X} \in \mathbb{R}^{m \times p}$ be any measurement matrix with orthonormal rows, and let $\mathcal{S} \subset \mathbb{R}^p$ be an r-dimensional subspace with orthonormal basis $\boldsymbol{U} \in \mathbb{R}^{r \times p}$. Suppose $m \geq r$ and $\boldsymbol{X}\boldsymbol{U} \in$*

[2]Here we do not assume R is a neural network with a particular architecture, but study the idealized case where R can represent any mapping from $\mathbb{R}^p$ to $\mathbb{R}^p$.

[3]This assumption is met in many practical settings. For example, in an inpainting setting it is equivalent to assuming there is no image in the subspace having support contained entirely within the inpainting region. Likewise, in compressed sensing by subsampling DFT coefficients, it is equivalent to assuming there is no image in the subspace bandlimited to the set of unobserved DFT coefficients. Both of these assumptions are reasonable for subspaces spanned by natural images.

$\mathbb{R}^{m\times r}$ *is full rank. Then for any* $\eta \in (0,1]$*, the* B*-term Neumann network estimator* $\widehat{\boldsymbol{\beta}}$ *with linear* $R(\boldsymbol{\beta}) = \boldsymbol{R}\beta$ *where* $\boldsymbol{R} \in \mathbb{R}^{p\times p}$ *is given by*

$$\boldsymbol{R} = -c_{\eta,B}\,(\boldsymbol{I} - \boldsymbol{X}^\top\boldsymbol{X})\boldsymbol{U}(\boldsymbol{U}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{U})^{-1}\boldsymbol{U}^\top\boldsymbol{X}^\top\boldsymbol{X} \tag{15}$$

for a constant $c_{\eta,B}$ *depending only on* η *and* B*, satisfies the error bounds*

$$\|\widehat{\boldsymbol{\beta}}(\boldsymbol{X}\boldsymbol{\beta}^\star) - \boldsymbol{\beta}^\star\| \le (1-\eta)^{B+1}\|\boldsymbol{X}\boldsymbol{\beta}^\star\|. \tag{16}$$

for all $\boldsymbol{\beta}^\star \in \mathcal{S}$*.*

The proof of Lemma 1 is given in the Appendix. The main idea behind the proof is that with this choice of R the Neumann network terms $\tilde{\boldsymbol{\beta}}^{(j)}$ simplify to

$$\tilde{\boldsymbol{\beta}}^{(j)} = a_j\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\beta}^\star + b_j(\boldsymbol{I} - \boldsymbol{X}^\top\boldsymbol{X})\boldsymbol{\beta}^\star \tag{17}$$

for some constants a_j and b_j that satisfy $\sum_j a_j \approx 1$ and $\sum_j b_j \approx 1$. Hence, we have $\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \sum_{j=0}^{B}\tilde{\boldsymbol{\beta}}^{(j)} \approx \boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\beta}^\star + (\boldsymbol{I} - \boldsymbol{X}^\top\boldsymbol{X})\boldsymbol{\beta}^\star = \boldsymbol{\beta}^\star$.

4.2 Images Belonging to a Union of Subspaces

Now we suppose that the images belong to a UoS $\cup_{k=1}^{K}\mathcal{S}_k \subset \mathbb{R}^p$ where, for simplicity, we assume each subspace $\mathcal{S}_k$ has dimension r. For all $k = 1, ..., K$ we let $\boldsymbol{U}_k \in \mathbb{R}^{p\times r}$ denote a matrix whose columns form an orthonormal basis for $\mathcal{S}_k$. Again, we assume $m \ge r$ and $\boldsymbol{X}\boldsymbol{U}_k \in \mathbb{R}^{m\times r}$ is full rank for every $k = 1, ..., K$. In other words, we assume there is no image in the UoS also in the nullspace of $\boldsymbol{X}$ besides the zero image.

Let $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^\star$ be the measurements of any point $\boldsymbol{\beta}^\star$ belonging to the UoS. If we know $\boldsymbol{\beta}^\star$ belongs to the kth subspace, *i.e.,* $\boldsymbol{\beta}^\star = \boldsymbol{U}_k\boldsymbol{w}^\star$ for some $\boldsymbol{w}^\star \in \mathbb{R}^p$, then similar to the single subspace case, we can apply the estimator

$$\widehat{\boldsymbol{\beta}}_o(\boldsymbol{y}; k) := \boldsymbol{U}_k(\boldsymbol{U}_k^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{U}_k)^{-1}\boldsymbol{U}_k^\top\boldsymbol{X}^\top\boldsymbol{y} \tag{18}$$

since it is easy to see that $\boldsymbol{\beta}^\star = \widehat{\boldsymbol{\beta}}_o(\boldsymbol{y}; k)$. We call $\widehat{\boldsymbol{\beta}}_o(\boldsymbol{y}; k)$ the *oracle estimator*, since it assumes knowledge of the subspace index k to which the image belongs.

We show that, under appropriate conditions on the subspaces and the measurement operator, there is a piecewise linear choice of Neumann network estimator (*i.e.,* an estimator of the form (9) with R piecewise linear) that recovers any image belonging to the UoS from its noise-free measurements with arbitrarily small reconstruction error. In other words, there is a Neumann network estimator that well-approximates the oracle estimator.

Specifically, we consider a piecewise linear function R^* of the form

$$R^*(\boldsymbol{\beta}) = \begin{cases} \boldsymbol{R}_1\boldsymbol{\beta} & \text{if } \boldsymbol{\beta} \in \mathcal{C}_1 \\ \vdots & \vdots \\ \boldsymbol{R}_K\boldsymbol{\beta} & \text{if } \boldsymbol{\beta} \in \mathcal{C}_K \end{cases} \tag{19}$$

where each $\boldsymbol{R}_k$ is a $p \times p$ matrix, and the regions $\mathcal{C}_k$ are disjoint and whose union is all of $\mathbb{R}^p$. The main idea behind our analysis is this: If the ground truth point $\boldsymbol{\beta}^\star$ belongs the kth subspace, then

we prove that the Neumann series summands $\tilde{\boldsymbol{\beta}}^{(0)}, \tilde{\boldsymbol{\beta}}^{(1)}, ..., \tilde{\boldsymbol{\beta}}^{(B)}$ all lie in the same region $\mathcal{C}_k$. This means that the same $\boldsymbol{R}_k$ is used in computing each summand, so we can write

$$\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \sum_{j=0}^{B} \tilde{\boldsymbol{\beta}}^{(j)} = \sum_{j=0}^{B} (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta \boldsymbol{R}_k)^j (\eta \boldsymbol{X}^\top \boldsymbol{y}).$$

Choosing $\boldsymbol{R}_k$ to have the same form as in the single subspace case (see Lemma 1), we then will have $\boldsymbol{\beta}^\star = \widehat{\boldsymbol{\beta}}_o(\boldsymbol{y}, k) \approx \widehat{\boldsymbol{\beta}}(\boldsymbol{y})$.

To be exact, we specify $\boldsymbol{R}_k$ and $\mathcal{C}_k$ as follows. Similar to Lemma 1, we choose

$$\boldsymbol{R}_k = -c_{\eta,B} \, (\boldsymbol{I} - \boldsymbol{X}^\top \boldsymbol{X}) \boldsymbol{U}_k (\boldsymbol{U}_k^\top \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{U}_k)^{-1} \boldsymbol{U}_k^\top \boldsymbol{X}^\top \boldsymbol{X},$$

for all $k = 1, ..., K$, where $c_{\eta,B} > 0$ is a constant depending only on η and B. We also define the corresponding region $\mathcal{C}_k$ as

$$\mathcal{C}_k = \{\boldsymbol{\beta} \in \mathbb{R}^p : d_{\boldsymbol{X},k}(\boldsymbol{\beta}) < d_{\boldsymbol{X},\ell}(\boldsymbol{\beta}) \text{ for all } \ell \neq k\}$$

where $d_{\boldsymbol{X},k}(\boldsymbol{\beta}) := \|(\boldsymbol{I} - \boldsymbol{X}\boldsymbol{U}_k(\boldsymbol{X}\boldsymbol{U}_k)^+)\boldsymbol{X}\boldsymbol{\beta}\|$ is the distance between the vector $\boldsymbol{X}\boldsymbol{\beta}$ and the subspace $\operatorname{span}(\boldsymbol{X}\boldsymbol{U}_k)$. In other words, $\mathcal{C}_k$ is the set of all points whose distance to the kth subspace is smaller than the distance to all other subspaces, as measured by the functions $d_{\boldsymbol{X},\ell}$ for all $\ell = 1, ..., K$.

We now state our main theorem:

Theorem 1. *Let $\boldsymbol{X} \in \mathbb{R}^{m \times p}$ be any measurement matrix with orthonormal rows, and for all $k = 1, ..., K$ let $\boldsymbol{U}_k \in \mathbb{R}^{p \times r}$ be an orthonormal basis for the kth subspace $\mathcal{S}_k$ with $\dim \operatorname{span}(\boldsymbol{X}\boldsymbol{U}_k) = r$. Suppose span$(\boldsymbol{X}\boldsymbol{U}_k) \cap$ span$(\boldsymbol{X}\boldsymbol{U}_\ell) = \{0\}$ for all $k \neq \ell$. Then the Neumann network estimator $\widehat{\boldsymbol{\beta}}$ with step size $\eta \in (0, 1)$ and piecewise linear $R = R^*$ as defined in* (19) *satisfies*

$$\|\widehat{\boldsymbol{\beta}}(\boldsymbol{X}\boldsymbol{\beta}^\star) - \boldsymbol{\beta}^\star\| \leq (1 - \eta)^{B+1} \|\boldsymbol{X}\boldsymbol{\beta}^\star\|, \tag{20}$$

for all $\boldsymbol{\beta}^\star \in \cup_{k=1}^K \mathcal{S}_k$.

The condition $\operatorname{span}(\boldsymbol{X}\boldsymbol{U}_k) \cap \operatorname{span}(\boldsymbol{X}\boldsymbol{U}_\ell) = \{0\}$ for all $\ell \neq k$, appearing in Theorem 1 is not overly restrictive if we take into account the statistics of natural images. For instance, this condition holds for a generic union of r-dimensional subspaces provided $m \geq 2r$, regardless of the number of subspaces in the union[4]. Moreover, based on results in compressive sensing using low-rank Gaussian mixture models [55, 58], we conjecture this condition can be weakened under appropriate assumptions on $\boldsymbol{X}$ and appropriate modification of R^*, but we do not pursue this refinement here.

Theorem 1 shows there exists a Neumann network estimator with a certain choice of R^* that well-approximates an oracle estimator for images belonging to a union of subspaces. In principle, since R^* is piecewise linear with a finite number of regions, it is realizable as a sufficiently deep

[4]This is because if $\operatorname{span}(\boldsymbol{U}_k)$, $k = 1, ..., K$, are generic r-dimensional subspaces in $\mathbb{R}^p$, then $\mathcal{V}_k = \operatorname{span}(\boldsymbol{X}\boldsymbol{U}_k)$, $k = 1, ..., K$, are generic r-dimensional subspaces in $\mathbb{R}^m$. Since two generic subspaces are linearly independent provided the sum of their dimensions does not exceed the ambient dimension, we see that $\mathcal{V}_k$ and $\mathcal{V}_\ell$, $k \neq \ell$ collectively span a $2r$-dimensional subspace, which is only possible if their intersection is trivial.

neural network with ReLU activations [59]. However, this does not necessarily mean that a Neumann network estimator with R given by a ReLU network when trained on images belonging to a union of subspaces will recover $R = R^*$ specified in Theorem 1. For example, there may be other R' that yield similar training loss as R^*, or the learned component may be under-parameterized (e.g., not enough layers) in such a way that it cannot well-approximate R^*. Nevertheless, one would hope that given sufficient training data and a sufficiently expressive network architecture for the learned component, it may be possible to learn a good approximation to R^* as specified in Theorem 1. Below we illustrate that this is indeed the case for a Neumann network trained on images belonging to synthetic union of subspaces.

Finally, using the equivalence of Neumann networks and unrolled gradient descent networks estimators in the case where the learned component R is linear (see Sec. 3.3), we show that an unrolled gradient descent network as defined in (3) with the same piecewise linear $R = R^*$ as defined in (19) satisfies the error bounds as in Theorem 1:

Corollary 1. *Under the same assumptions as Theorem 1, the unrolled gradient descent estimator* $\widehat{\boldsymbol{\beta}}'(\boldsymbol{y}) = \boldsymbol{\beta}^{(B)}$ *with step size* $\eta \in (0, 1)$ *and* $R = R^*$ *as defined in* (19) *satisfies*

$$\|\widehat{\boldsymbol{\beta}}'(\boldsymbol{X}\boldsymbol{\beta}^\star) - \boldsymbol{\beta}^\star\| \leq (1-\eta)^{B+1}\|\boldsymbol{X}\boldsymbol{\beta}^\star\|, \tag{21}$$

for all $\boldsymbol{\beta}^\star \in \cup_{k=1}^K \mathcal{S}_k$,

Corollary 1 shows that the equivalence between unrolled gradient descent estimators and Neumann network estimators observed in the case where R is linear carries over to the special case where $R = R^*$ is piecewise linear and the networks are evaluated on linear measurements of points belonging to the union of subspaces.

4.3 Empirical Validation

Here we illustrate empirically that the optimal R^* predicted by Theorem 1 is well-approximated by training a Neumann network for a 1-D inpainting task on synthetic UoS data. We generate random training data belonging to a union of three 3-dimensional subspaces in $\mathbb{R}^{10}$, and train a Neumann network to inpaint five missing coordinates (*i.e.,* $\boldsymbol{X} \in \mathbb{R}^{5\times 10}$ restricts a vector to coordinates 1–5). We parameterize the learned component R of the Neumann Network as a 7-layer fully connected neural network with ReLU activations, which is trained by minimizing the mean squared error of the reconstruction over the training set using stochastic gradient descent (more details on this experiment can be found in the Supplementary Materials).

Figure 4 illustrates the output of the trained Neumann network for one specific input, including the outputs from the intermediate Neumann network terms $\tilde{\boldsymbol{\beta}}^{(j)}$ and the learned component outputs $R(\tilde{\boldsymbol{\beta}}^{(j)})$. As predicted by Theorem 1, the Neumann network terms $\tilde{\boldsymbol{\beta}}^{(j)}$ have the form $a_j \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\beta}^\star + b_j(\boldsymbol{I} - \boldsymbol{X}^\top\boldsymbol{X})\boldsymbol{\beta}^\star$ for some constants a_j and b_j. Also, the outputs of the learned component $R(\tilde{\boldsymbol{\beta}}^{(j)})$ all lie in the null space of $\boldsymbol{X}$, *i.e.,* are vectors supported on coordinates $6-10$.

Figure 5 displays the results of a quantitative experiment to assess whether the learned component R is piecewise linear as predicted by Theorem 1. First, we test whether the learned R is approximately linear when restricted to inputs belonging to each subspace, *i.e.,* we test whether $R(\boldsymbol{\beta}_1^\star + \boldsymbol{\beta}_2^\star) \approx R(\boldsymbol{\beta}_1^\star) + R(\boldsymbol{\beta}_2^\star)$, for all $\boldsymbol{\beta}_1^\star, \boldsymbol{\beta}_2^\star$ belonging to the same subspace. As baselines we compare to the case where $\boldsymbol{\beta}_1^\star$ and $\boldsymbol{\beta}_2^\star$ belong to different subspaces, and the case where $\boldsymbol{\beta}_1^\star$

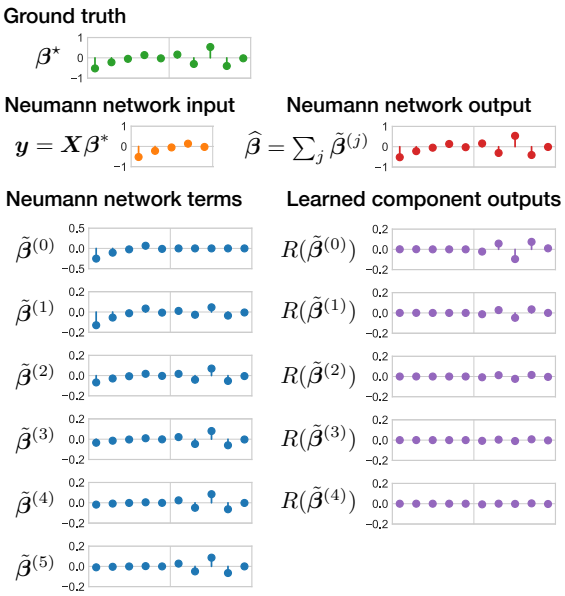


Figure 4: Example output of Neumann network trained on synthetic union of subspaces data for a 1-D inpainting task. Here a vector $\beta^\star \in \mathbb{R}^{10}$ is drawn from one of the subspaces, and its measurements $y = X\beta^*$ (restriction to first five coordinates) are input into the Neumann network, which faithfully restores the missing coordinates. The output $\widehat{\beta}$ of the Neumann network is a sum of terms $\tilde{\beta}^{(j)}$ (shown in bottom left). As predicted by Theorem 1, the terms $\tilde{\beta}^{(j)}$ are weighted linear combinations the projections of $\beta^\star$ onto the observed and unobserved coordinates. Also as predicted by Theorem 1, the outputs of the learned component $R(\tilde{\beta}^{(j)})$ (shown in bottom right) are zero in the observed coordinates and scaled projections of $\beta^\star$ in the unobserved coordinates.

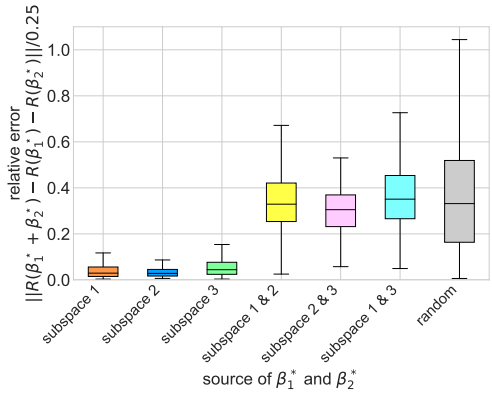


Figure 5: Piecewise linearity test. We measure how linear the learned R is when evaluated at two vectors drawn from the same subspace, from two different subspaces, or from two random Gaussian vectors. The plot illustrates that the learned R only behaves like a linear operator when the vectors belong the same subspace (*i.e.,* the relative error is small), which indicates the learned R is approximately piecewise linear, as predicted by Theorem 1.

and $\beta_2^\star$ are Gaussian random vectors. In Figure 5 we display a boxplot of the relative error $\|R(\beta_1^\star + \beta_2^\star) - R(\beta_1^\star) - R(\beta_2^\star)\|/\gamma$ of 1024 randomly generated $\beta_1^\star, \beta_2^\star$, which are normalized

such that $\|\boldsymbol{\beta}_1^\star\| = \|\boldsymbol{\beta}_2^\star\| = \gamma$. Here we set normalization to $\gamma = 0.25$, though similar results were obtained for $\gamma \in [0.1, 0.5]$ (not shown). As predicted, the relative error concentrates near zero in the case where $\boldsymbol{\beta}_1^\star, \boldsymbol{\beta}_2^\star$ belong to the same subspace, and is otherwise large, indicating the learned R is indeed approximately piecewise linear as predicted by Theorem 1.

In the Supplementary Materials we provide more empirical evidence that the R learned in this experiment closely approximates the ideal R^* predicted by Theorem 1. Specifically, we demonstrate that the learned component behaves as expected on inputs restricted to the column space and row space of the forward model $\boldsymbol{X}$. These experiments verify that, at least for this 1-D inpainting task on synthetic data, the ideal piecewise linear R^* predicted by Theorem 1 is well-approximated with standard neural network architectures and training.

A more systematic study involving different forward models $\boldsymbol{X}$ and different network architectures is needed to determine whether the ideal R^* identified in Theorem 1 is learnable more generally for large-scale imaging data and using practical architectures like convolutional neural networks. Also, our results do not address the case of noisy measurements or forward models with non-orthogonal rows, which are important considerations for many inverse problems. We leave these as open questions for future work.

Finally, while our focus in this section was on UoS models, our analysis does not rule out the applicability of Neumann networks to other non-linear models. Indeed, in the next section we show empirically that Neumann networks perform well on a variety of linear inverse problems when trained on realistic image datasets that are unlikely to be perfectly captured by a low-dimensional UoS model.

5 Experiments

We begin this section with a comparison of Neumann networks against other methods of solving several different inverse problems with learned components. After that, we investigate the effect of larger and smaller training sets on all methods, demonstrating that Neumann networks are robust to small training set sizes. We follow these experiments with an illustration of the effects of incorporating preconditioning into the Neumann network for deblurring, which is shown to give a gain of several dBs of PSNR, permitting smaller networks and allowing for faster training and implementation. We follow with an investigation into an MRI reconstruction problem to demonstrate the proposed methods in a large-scale setting. Finally, we explore the optimization landscape of Neumann networks relative to unrolled gradient descent, illustrating that Neumann networks have smoother loss landscapes than unrolled Gradient Descent, while also generally achieving lower test set errors.

5.1 Datasets and Comparison Methods

In our experiments, we consider three different small-scale training sets: CIFAR10 [60], CelebA [61], and STL10 [62], and one larger-scale undersampled MRI reconstruction task.

The CIFAR10 dataset is a machine learning standard, consisting of real-world images of both man-made and natural scenes [60]. The dataset has been resized to be 32×32 pixels.

We use a subset of the aligned Celebrity Faces With Attributes (CelebA) dataset [61]. The CelebA dataset consists of human faces at a variety of angles, and the subset that is used here has

been aligned so that all faces lie in the center of the image.

The STL10 dataset [62] is a curated subset of the ImageNet dataset, and was originally intended to be used with semisupervised learning problems. In our experiments, we have resized all CelebaA and STL10 images to be 64×64 pixels.

We select a subset of images of size 30,000 uniformly from each individual dataset to be used for training in the results presented below.

We use these training sets in seven different inverse problems in imaging: Block inpainting, deblurring, deblurring with additive noise ϵ of variance 0.01, superresolution (SR4 and SR10) with two different upsampling levels (4x and 10x across the entire image, respectively), and compressive sensing (CS2 and CS8) with two separate levels of compression (2x and 8x, respectively). The compressed sensing design matrices are random Gaussian matrices.

We compare Neumann networks **(NN)** and preconditioned Neumann networks **(PNN)** with four methods which can be applied to solve a variety of inverse problems:

We first compare to the gradient descent network **(GDN)**, an *unrolled optimization* algorithm that is trained end-to-end. While theoretical properties GDN and NN are examined in Section 4, we hope to compare the qualitative and quantitative differences between the two architectures. For a fair comparison, we use an identical architecture for the nonlinear learned component in the gradient descent network and the nonlinear learned component in the Neumann network.

We also compare to MOdel-based reconstruction with Deep Learned priors **(MoDL)** [47], another unrolled algorithm containing a novel data-consistency step that performs conjugate gradient iterations inside the unrolled algorithm. MoDL is also trained end-to-end, and also shares learned parameters between the learned algorithm. In our main experimental section, we use an identical architecture for the learned component of MoDL as is used in GDN and NN.

Trainable Nonlinear Reaction Diffusion **(TNRD)** [33] is an unrolled optimization algorithm that closely resembles GDN, but with a specific, novel architecture for the learned component motivated by insights from diffusion methods for inverse problems. The learned components in each block consist of a single filter, followed by a learned nonlinearity, and then the transpose of the single filter is applied. Weights are *not* shared across blocks in TNRD.

The residual autoencoder **(ResAuto)**, first proposed in [63], is an *agnostic* method. In Section 2, we discussed agnostic methods that learn a mapping from $\boldsymbol{y}$ to $\boldsymbol{\beta}$, but in these experiments, we consider a variant of an agnostic learner that learns a mapping from $\boldsymbol{X}^\top \boldsymbol{y}$ to $\widehat{\boldsymbol{\beta}}$ but does not otherwise use $\boldsymbol{X}$. Specifically, we construct a 12-layer convolutional-deconvolutional residual neural network (almost twice as many layers as the network used in the Neumann network), with a channel-wise fully connected layer.

Compressed Sensing using Generative Models, **(CSGM)** [17] is a *decoupled* method which first trains a generative model for the data. After training the generative model, arbitrary inverse problems can be solved by finding the image in the range of the generator which is closest to the distorted image. As in the setup of [17], in our experiments we train three generative networks, one for each dataset.

Our final method does not incorporate training data at all into the solution of the inverse problem. We reconstruct using total-variation regularized least squares **(TV)**. We minimize our objective using the algorithm of [64], with hyperparameters chosen via cross-validation over a held-out validation set for each dataset and inverse problem.

5.2 Training and Implementation

Given training pairs $\{(\boldsymbol{\beta}_i, \boldsymbol{y}_i)\}_{i=1}^N$, and assuming the learned component R inside the Neumann network depends smoothly on a set of parameters $\boldsymbol{\theta}$, *i.e.,* the partial derivatives $\partial_{\boldsymbol{\theta}} R(\boldsymbol{\beta}; \boldsymbol{\theta})$ exist, we train a Neumann network $\widehat{\boldsymbol{\beta}}$ by minimizing the empirical risk $\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^N \|\widehat{\boldsymbol{\beta}}(\boldsymbol{y}_i; \boldsymbol{\theta}) - \boldsymbol{\beta}_i\|^2$.

In the Supplemental Materials we derive the backpropagation gradients $\partial_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ in the case where $\widehat{\boldsymbol{\beta}}$ is a Neumann network or a gradient descent network.

The learned components of NN, GDN, and MoDL have identical architectures: a 7-layer convolutional-deconvolutional neural network with a single channel-wise fully-connected layer [65], inspired by architectural choices in [63, 66, 67].

For NN, GDN, MoDL, and TNRD we used architectures with $B = 6$ blocks. The learned component is fixed per network, *i.e.,* the learned component in the first block has identical weights to the learned component in all other blocks in any given method and inverse problem, except in TNRD. While using a larger B is possible, we found that increasing B beyond 8 led to greatly increased sensitivity to SGD step size schedule choices. This phenomenon can be observed in Section 5.5. Anecdotally, we find that it is more difficult to choose SGD step sizes for GDN than for NN even for small B, and this difficulty became problematic for B greater than 6.

The ResAuto architecture imitates the architecture of [63], an approach that highly resembles the U-Net [68], but adjusted for good performance on inverse problems like superresolution, deblurring, and inpainting. Superficially, the architecture resembles an expanded version of the previously-described learned component, with 12 convolution or deconvolution layers instead of 7. Further implementation details can be found in supplementary materials.

5.3 Small-scale Experiments

In this section, a variety of methods are used to solve the previously-described inverse problems on three datasets. First, a quantitative comparison in terms of PSNR of the previously-outlined approaches on a variety of datasets and inverse problems is described in Table 1.

We observe that NN and GDN are competitive across all inverse problems and datasets. State-of-the-art methods like MoDL and TNRD perform quite well across all datasets, but the differences in architecture between PNN and MoDL appear to give an edge to PNN, which we hypothesize is an effect of our previously-highlighted skip connections. All methods that incorporate the forward model into the training and reconstruction process perform competitively in our small-scale experiments.

CSGM appears to suffer because of the lack of training data across all experiments. CSGM must learn the manifold associated with each dataset before being able to produce accurate reconstructions, which in our relatively sample-limited setting appears not to happen. See Figure 8 or the Supplement for examples of images produced by CSGM. While TV reconstructions are reasonably accurate across problems, they are not as accurate as learned approaches, especially in inpainting and compressed sensing.

The residual autoencoder in particular has excellent performance on certain problems like inpainting and superresolution, but is not competitive for compressed sensing and deblurring. Recall the motivation for the residual autoencoder: the closer $\boldsymbol{X}^\top \boldsymbol{y}$ is to the ground truth $\boldsymbol{\beta}^\star$, the simpler the residual $\boldsymbol{\beta}^\star - \boldsymbol{X}^\top \boldsymbol{y}$ that the network must learn. With this in mind, it seems reasonable that the residual autoencoder should perform well on small-scale downsampling, inpainting, and de-

		Inpaint	Deblur	Deblur+ϵ	CS2	CS8	SR4	SR10
CIFAR10	NN	28.20	36.55	29.43	33.83	**25.15**	24.48	**23.09**
	PNN	28.40	**37.83**	**30.47**	33.75	23.43	**26.06**	21.79
	GDN	27.76	31.25	29.02	**34.99**	25.00	24.49	20.47
	MoDL	28.18	34.89	29.72	33.47	23.72	24.54	21.90
	TNRD	27.87	34.84	29.70	32.74	25.11	23.84	21.99
	ResAuto	**29.05**	31.04	25.24	18.51	9.29	24.84	21.92
	CSGM	17.88	15.20	14.61	17.99	19.33	16.87	16.66
	TV	25.90	27.57	26.64	25.41	20.68	24.71	20.68
CelebA	NN	**31.06**	31.01	30.43	**35.12**	**28.38**	27.31	23.57
	PNN	30.45	**33.79**	**30.89**	32.61	26.41	**28.70**	23.74
	GDN	30.99	30.19	29.27	34.93	28.33	27.14	23.46
	MoDL	30.75	30.80	29.59	30.22	25.84	26.42	24.12
	TNRD	30.21	29.92	29.79	33.89	28.19	25.75	22.73
	ResAuto	29.66	25.65	25.29	19.41	9.16	25.62	**24.92**
	CSGM	17.75	15.68	15.30	17.99	18.21	18.11	17.88
	TV	24.07	30.96	26.24	25.91	23.01	26.83	20.70
STL10	NN	27.47	29.43	26.12	**31.98**	**26.65**	24.88	21.80
	PNN	28.00	**30.66**	**27.21**	31.40	23.43	**25.95**	**22.19**
	GDN	**28.07**	30.19	25.61	31.11	26.19	24.88	21.46
	MoDL	28.03	29.42	26.06	27.29	23.16	24.67	16.88
	TNRD	27.88	29.33	26.32	31.05	25.38	24.55	21.21
	ResAuto	27.28	25.42	25.13	19.48	9.30	24.12	21.13
	CSGM	16.50	14.04	15.59	16.67	16.39	16.58	16.47
	TV	26.29	29.96	26.85	24.82	22.04	26.37	20.12

Table 1: PSNR comparison for the CIFAR, CelebA, and STL10 datasets respectively. Values reported are the median across a test set of size 256.

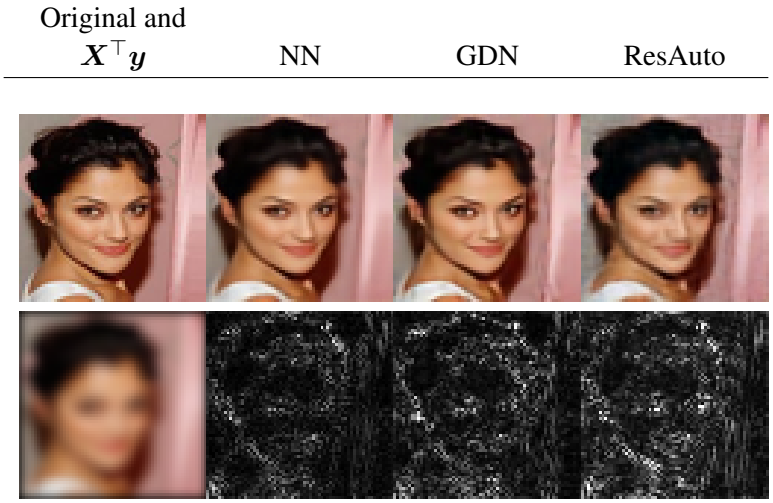


Figure 6: Reconstruction comparison on the CelebA dataset for the deblur plus noise problem. While the Neumann networks (NN) and gradient descent networks (GDN) perform well, the differences are most apparent the residual images in the second row, especially in the background reconstruction. Residuals are formed by displaying the norm across color channels of the error at each pixel, scaled by a factor of 6.

blurring, but would fail to generate high-quality reconstructions for compressed sensing or heavy downsampling where $\boldsymbol{X}^\top \boldsymbol{y}$ is likely to be a poor approximation of $\boldsymbol{\beta}^\star$.

The difference in performance between PNN and NN in Table 1 can provide some insight regarding the usage of various architectures. First, although PNN performs well for 4x super-resolution, deblurring, and deblurring with noise, preconditioning is not a universal solution: inpainting and compressed sensing are perfectly conditioned and preconditioning appears to worsen performance. We see similar effects with MoDL, which performs better than NN or GDN on certain problems, but suffers especially in compressed sensing. These results further emphasize that consideration of the specific forward model at hand should be an important element of designing learned inverse problem solvers.

In addition, we observe some variance in results across datasets. CIFAR10 in particular seems to be a an outlier: while the best reconstructions on CelebA are uniformly more accurate than on STL10, the apparent "difficulty" of reconstruction in CIFAR10 is more task-specific.

Figures 6 and 7 demonstrate more qualitative and quantitative detail in some examples from several different inverse problems and all three datasets. In these figures the residuals are shown for illustrative purposes: the residuals are formed by displaying the scaled pixelwise norm across color channels of the difference $\boldsymbol{\beta}^\star - \widehat{\boldsymbol{\beta}}$ where $\boldsymbol{\beta}^\star$ is the true image, and $\widehat{\boldsymbol{\beta}}$ the estimate, scaled by a factor of 6. Magnitudes are clipped to be less than or equal to 1.

5.4 Effect of Sample Size

In section 2.2 we hypothesized that incorporating information about the forward operator would have implications for the sample sizes required to achieve particular error rates.

A coarse comparison of the presented learning-based methods at different sample sizes is provided in Figure 9a. We observe that while all methods suffer a decrease in PSNR at low sample sizes, the Neumann network has the highest-quality reconstructions at only 2,000 images, and also

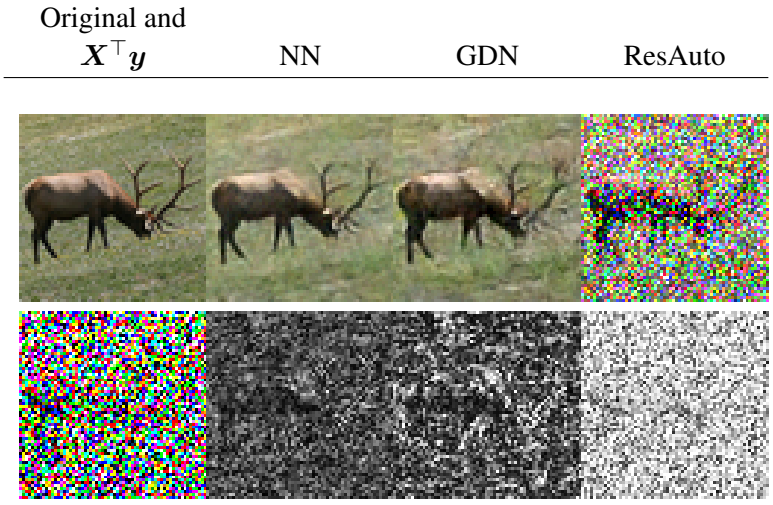


Figure 7: 8x compressed sensing reconstruction comparison on STL10. ResAuto fails to invert the compressed sensing problem adequately. The Gradient Descent network (GDN) reconstructs accurately but generates more artifacts than the Neumann network (NN).

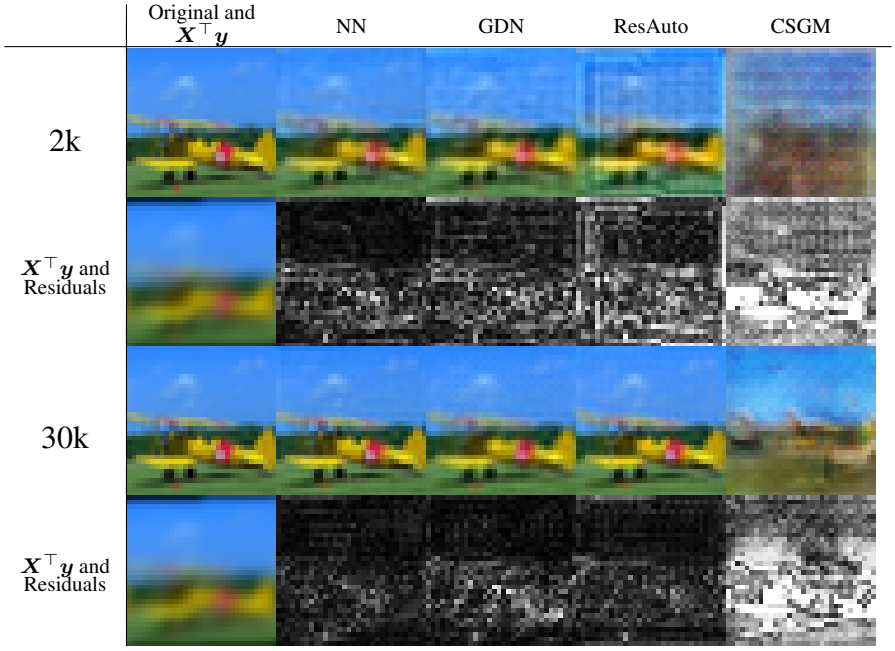
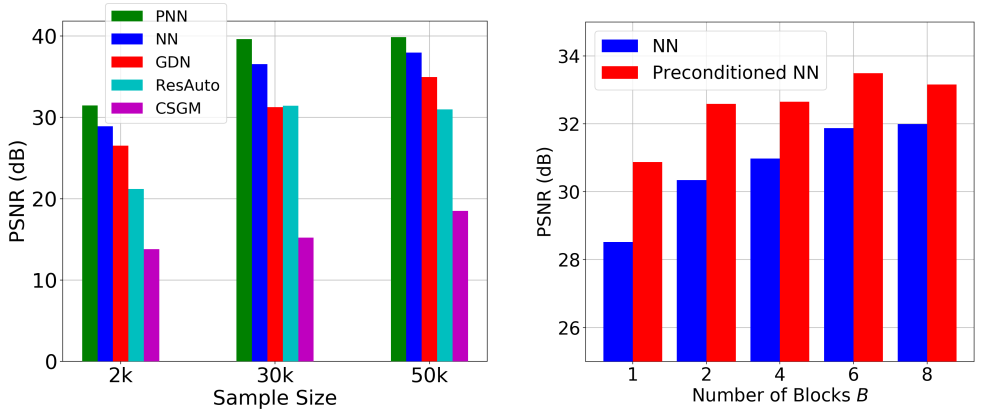


Figure 8: A qualitative comparison of the reconstructions produced for the deblurring problem on a single image at two different training set sizes, along with the associated residual images. Residual images are scaled by a factor of 6.

enjoys the largest increase of performance when going from 2,000 to 30,000 training images. Gradient descent network performs well even at very low sample sizes, but artifacts are present in reconstructions at low sample sizes, visible in Figure 8.

Methods that do not incorporate the forward model, like ResAuto and CSGM, perform poorly in the low-sample regime, as discussed in Section 2.2. While ResAuto performs competitively at 30k iterations, there is little change between image qualities produced at these sample sizes, and



(a) Sample Complexity (b) Preconditioning

Figure 9: Performance comparisons. (a) Median PSNR of methods trained with different sample sizes of 2,000, 30,000, and 50,000. Neumann networks (NN) and Preconditioned Neumann networks (PNN) scale very well with training set size, with smaller marginal gains as training sizes increase. All PSNR values are for the CIFAR-10 dataset, and the inverse problem used is the previously described deblurring problem. (b) PSNR (dB) for the standard and preconditioned NN. The inverse problem in this case is deblurring with a Gaussian kernel of size 5×5 and variance $\sigma = 5.0$.

even a very slight decrease in performance. CSGM improves significantly with increasing samples, but does not produce high-quality reconstructions on this inverse problem.

5.5 Effect of Preconditioning

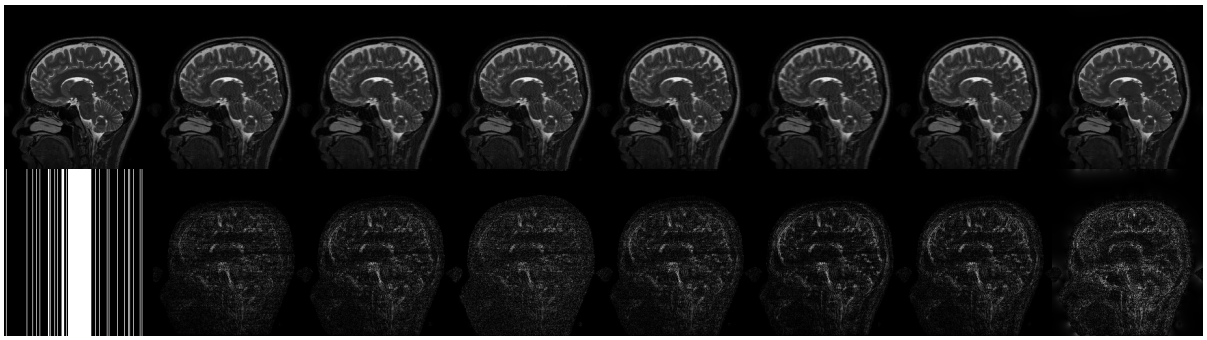
Figure 9b illustrates the effect of preconditioning on the performance of the Neumann network with different numbers of blocks B on a deblurring task. While the original Neumann network does not surpass 32 dB PSNR with 8 blocks, the preconditioned Neumann network surpasses the original with only $B = 2$, and continues to improve as the number of blocks increases. Example images are included in the supplementary materials.

The forward problem in this case is Gaussian deblurring with $\sigma = 5.0$ and a blur kernel of size 5×5. The corresponding $\boldsymbol{X}$ is very poorly conditioned, and a λ of 0.01 is used in the preconditioning matrix $(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1}$.

Depending on the structure of $\boldsymbol{X}$ and how easily $(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1}$ can be computed, preconditioning can be computationally costly, but it appears to permit fewer Neumann network blocks for comparable performance. Since the primary resource bottleneck for training the Neumann network end-to-end is memory, fewer blocks permits faster training, or alternately, allows implementations to achieve higher performance than would otherwise be possible with fixed computational resources.

5.6 MRI Experiments

In this section we provide results of multi-coil MRI reconstruction from undersampled measurements. Full training and test data is the data used for the experiments in [47], consisting of 12-coil



Original/Mask	PNN	NN	MoDL	GDN2	GDN1	TNRD	TV
PSNR (dB)	34.95 dB	33.09 dB	34.09 dB	33.18 dB	31.37 dB	32.39 dB	32.29 dB
Test Time (sec)	16.3 sec	5.5 sec	14.3 sec	5.7 sec	3.1 sec	4.0 sec	349.2 sec

Figure 10: A comparison of MRI reconstruction quality for a variety of trainable and non-trainable image reconstruction methods. The 12-coil data is undersampled by a factor of $4\times$ and Gaussian noise with $\sigma = 0.01$ is added in k-space. The reconstructions are displayed in the first row, while the second row contains the residual images scaled by a factor of 4. PSNR is displayed next to the method name, while below each method name is the mean time required to reconstruct a single MRI image in seconds. GDN2 denotes the Gradient Descent network using the same initialization as the preconditioned Neumann network, while GDN1 uses the same initialization as the Neumann network.

Cartesian sampled k-space data of dimension $232 \times 208 \times 12$ with known coil sensitivity maps. The size of the training set is 360 such acquisitions across 4 subjects, with testing being performed on 40 images from one, separate subject who was not used for training. The sum-of-squares reconstruction is treated as ground truth. Further details of the data acquisition can be found in [47].

All experiments are for $4\times$ undersampling, although we differ from [47] in that we train on a fixed k-space undersampling mask. The undersampling mask is fully sampled in the center 0.15 fraction of frequencies, with the remaining frequencies being sampled according to a random Gaussian pattern. The mask is visualized in figure 10.

For the MRI experiments we follow the precedent set by [47] in our choice of learned component, using only a simple five-layer convolutional network with 64 filters per layer and ReLU nonlinearities for all architectures other than TNRD. The TNRD architecture follows the architecture proposed in [33]. The Neumann network results presented here are for the preconditioned Neumann network (PNN), and the number of blocks for GDN, PNN, MoDL, and TNRD is fixed to be 5. The preconditioning operator in PNN is implemented through 10 conjugate gradient iterations, identically to [47]. We compare to GDN with the same initialization as NN (GDN1) and as PNN (GDN2) to study the effect of different initializations on GDN.

We observe that unrolled optimization approaches are advantageous in this setting compared to the more traditional TV-regularized reconstruction. Preconditioning, both to improve initialization as in GDN2, and incorporated into the architectures, as in PNN and MoDL, improves PSNR significantly in this setting.

A major benefit of learned reconstruction methods is their test time, which is displayed beneath the method name and PSNR in Figure 10. We note that all learned approaches reconstruct an order of magnitude faster than the agnostic TV approach. Although preconditioning incurs an additional cost in terms of test time, the performance increase is substantial for MoDL and PNN.

5.7 Optimization Landscapes

The performance of the Neumann networks (NN) and Gradient Descent networks (GDN) are very similar across a range of problems and datasets, but NN slightly outperforms GDN consistently. We hypothesize this is due to differences in the connectivity of their network architectures and the effect this has on training.

Specifically, both NN and GDN contain connections across blocks, but differ mainly in their *direction* and *extent*. Adjacent blocks in both networks share residual connections as in a ResNet [14] (the inclusion of the identity $\boldsymbol{I}$ in the linear part $[\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X}](\cdot)$ of each block of NN and GDN is a residual connection). However, the main difference is that NN contain additional "skip" connections that connect each block with the final layer, similar to architectures like DenseNets [15]. Recent work [69] has highlighted the role of residual connections in the optimization landscape of deep architectures, implying that residual connections "smooth" the optimization landscape. Specifically, fewer local minima tend to be present, and those minima tend to be wide, as opposed to sharp. In addition, the authors of [69] note that skip connections from intermediate or early layers of deep networks to final layer tend to provide stronger smoothing effects on optimization landscapes than residual connections alone. Hence, we might expect the additional skip connections present in NN also lead to a smoother optimization landscape.

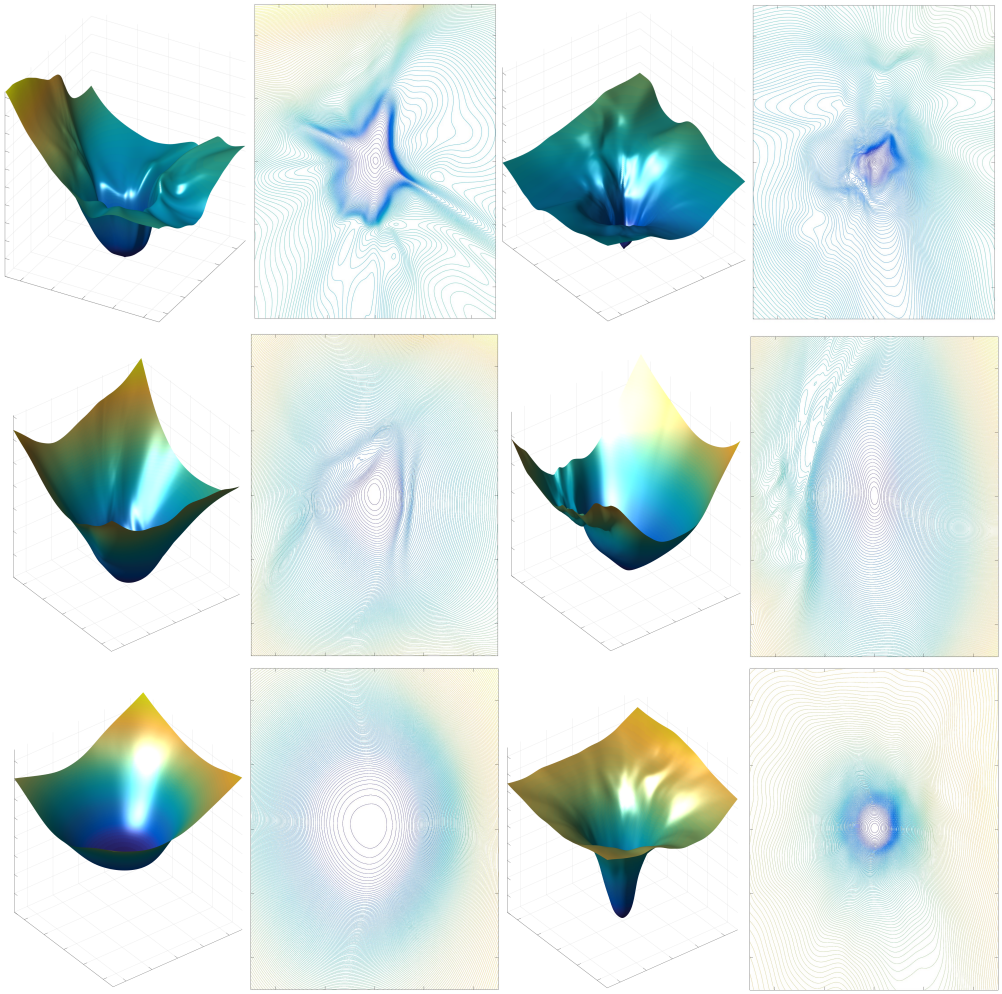
In Figure 11 we illustrate the optimization landscapes using the method of [69], which proposes a procedure for projecting the loss landscape of very high-dimensional models into two dimensions for visualization purposes. Suppose that the fully-trained network has a set of parameters which is vectorized $\widehat{\boldsymbol{\theta}} \in \mathbb{R}^K$. We draw two independent standard Gaussian vectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2$ with dimension K, and normalize them in the manner described in [69] that accounts for the scaling ambiguity of ReLU networks. Then we compute the test and training set error for parameters given by $\widehat{\boldsymbol{\theta}} + \tau(i\boldsymbol{v}_1 + j\boldsymbol{v}_2)$ for step size $\tau > 0$ and integers i, j. The plots above are generated for $i, j \in \{-125, ..., 125\}$ and $\tau = 0.01$. We demonstrate plots for three different forward models: deblurring, compressed sensing with $8\times$ compression, and $10\times$ superresolution.

Figure 11 illustrates several attractive properties of the NN and GDN. Local minima appear to be rare in the neighborhood of the trained minima for GDN and NN. While neither is convex, it is interesting to note that the NN landscapes seem to have much wider basins around minima and higher slope outside this main basin. GDN's optimization landscape appears to require a search around a low-slope landscape until finding a region of high curvature in the deblurring and compressed sensing case, and contains more local minima than NN.

In addition, experimental evidence and some theory indicate that wider local minima have better generalization properties [70]. This does not indicate that one architecture should perform better than another, but if both networks achieve similar training error, wider local minima may translate to better test performance.

6 Discussion and Conclusions

This paper describes a novel network architecture that departs from the currently-popular unrolled optimization framework described in Section 2.3. Our approach is based on the Neumann series expansion for inverting linear operators and has several key features. First, Neumann networks naturally contain "skip connections" [14, 15] that appear to yield optimization landscapes that



(a) Neumann Loss Surface (b) Neumann Contour (c) GDN Loss Surface (d) GDN Contour

Figure 11: Optimization landscapes and contour plots. (a) The optimization landscape associated with the training loss of the Neumann network around the center optimal point. (b) The associated contour plot. (c) The optimization landscape associated with the training loss of the gradient descent network. (d) The associated contour plot. Neumann network landscapes tend to have wider basins around the minimizer and be steeper outside the basin of the minimizer, which are both more favorable to practical optimization by SGD. Figures use the CIFAR-10 dataset and, from top to bottom, deblurring inverse problem with $\sigma = 2.5$, $10\times$ superresolution, and compressed sensing with $8\times$ compression.

facilitate more efficient training, but are absent from previously proposed network architectures.

Our theoretical analysis reveals that when the training data lie in a union of subspaces, the optimal *oracle* estimator that has prior knowledge of both the subspaces in the union and the identity of the subspace to which true image belongs is piecewise linear. We show this piecewise linear oracle estimator can be approximated arbitrarily well by a Neumann network whose learned component coincides with a specific piecewise linear map, which in principle can always be realized by a neural network using ReLU activations. Furthermore, we observe empirically on simulated union-of-subspaces data that the nonlinear learned component in the trained Neumann network well-approximates the specific piecewise linear map predicted by theory. We are unaware of past work on using neural networks to solve inverse problems demonstrating such properties.

Third, we describe a simple preconditioning step that, when combined with the Neumann network architecture, provides an additional increase in reconstruction PSNR and can reduce the number of blocks B needed for accurate reconstruction, which in turn decreases reconstruction computational complexity when the preconditioning can be computed efficiently. As a result, using a truncated series expansion with only B blocks results in a small, bounded approximation error. Finally, we explore the proposed Neumann network's empirical performance on a variety of inverse problems relative to the performance of representative agnostic, decoupled, and unrolled optimization methods described in Section 2.

While this paper has focused on solving linear inverse problems in imaging using training data to train a neural network, *more generally we can think of this paper as a case study in leveraging physical models to guide neural network architecture design.* More specifically, we can think of networks such as the Neumann network as a single large neural network in which a subset of edge weights (*i.e.,* those corresponding to the operation $\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X}$ and other zero-valued "edges" that define the general architecture of Figure 2) are determined by the physical forward model that specifies the inverse problem at hand and are held fixed during training, while the remaining edges (*i.e.,* those that correspond to the operation $R(\cdot)$) can be learned during training. In other words, *we use knowledge of the inverse problem structure to define the neural network architecture.*

This perspective leads to interesting potential avenues for future work. Specifically, our proposed Neumann network is inspired by series expansions for inverting linear operators, but there are alternative methods for inverting nonlinear operators that may yield new challenges and opportunities. For instance, Adomian decompositions and polynomial expansions have been successfully used to solve differential equations with both linear and nonlinear components [71, 72], and so designing future networks inspired by this framework could lead to new theoretical insights beyond what we present above.

In addition, the reader might note that neural networks based on the Neumann series expansion or iterative optimization methods have several repeated blocks, leading to the question of whether standard stochastic gradient descent is the most efficient training regimen. For instance, recent work on "Neural Ordinary Differential Equations" [73] has considered an ODE representation of the operation of a neural network instead of a series of discrete layers and used this perspective to devise training methods that leverage ODE solvers for more efficient training. Such techniques might be leveraged to improve training of Neumann networks.

Appendix

Here we let $\boldsymbol{P_X}$ and $\boldsymbol{P_{X_\perp}}$ denote the projectors onto the row space of $\boldsymbol{X}$ and the null space of $\boldsymbol{X}$, respectively. In particular, $\boldsymbol{P_X} = \boldsymbol{X}^\top \boldsymbol{X}$ and $\boldsymbol{P_{X_\perp}} = \boldsymbol{I} - \boldsymbol{X}^\top \boldsymbol{X}$, since we assume $\boldsymbol{X}$ has orthonormal rows in Lemma 1, Theorem 1, and Corollary 1.

6.1 Proof of Lemma 1

We have $\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \sum_{j=0}^{B} \tilde{\boldsymbol{\beta}}^{(j)}$ where $\tilde{\boldsymbol{\beta}}^{(0)} = \eta \boldsymbol{X}^\top \boldsymbol{y}$, $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^\star$, and

$$\tilde{\boldsymbol{\beta}}^{(j)} = (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R)\tilde{\boldsymbol{\beta}}^{(j-1)} \tag{22}$$

$$= (\boldsymbol{P_{X_\perp}} + (1-\eta)\boldsymbol{P_X} - \eta R)\tilde{\boldsymbol{\beta}}^{(j-1)} \tag{23}$$

for all $j = 1, ..., B$, and where in the last line we used the identity $\boldsymbol{I} = \boldsymbol{P_X} + \boldsymbol{P_{X_\perp}}$.

We show that $R(\boldsymbol{\beta}) = \boldsymbol{R}\boldsymbol{\beta}$ with $\boldsymbol{R}$ as specified in Lemma 1 satisfies the desired error bounds. Define $\boldsymbol{Q} = \boldsymbol{P_{X_\perp}} \boldsymbol{U}(\boldsymbol{U}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{U})^{-1}\boldsymbol{U}^\top \boldsymbol{X}^\top$ so that $\boldsymbol{R} = -c_{\eta,B}\, \boldsymbol{Q}\boldsymbol{X}$. With this choice of $\boldsymbol{R}$ we have $\boldsymbol{P_X}\boldsymbol{R} = 0$, and an easy induction shows

$$\tilde{\boldsymbol{\beta}}^{(j)} = \eta(1-\eta)^j \boldsymbol{X}^\top \boldsymbol{y} - \eta \sum_{k=0}^{j-1} \boldsymbol{R}\tilde{\boldsymbol{\beta}}^{(k)} \tag{24}$$

for all $j \geq 1$. Summing this over $j = 0, 1, ..., B$ gives

$$\begin{aligned}\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) &= \sum_{j=0}^{B} \eta(1-\eta)^j \boldsymbol{X}^\top \boldsymbol{y} - \eta \sum_{j=1}^{B}\sum_{k=0}^{j-1} \boldsymbol{R}\tilde{\boldsymbol{\beta}}^{(k)} \\ &= \sum_{j=0}^{B} \eta(1-\eta)^j \boldsymbol{X}^\top \boldsymbol{y} - \eta \sum_{j=0}^{B-1} (B-j)\boldsymbol{R}\tilde{\boldsymbol{\beta}}^{(j)}\end{aligned} \tag{25}$$

Next, we show we can choose the constant $c_{\eta,B}$ so that the second term above simplifies to

$$-\eta \sum_{j=0}^{B-1} (B-j)\boldsymbol{R}\tilde{\boldsymbol{\beta}}^{(j)} = \boldsymbol{Q}\boldsymbol{y}. \tag{26}$$

Observe that $\boldsymbol{R}^2\tilde{\boldsymbol{\beta}}^{(k)} = 0$ for all $k = 0, ..., j-1$, and so from (24) we have $\boldsymbol{R}\tilde{\boldsymbol{\beta}}^{(j)} = \eta(1-\eta)^j \boldsymbol{R}\boldsymbol{X}^\top \boldsymbol{y}$, which gives

$$\sum_{j=0}^{B-1} (B-j)\boldsymbol{R}\tilde{\boldsymbol{\beta}}^{(j)} = \eta \sum_{j=0}^{B-1} (B-j)(1-\eta)^j \boldsymbol{R}\boldsymbol{X}^\top \boldsymbol{y}.$$

Letting $c_{\eta,B} = \left(\eta^2 \sum_{j=0}^{B-1} (B-j)(1-\eta)^j\right)^{-1}$ we obtain (26). Therefore, combining (25) and (26) we have

$$\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \eta \sum_{j=0}^{B} (1-\eta)^j \boldsymbol{X}^\top \boldsymbol{y} + \boldsymbol{Q}\boldsymbol{y}. \tag{27}$$

Finally, since we assume $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^\star$, we see that $\boldsymbol{X}^\top \boldsymbol{y} = \boldsymbol{P_X}\boldsymbol{\beta}^*$ and $\boldsymbol{Q}\boldsymbol{y} = \boldsymbol{P_{X_\perp}}\boldsymbol{\beta}^*$, and using the fact that $\eta \sum_{j=0}^{B} (1-\eta)^j = 1 - (1-\eta)^{B+1}$ from (27) we have $\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \boldsymbol{\beta}^\star - (1-\eta)^{B+1}\boldsymbol{P_X}\boldsymbol{\beta}^*$ which gives the desired error bound.

6.2 Proof of Theorem 1 and Corollary 1

To prove Theorem 1 we show that if $\boldsymbol{\beta}^\star$ belongs to the kth subspace then R^* acts acts as the linear map $\boldsymbol{R}_k$ when applied to each Neumann network term $\tilde{\boldsymbol{\beta}}^{(j)}$. That is, we show $\tilde{\boldsymbol{\beta}}^{(j)} \in \mathcal{C}_k$ for all $j = 0, ..., B$, where $\tilde{\boldsymbol{\beta}}^{(0)} = \eta \boldsymbol{X}^\top \boldsymbol{y}$ and $\tilde{\boldsymbol{\beta}}^{(j)} = (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{\beta}}^{(j-1)} - \eta R^*(\tilde{\boldsymbol{\beta}}^{(j-1)})$ for $j = 1, ..., B$. The desired error bounds then follow by direct application of Lemma 1.

First, an easy induction shows that

$$\tilde{\boldsymbol{\beta}}^{(j)} = \eta(1-\eta)^j \boldsymbol{X}^\top \boldsymbol{y} - \eta \sum_{i=0}^{j-1} R^*(\tilde{\boldsymbol{\beta}}^{(k)}). \tag{28}$$

Using the fact that $\boldsymbol{P}_{\boldsymbol{X}} R^*(\boldsymbol{\beta}) = 0$ for all $\boldsymbol{\beta} \in \mathbb{R}^p$, and $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^\star$, we have $\boldsymbol{P}_{\boldsymbol{X}} \tilde{\boldsymbol{\beta}}^{(j)} = \eta(1-\eta)^j \boldsymbol{P}_{\boldsymbol{X}} \boldsymbol{\beta}^\star$ for all $j = 0, ..., B$. Since the region $\mathcal{C}_k$ is a cone, in order to prove $\tilde{\boldsymbol{\beta}}^{(j)} \in \mathcal{C}_k$ for all $j = 0, ..., B$ it suffices to show $\boldsymbol{P}_{\boldsymbol{X}} \boldsymbol{\beta}^\star \in \mathcal{C}_k$. This means we need to show $d_{\boldsymbol{X},k}(\boldsymbol{\beta}^\star) < d_{\boldsymbol{X},\ell}(\boldsymbol{\beta}^\star)$ for all $\ell \neq k$, or equivalently,

$$\|(\boldsymbol{I} - \boldsymbol{X}\boldsymbol{U}_k(\boldsymbol{X}\boldsymbol{U}_k)^+)\boldsymbol{X}\boldsymbol{\beta}^\star\| < \|(\boldsymbol{I} - \boldsymbol{X}\boldsymbol{U}_\ell(\boldsymbol{X}\boldsymbol{U}_\ell)^+)\boldsymbol{X}\boldsymbol{\beta}^\star\|$$

for all $\ell \neq k$. Since $\boldsymbol{X}\boldsymbol{U}_k(\boldsymbol{X}\boldsymbol{U}_k)^+$ is projection onto $\text{span}(\boldsymbol{X}\boldsymbol{U}_k)$, and $\boldsymbol{X}\boldsymbol{\beta}^\star \in \text{span}(\boldsymbol{X}\boldsymbol{U}_k)$, we have $(\boldsymbol{I} - \boldsymbol{X}\boldsymbol{U}_k(\boldsymbol{X}\boldsymbol{U}_k)^+)\boldsymbol{X}\boldsymbol{\beta}^\star = 0$ and so $\|(\boldsymbol{I} - \boldsymbol{X}\boldsymbol{U}_k(\boldsymbol{X}\boldsymbol{U}_k)^+)\boldsymbol{X}\boldsymbol{\beta}^\star\| = 0$. Furthermore, since by assumption $\boldsymbol{X}\boldsymbol{\beta}^\star \notin \text{span}(\boldsymbol{X}\boldsymbol{U}_\ell)$ for all $\ell \neq k$, we have $(\boldsymbol{I} - \boldsymbol{X}\boldsymbol{U}_k(\boldsymbol{X}\boldsymbol{U}_k)^+)\boldsymbol{X}\boldsymbol{\beta}^\star \neq 0$, which means $\|(\boldsymbol{I} - \boldsymbol{X}\boldsymbol{U}_\ell(\boldsymbol{X}\boldsymbol{U}_\ell)^+)\boldsymbol{X}\boldsymbol{\beta}^\star\| > 0$, proving the claim.

Similarly, to prove Corollary 1 we need to show $\boldsymbol{\beta}^{(j)} \in \mathcal{C}_k$ for all $j = 0, ..., B$, where $\boldsymbol{\beta}^{(0)} = \eta \boldsymbol{X}^\top \boldsymbol{y} \in \mathcal{C}_k$ and $\boldsymbol{\beta}^{(j)} = (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X})\boldsymbol{\beta}^{(j-1)} - \eta R^*(\boldsymbol{\beta}^{(j-1)}) + \boldsymbol{\beta}^{(0)}$. Since $\boldsymbol{P}_{\boldsymbol{X}} R^*(\boldsymbol{\beta}) = 0$ for all $\boldsymbol{\beta} \in \mathbb{R}^p$, an easy induction shows that

$$\boldsymbol{P}_{\boldsymbol{X}} \boldsymbol{\beta}^{(j)} = \eta \sum_{i=0}^{j} (1-\eta)^i \boldsymbol{P}_{\boldsymbol{X}} \boldsymbol{\beta}^\star. \tag{29}$$

Since each $\boldsymbol{\beta}^{(j)}$ is a scalar multiple of $\boldsymbol{P}_{\boldsymbol{X}} \boldsymbol{\beta}^\star$, by the same argument as above we have $\boldsymbol{\beta}^{(j)} \in \mathcal{C}_k$ for all $j = 0, ..., B$, which proves the claim.

References

[1] R. C. Gonzalez and R. E. Woods, *Digital image processing*, 3rd ed. Pearson, 2007.

[2] J. A. Fessler, "Model-based image reconstruction for MRI," *IEEE Signal Processing Magazine*, vol. 27, no. 4, pp. 81–89, 2010.

[3] I. Elbakri and J. Fessler, "Statistical image reconstruction for polyenergetic X-ray computed tomography," *IEEE Transactions on Medical Imaging*, vol. 21, no. 2, pp. 89–99, 2002.

[4] R. E. Blahut, *Theory of remote image formation*. Cambridge University Press, 2004.

[5] H. H. Barrett and K. J. Myers, *Foundations of image science*. John Wiley & Sons, 2013.

[6] A. N. Tychonoff and V. Arsenin, "Solution of ill-posed problems," *Winston & Sons, Washington*, 1977.

[7] M. A. Figueiredo and R. D. Nowak, "A bound optimization approach to wavelet-based image deconvolution." in *IEEE International Conference on Image Processing (ICIP)*, 2005, pp. 782–785.

[8] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online dictionary learning for sparse coding," in *International Conference on Machine Learning (ICML)*. ACM, 2009, pp. 689–696.

[9] G. Yu, G. Sapiro, and S. Mallat, "Solving inverse problems with piecewise linear estimators: From Gaussian mixture models to structured sparsity," *IEEE Transactions on Image Processing*, vol. 21, no. 5, pp. 2481–2499, 2012.

[10] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: nonlinear phenomena*, vol. 60, no. 1-4, pp. 259–268, 1992.

[11] R. M. Willett and R. D. Nowak, "Platelets: a multiscale approach for recovering edges and surfaces in photon-limited medical imaging," *IEEE Transactions on Medical Imaging*, vol. 22, no. 3, pp. 332–350, 2003.

[12] A. Danielyan, V. Katkovnik, and K. Egiazarian, "BM3D frames and variational image deblurring," *IEEE Transactions on Image Processing*, vol. 21, no. 4, pp. 1715–1728, 2012.

[13] W. Marais and R. Willett, "Proximal-gradient methods for Poisson image reconstruction with BM3D-based regularization," in *IEEE 7th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, 2017, pp. 1–5.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[15] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks." in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, no. 2, 2017, p. 3.

[16] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," *arXiv preprint arXiv:1605.07648*, 2016.

[17] A. Bora, A. Jalal, E. Price, and A. G. Dimakis, "Compressed sensing using generative models," in *International Conference on Machine Learning (ICML)*, 2017, pp. 537–546.

[18] T. Meinhardt, M. Moeller, C. Hazirbas, and D. Cremers, "Learning proximal operators: Using denoising networks for regularizing inverse imaging problems," in *IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1799–1808.

[19] P. Schafheitlin, *Die theorie der Besselschen funktionen*. BG Teubner, 1908, vol. 4.

[20] I. Gohberg and S. Goldberg, *Basic operator theory*. Birkhäuser, 2013.

[21] L. Xu, J. S. Ren, C. Liu, and J. Jia, "Deep convolutional neural network for image deconvolution," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2014, pp. 1790–1798.

[22] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *European Conference on Computer Vision (ECCV)*. Springer, 2014, pp. 184–199.

[23] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. P. Aitken, A. Tejani, J. Totz, and Z. Wang, "Photo-realistic single image super-resolution using a generative adversarial network." in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4681–4690.

[24] Y. Tan, D. Zhang, F. Xu, and D. Zhang, "Motion deblurring based on convolutional neural network," in *International Conference on Bio-Inspired Computing: Theories and Applications*. Springer, 2017, pp. 623–635.

[25] M. Schmidt, N. L. Roux, and F. R. Bach, "Convergence rates of inexact proximal-gradient methods for convex optimization," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2011, pp. 1458–1466.

[26] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[27] J. R. Chang, C.-L. Li, B. Póczos, and B. V. Kumar, "One network to solve them all — Solving linear inverse problems using deep projection models," in *IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 5889–5898.

[28] H. Gupta, K. H. Jin, H. Q. Nguyen, M. T. McCann, and M. Unser, "CNN-based projected gradient descent for consistent CT image reconstruction," *IEEE Transactions on Medical Imaging*, vol. 37, no. 6, pp. 1440–1453, 2018.

[29] S. Efromovich, "Conditional density estimation in a regression setting," *The Annals of Statistics*, vol. 35, no. 6, pp. 2504–2535, 2007.

[30] D. L. Donoho, I. M. Johnstone, G. Kerkyacharian, and D. Picard, "Density estimation by wavelet thresholding," *The Annals of Statistics*, pp. 508–539, 1996.

[31] B. Delyon and A. Juditsky, "On minimax wavelet estimators," *Applied and Computational Harmonic Analysis*, vol. 3, no. 3, pp. 215–228, 1996.

[32] J. Lafferty, H. Liu, and L. Wasserman, "Minimax theory," http://www.stat.cmu.edu/~larry/=sml/Minimax.pdf, 2008, [Online; accessed 07-January-2019].

[33] Y. Chen and T. Pock, "Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1256–1272, 2017.

[34] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *International Conference on Machine Learning (ICML)*, 2010, pp. 399–406.

[35] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM journal on imaging sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[36] P. Sprechmann, A. Bronstein, and G. Sapiro, "Learning efficient structured sparse models," in *International Conference on Machine Learning (ICML)*, 2012, pp. 219–226.

[37] U. S. Kamilov and H. Mansour, "Learning optimal nonlinearities for iterative thresholding algorithms," *IEEE Signal Processing Letters*, vol. 23, no. 5, pp. 747–751, 2016.

[38] S. Diamond, V. Sitzmann, F. Heide, and G. Wetzstein, "Unrolled optimization with deep priors," *arXiv preprint arXiv:1705.08041*, 2017.

[39] M. Mardani, Q. Sun, S. Vasawanala, V. Papyan, H. Monajemi, J. Pauly, and D. Donoho, "Neural proximal gradient descent for compressive imaging," *arXiv preprint arXiv:1806.03963*, 2018.

[40] J. Sun, H. Li, and Z. Xu, "Deep ADMM-Net for compressive sensing MRI," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 10–18.

[41] J. Adler and O. Öktem, "Learned primal-dual reconstruction," *IEEE Transactions on Medical Imaging*, vol. 37, no. 6, pp. 1322–1332, 2018.

[42] U. Schmidt and S. Roth, "Shrinkage fields for effective image restoration," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 2774–2781.

[43] K. Zhang, W. Zuo, S. Gu, and L. Zhang, "Learning deep CNN denoiser prior for image restoration," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2017.

[44] S. Ravishankar, I. Y. Chun, and J. A. Fessier, "Physics-driven deep training of dictionary-based algorithms for MR image reconstruction," in *Asilomar Conference on Signals, Systems, and Computers*, 2017, pp. 1859–1863.

[45] S. Ravishankar, A. Lahiri, C. Blocker, and J. A. Fessler, "Deep dictionary-transform learning for image reconstruction," in *IEEE International Symposium on Biomedical Imaging (ISBI)*, 2018, pp. 1208–1212.

[46] Y. Chun and J. A. Fessler, "Deep BCD-net using identical encoding-decoding CNN structures for iterative image recovery," in *IEEE Image, Video, and Multidimensional Signal Processing Workshop (IVMSP)*. IEEE, 2018, pp. 1–5.

[47] H. K. Aggarwal, M. P. Mani, and M. Jacob, "MoDL: Model based deep learning architecture for inverse problems," *IEEE Transactions on Medical Imaging*, 2018.

[48] ——, "Multi-shot sensitivity-encoded diffusion MRI using model-based deep learning (MoDL-MUSSELS)," *arXiv preprint arXiv:1812.08115*, 2018.

[49] A. Pramanik, H. K. Aggarwal, and M. Jacob, "Off-the-grid model based deep learning (O-MoDL)," *arXiv preprint arXiv:1812.10747*, 2018.

[50] C. Metzler, A. Mousavi, and R. Baraniuk, "Learned D-AMP: Principled neural network based compressive image recovery," in *Advances in Neural Information Processing Systems*, 2017, pp. 1772–1783.

[51] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*. NBS Washington, DC, 1952, vol. 49, no. 1.

[52] E. Elhamifar and R. Vidal, "Sparse subspace clustering," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 2790–2797.

[53] T. Blumensath, "Sampling and reconstructing signals from a union of linear subspaces," *IEEE Transactions on Information Theory*, vol. 57, no. 7, pp. 4660–4671, 2011.

[54] G. Liu, Z. Lin, S. Yan, J. Sun, Y. Yu, and Y. Ma, "Robust recovery of subspace structures by low-rank representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 171–184, 2013.

[55] F. Renna, R. Calderbank, L. Carin, and M. R. Rodrigues, "Reconstruction of signals drawn from a Gaussian mixture via noisy compressive measurements," *IEEE Transactions on Signal Processing*, vol. 62, no. 9, pp. 2265–2277, 2014.

[56] J. Yang, X. Liao, X. Yuan, P. Llull, D. J. Brady, G. Sapiro, and L. Carin, "Compressive sensing by learning a Gaussian mixture model from measurements," *IEEE Transactions on Image Processing*, vol. 24, no. 1, pp. 106–119, 2015.

[57] A. Houdard, C. Bouveyron, and J. Delon, "High-dimensional mixture models for unsupervised image denoising (HDMI)," *SIAM Journal on Imaging Sciences*, vol. 11, no. 4, pp. 2815–2846, 2018.

[58] H. Reboredo, F. Renna, R. Calderbank, and M. R. Rodrigues, "Compressive classification," in *2013 IEEE International Symposium on Information Theory*. IEEE, 2013, pp. 674–678.

[59] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, "Understanding deep neural networks with rectified linear units," in *International Conference on Learning Representations (ICLR)*, 2018.

[60] A. Krizhevsky, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[61] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[62] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *International Conference on Artificial Intelligence and Statistics (AIStats)*, 2011, pp. 215–223.

[63] X. Mao, C. Shen, and Y.-B. Yang, "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 2802–2810.

[64] Y. Wang, J. Yang, W. Yin, and Y. Zhang, "A new alternating minimization algorithm for total variation image reconstruction," *SIAM Journal on Imaging Sciences*, vol. 1, no. 3, pp. 248–272, 2008.

[65] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2536–2544.

[66] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2016.

[67] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 1646–1654.

[68] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer, 2015, pp. 234–241.

[69] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 6389–6399.

[70] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016.

[71] L. Gabet, "The theoretical foundation of the Adomian method," *Computers & Mathematics with Applications*, vol. 27, no. 12, pp. 41–52, 1994.

[72] G. Adomian, *Solving frontier problems of physics: the decomposition method*. Springer Science & Business Media, 2013, vol. 60.

[73] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *arXiv preprint arXiv:1806.07366*, 2018.

[74] H. Chen, Y. Zhang, M. K. Kalra, F. Lin, Y. Chen, P. Liao, J. Zhou, and G. Wang, "Low-dose CT with a residual encoder-decoder convolutional neural network," *IEEE Transactions on Medical Imaging*, vol. 36, no. 12, pp. 2524–2535, 2017.

[75] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

7 Supplement

7.1 Additional Empirical Support for Union of Subspaces Theory

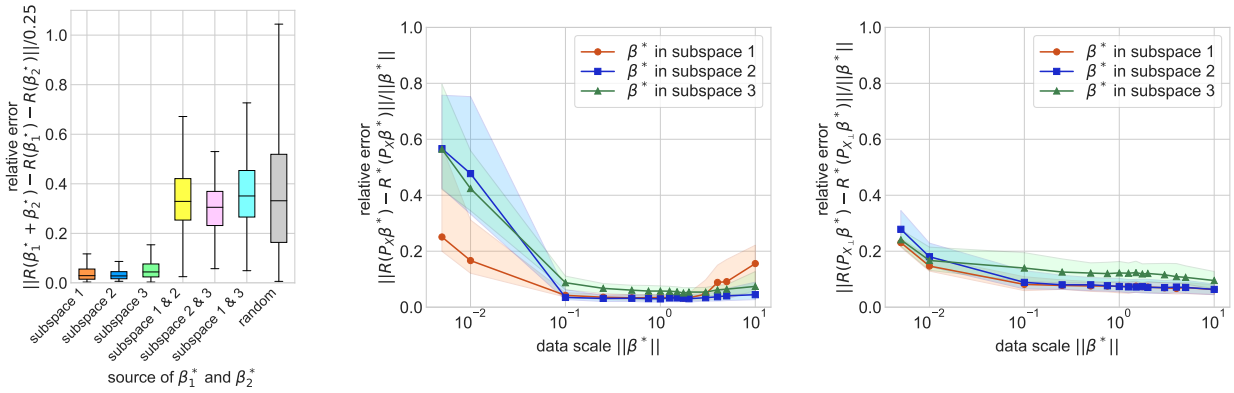
In the experiment discussed in Sec. IV, we train a Neumann network on pairs $(\boldsymbol{\beta}_i, \boldsymbol{y}_i)_{i=1}^N$ where each $\boldsymbol{\beta}_i \in \mathbb{R}^{10}$ belongs one of three randomly chosen three-dimensional subspaces spanned by matrices $\boldsymbol{U}_1, \boldsymbol{U}_2, \boldsymbol{U}_3 \in \mathbb{R}^{10\times 3}$ with orthonormal columns. We generate a random training point as $\boldsymbol{\beta}_i = \boldsymbol{U}_{k_i}\boldsymbol{w}_i$ where we select the index $k_i \in \{1,2,3\}$ uniformly at random and $\boldsymbol{w}_i \in \mathbb{R}^3$ is a random vector with i.i.d. $\mathcal{N}(0,1)$ entries. Here we take $\boldsymbol{X} \in \mathbb{R}^{5\times 10}$ to be the first five rows of the 10×10 identity matrix such that $\boldsymbol{y}_i = \boldsymbol{X}\boldsymbol{\beta}_i$ is the restriction of $\boldsymbol{\beta}_i$ to its first five coordinates. We train a 6-block ($B = 6$) Neumann network where the learned component $R : \mathbb{R}^{10} \to \mathbb{R}^{10}$ is a seven-layer fully-connected neural network with ReLU activations such that the five hidden layers have sizes $(10, 10, 6, 10, 10)$. We learn a set of weights for the learned component and the Neumann network step size η by minimizing the empirical risk using SGD with ADAM acceleration and a batch size of 64, training for 100,000 epochs. When evaluated on a test set of $M = 1024$ points drawn at random from the union of subspaces in the same manner as the training set, the learned component achieves mean squared error $\frac{1}{M}\sum_{i=1}^M \|\widehat{\boldsymbol{\beta}}(\boldsymbol{y}_i) - \boldsymbol{\beta}_i\|^2 = 0.0176$ with variance 0.001, indicating the trained network learned to accurately reconstruct inputs belonging to the union of subspaces.

Figure 12 displays the results of three quantitative experiments to assess whether the learned component R behaves as the piecewise linear R^* predicted by Theorem 1. First, we test whether the learned R is approximately linear when restricted to inputs belonging to each subspace, *i.e.,* we test whether $R(\boldsymbol{\beta}_1^\star + \boldsymbol{\beta}_2^\star) \approx R(\boldsymbol{\beta}_1^\star) + R(\boldsymbol{\beta}_2^\star)$, for all $\boldsymbol{\beta}_1^\star, \boldsymbol{\beta}_2^\star$ belonging to the same subspace. As baselines we compare to the case where $\boldsymbol{\beta}_1^\star$ and $\boldsymbol{\beta}_2^\star$ belong to different subspaces, and the case where $\boldsymbol{\beta}_1^\star$ and $\boldsymbol{\beta}_2^\star$ are Gaussian random vectors. In Figure 12(a) we display a boxplot of the relative error $\|R(\boldsymbol{\beta}_1^\star + \boldsymbol{\beta}_2^\star) - R(\boldsymbol{\beta}_1^\star) - R(\boldsymbol{\beta}_2^\star)\|/\gamma$ of 1024 randomly generated $\boldsymbol{\beta}_1^\star, \boldsymbol{\beta}_2^\star$ normalized such that $\|\boldsymbol{\beta}_1^\star\| = \|\boldsymbol{\beta}_2^\star\| = \gamma$ for the various cases. Here we set normalization to $\gamma = 0.25$, though similar results were obtained for $\gamma \in [0.1, 0.5]$ (not shown). As predicted, the relative error concentrates near zero in the case where $\boldsymbol{\beta}_1^\star, \boldsymbol{\beta}_2^\star$ belong to the same subspace, and is otherwise large, indicating the learned R is indeed approximately piecewise linear as predicted by Theorem 1.

The R^* specified in Theorem 1 acts differently on vectors in the row space of $\boldsymbol{X}$ and the nullspace of $\boldsymbol{X}$. We perform two experiments to verify this is true of our learned R as well.

First, we evaluate R on inputs of the form $\boldsymbol{P}_{\boldsymbol{X}}\boldsymbol{\beta}^\star$ where $\boldsymbol{\beta}^\star$ belongs to one of the three subspaces and $\boldsymbol{P}_{\boldsymbol{X}} := \boldsymbol{X}^\top\boldsymbol{X}$ is the projection onto the row space of $\boldsymbol{X}$. If $\boldsymbol{\beta}^\star$ belongs to one of the three subspaces we have $R^*(\boldsymbol{P}_{\boldsymbol{X}}\boldsymbol{\beta}^\star) = -c_{\eta,B}\boldsymbol{P}_{\boldsymbol{X}_\perp}\boldsymbol{\beta}^\star$, where $c_{\eta,B}$ is a constant depending on η and B. In the present setting ($B = 6$ blocks, and learned $\eta = 0.482$) we have $c_{\eta,B} = 0.349$. In Figure 12(b), we plot the median of the relative error $\|R(\boldsymbol{P}_{\boldsymbol{X}}\boldsymbol{\beta}^\star) - R^*(\boldsymbol{P}_{\boldsymbol{X}}\boldsymbol{\beta}^\star)\|/\|\boldsymbol{\beta}^\star\|$ where $\boldsymbol{\beta}^\star$ is normalized to different scales $10^{-3} \le \|\boldsymbol{\beta}^\star\| \le 10$. Observe that the relative error is small over a wide range of scales, even though the network was trained on inputs $\boldsymbol{\beta}^\star$ with $\|\boldsymbol{\beta}^\star\| \approx 1$, which indicates the learned R generalizes well to other scales.

Next, we evaluate R on inputs of the form $\boldsymbol{P}_{\boldsymbol{X}_\perp}\boldsymbol{\beta}^\star$ where $\boldsymbol{P}_{\boldsymbol{X}_\perp} = \boldsymbol{I} - \boldsymbol{X}^\top\boldsymbol{X}$ denotes projection onto the nullspace of $\boldsymbol{X}$. The predicted output in this case is $R^*(\boldsymbol{P}_{\boldsymbol{X}_\perp}\boldsymbol{\beta}^\star) = 0$. In Figure 12(c), we plot the median of the relative error $\|R(\boldsymbol{P}_{\boldsymbol{X}_\perp}\boldsymbol{\beta}^\star) - R^*(\boldsymbol{P}_{\boldsymbol{X}_\perp}\boldsymbol{\beta}^\star)|/\|\boldsymbol{\beta}^\star\|$ of 1024 randomly generated $\boldsymbol{\beta}^\star$ normalized to various scales. In this case, we find the relative error is low over all scales, again indicating good generalization of the learned R.



(a) Piecewise linearity test (b) Input in row space of $\boldsymbol{X}$ (c) Input in null space of $\boldsymbol{X}$

Figure 12: Empirical support for Theorem 1. We train a Neumann network on synthetic "images" (vectors in $\mathbb{R}^{10}$) belonging to a union of subspaces for a 1-D inpainting problem ($K = 3$ random subspaces each of dimension $r = 3$, $m = 5$ measurements). Our theory predicts that the learned component R should be close to the piecewise linear mapping R^* defined in Theorem 1, and we perform three empirical tests to see if this is true. In (a) we measure how linear R is when evaluated at two vectors drawn from the same subspace, from two different subspaces, or from two random Gaussian vectors. The plot illustrates that the learned R only behaves like a linear operator when the vectors belong the same subspace (*i.e.,* the relative error is small), which indicates R is approximately piecewise linear. In (b) we demonstrate that the learned R behaves like R^* when restricted to inputs of the form $\boldsymbol{P_X}\boldsymbol{\beta}^\star$ (projection of $\boldsymbol{\beta}^\star$ onto the row space of $\boldsymbol{X}$), over several $\boldsymbol{\beta}^\star$ drawn from each subspace uniformly at random, normalized to different scales $\|\boldsymbol{\beta}^\star\|$. Similarly, in (c) we demonstrate the output of the learned R is close to the output of R^* when restricted to inputs of the form $\boldsymbol{P}_{\boldsymbol{X}_\perp}\boldsymbol{\beta}^\star$ (projection of $\boldsymbol{\beta}^\star$ onto the null space of $\boldsymbol{X}$) over a range of scales. In (a) is a box-and-whisker plot of the results from 1024 random trials for each input source. The points plotted in (b) and (c) are the median of the relative error computed over 1024 random trials at that scale, with the shaded region indicating the interquartile range. The learned component was trained on vectors with norm approximately 1, yet we find the learned R generalizes across a range of scales.

7.2 Backpropagation Gradients of Neumann and Gradient Descent Networks

Let $\widehat{\boldsymbol{\beta}}(\boldsymbol{y};\boldsymbol{\theta})$ be a Neumann network estimator depending on parameters $\boldsymbol{\theta}$ (*i.e.,* the parameters defining regularizer network $R(\boldsymbol{\beta};\boldsymbol{\theta})$). Gradients of the empirical risk $\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^N \|\widehat{\boldsymbol{\beta}}(\boldsymbol{y}_i;\boldsymbol{\theta}) - \boldsymbol{\beta}_i\|$ have the form

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{i=1}^N \frac{\partial \widehat{\boldsymbol{\beta}}(\boldsymbol{y}_i;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}^\top \left(\widehat{\boldsymbol{\beta}}(\boldsymbol{y}_i;\boldsymbol{\theta}) - \boldsymbol{\beta}_i\right) \tag{30}$$

where $\frac{\partial \widehat{\boldsymbol{\beta}}(\boldsymbol{y}_i;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ is the Jacobian of $\widehat{\boldsymbol{\beta}}$ with respect to $\boldsymbol{\theta}$ evaluated at $(\boldsymbol{y}_i;\boldsymbol{\theta})$.

Dropping the dependence on $\boldsymbol{y}_i$, we write the Neumann network as a sum $\widehat{\boldsymbol{\beta}}(\boldsymbol{\theta}) = \sum_{j=0}^B \tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta})$ where

$$\tilde{\boldsymbol{\beta}}^{(j+1)}(\boldsymbol{\theta}) = (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta}) - \eta R(\tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta});\boldsymbol{\theta}) \tag{31}$$

with $\tilde{\boldsymbol{\beta}}^{(0)} = \eta \boldsymbol{X}^\top \boldsymbol{y}_i$ and where $R(\boldsymbol{\theta};\boldsymbol{\beta})$ denotes the learned component depending on parameters

$\boldsymbol{\theta}$ evaluated at input $\boldsymbol{\beta}$. Hence, we have

$$\frac{\partial \widehat{\boldsymbol{\beta}}(\boldsymbol{y};\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{j=0}^{B} \frac{\partial \tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{32}$$

The zero order term vanishes because it is constant (assuming η is fixed). To compute the Jacobian of the $\tilde{\boldsymbol{\beta}}^{(j)}$, $j \geq 1$, we use the recursive formula (31) and the chain rule to get:

$$\frac{\partial \tilde{\boldsymbol{\beta}}^{(j+1)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X}) \frac{\partial \tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} - \eta \left(\frac{\partial R(\tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta});\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} + \frac{\partial R(\tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta});\boldsymbol{\theta})}{\partial \boldsymbol{\beta}} \frac{\partial \tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right) \tag{33}$$

$$= \left(\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta \frac{\partial R(\tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta});\boldsymbol{\theta})}{\partial \boldsymbol{\beta}} \right) \frac{\partial \tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} - \eta \frac{\partial R(\tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta});\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{34}$$

To simplify notation, define $F(\boldsymbol{\beta}) = (\boldsymbol{I} - \eta \boldsymbol{X}^T \boldsymbol{X} - \eta R)(\boldsymbol{\beta})$, suppressing the dependence of R on $\boldsymbol{\theta}$, and we write $\partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^\star}$ for the Jacobian of F with respect to $\boldsymbol{\beta}$ evaluated at $\boldsymbol{\beta}^\star$ with the current value of $\boldsymbol{\theta}$. Similarly, write $\partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\boldsymbol{\beta}^\star}$ for the Jacobian of R with respect to $\boldsymbol{\theta}$ evaluated at $\boldsymbol{\beta} = \boldsymbol{\beta}^\star$ with the current value of $\boldsymbol{\theta}$. In this notation, the above becomes:

$$\partial_{\boldsymbol{\theta}} \tilde{\boldsymbol{\beta}}^{(j+1)} = \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j)}} \cdot \partial_{\boldsymbol{\theta}} \tilde{\boldsymbol{\beta}}^{(j)} - \eta \, \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j)}} \tag{35}$$

For example,

$$\partial_{\boldsymbol{\theta}} \tilde{\boldsymbol{\beta}}^{(1)} = -\eta \, \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(0)}} \tag{36}$$

and

$$\partial_{\boldsymbol{\theta}} \tilde{\boldsymbol{\beta}}^{(2)} = -\eta \left(\partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(0)}} + \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} \right) \tag{37}$$

and

$$\partial_{\boldsymbol{\theta}} \boldsymbol{\beta}_3 = -\eta \left(\partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(2)}} \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(0)}} + \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(2)}} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} + \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(2)}} \right) \tag{38}$$

and so in general

$$\partial_{\boldsymbol{\theta}} \boldsymbol{\beta}_k = -\eta \left(\sum_{j'=0}^{j-1} \left(\partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j-1)}} \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j-2)}} \cdot \cdots \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j'+1)}} \right) \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j')}} \right). \tag{39}$$

Therefore,

$$\partial_{\boldsymbol{\theta}} \widehat{\boldsymbol{\beta}} = -\eta \sum_{j=0}^{B} \left(\sum_{j'=0}^{j-1} \left(\partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j-1)}} \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j-2)}} \cdot \cdots \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j'+1)}} \right) \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j')}} \right). \tag{40}$$

Now we perform the same analysis for the gradient descent network. We write a B-block gradient descent network as $\widehat{\boldsymbol{\beta}}'(\boldsymbol{\theta}) = \boldsymbol{\beta}^{(B)}(\boldsymbol{\theta})$ where

$$\boldsymbol{\beta}^{(j+1)}(\boldsymbol{\theta}) = (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X}) \boldsymbol{\beta}^{(j)}(\boldsymbol{\theta}) - \eta R(\boldsymbol{\beta}^{(j)}(\boldsymbol{\theta});\boldsymbol{\theta}) + \boldsymbol{\beta}^{(0)} \tag{41}$$

with $\boldsymbol{\beta}^{(0)} = \eta \boldsymbol{X}^\top \boldsymbol{y}_i$ and where $R(\boldsymbol{\beta}; \boldsymbol{\theta})$ denotes the learned component depending on parameters $\boldsymbol{\theta}$ evaluated at input $\boldsymbol{\beta}$. Similar to the Neumann network case, the derivatives have the recursive formula:

$$\partial_{\boldsymbol{\theta}} \boldsymbol{\beta}^{(j+1)} = \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(j)}} \cdot \partial_{\boldsymbol{\theta}} \boldsymbol{\beta}^{(j)} - \eta \, \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(j)}} \tag{42}$$

And so

$$\partial_{\boldsymbol{\theta}} \widehat{\boldsymbol{\beta}}' = -\eta \left(\sum_{j'=0}^{B-1} \left(\partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(j-1)}} \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(j-2)}} \cdot \, \cdots \, \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(j'+1)}} \right) \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(j')}} \right). \tag{43}$$

To compare the two gradient updates, for a $B = 3$ block Neumann network we have

$$\begin{aligned} \partial_{\boldsymbol{\theta}} \widehat{\boldsymbol{\beta}} = -\eta \Big(& \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(0)}} + \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} + \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(2)}} \\ & + \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(0)}} + \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}_2} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} \\ & + \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(2)}} \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(0)}} \Big). \end{aligned} \tag{44}$$

and for $B = 3$ block gradient descent network we have

$$\partial_{\boldsymbol{\theta}} \widehat{\boldsymbol{\beta}}' = -\eta \left(\partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(2)}} + \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(2)}} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(1)}} + \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(2)}} \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(1)}} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(0)}} \right) \tag{45}$$

8 Implementation Details of the Learned Component of the Neumann and Gradient Descent Networks

Implementation details for the learned components of the Neumann and gradient descent networks are described here, along with details of the ResAuto architecture.

For all networks, bias initialization was a constant at 0.001, while other weights were initialized with a truncated normal with variance 0.05 for Neumann networks and ResAuto, and 0.01 for gradient descent networks. Weight decay was not used.

The learned components of all networks used were convolutional-deconvolutional networks, similar to those used in [27, 74]. The network structures can be described by the sizes and strides of the filters used, along with the nonlinearities. The nonlinearity was chosen to be the ELU [75].

The filters on the network used for CIFAR-10 are square with sizes $5, 3, 2, 3, 3, 5, 3$ and strides $1, 2, 1, 2, 1, 1, 1$. The first three layers convolutional layers, and the last four are convolution transposed ("deconvolution") layers. The number of outputs of each layer are $64, 256, 256, 256, 256, 64, 3$. The channelwise fully-connected layer is after the third, final convolutional layer. The filters on the network used for STL-10 and CelebA are similar to the architecture used on CIFAR-10, except for some adjustments to the number of outputs at each layer, which are $64, 256, 512, 512, 256, 64, 3$.

In the Residual Autoencoder, the filters were square with sizes $4, 4, 4, 4, 4, 2, 4, 4, 4, 4, 4$, with strides $1, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1$ and output sizes $32, 64, 128, 256, 512, 512, 512, 256, 128, 64, 3$. There are residual connections between the second layer and the penultimate layer, and between the input and output layers.

Residual connections were not used in the learned component inside the Neumann network because each block of the network contains an implicit residual connection. Specifically, each block contains an operator of the form $-\eta \boldsymbol{X}^\top \boldsymbol{X} + (\boldsymbol{I} - \eta R)$, where the R is the learned component. Aside from the scaling by $-\eta$, the second term is identical to a network with a residual connection from the input to the output. For almost identical reasons, including a residual connection in the gradient descent network learned component would be redundant as well.

Training was done in Tensorflow, and optimized by ADAM with a beginning step size that was tuned for each problem and decayed exponentially with a decay rate of 0.99 per 500 steps. Beginning step sizes for Neumann networks and gradient descent networks tended to be around 10^{-3}, while the step size for the Residual Autoencoder was closer to 10^{-1}. Anecdotally, gradient descent networks tended to be more sensitive to initialization and step size tuning.

Training was run for 50 epochs across all datasets. Batch size was 32 for the CIFAR tests, and 16 for CelebA and STL networks, due to memory constraints. All training was done on Amazon EC2 p2.xlarge instances on a single NVidia Tesla K80 GPU with 12 GB of GPU memory.

8.1 Time Requirements Comparison of Inverse Problem Solvers

We present here a comparison of the time required to train and test several inverse problem solution methods. The results presented here should be considered relative only to each other, and may be different from problem to problem because of the difference in runtimes for forward problems.

	PNN	NN	GDN	TNRD	MoDL	ResAuto
Train (hours)	4.6	4.2	3.9	1.4	4.7	1.1
Test (sec)	7.0	5.9	5.8	2.2	6.6	1.4

Table 2: Train and testing times for the deblur plus noise reconstruction problem on the CIFAR10 dataset. The train times are in hours, while the test times are listed in seconds for a single batch of size 32.

The time required to train and test any one of the unrolled methods presented here is dependent on a variety of factors. Briefly, one may consider the number of blocks B, the time to run the forward gramian operator $X^\top X$, the complexity of the learned component, and the computational resources at hand. For example, if GPU time is freely available but the forward model is not easily parallelizable and expensive to run, the dominant factor in reconstruction may be the data-consistency terms in all iterative methods.

Overall, however, we find that results are largely intuitive: preconditioning requires more time and for our settings, the learned components dominate time requirements for training. TNRD and ResAuto have approximately the same number of trainable parameters, but TNRD's additional linear components add time.

8.2 Qualitative Comparison of Inverse Problem Solvers

A qualitative comparison on sample CIFAR-10 images for the proposed inverse problems is presented in Figure 13, and similar qualitative comparisons for CelebA and STL10 are presented in Figure 14.

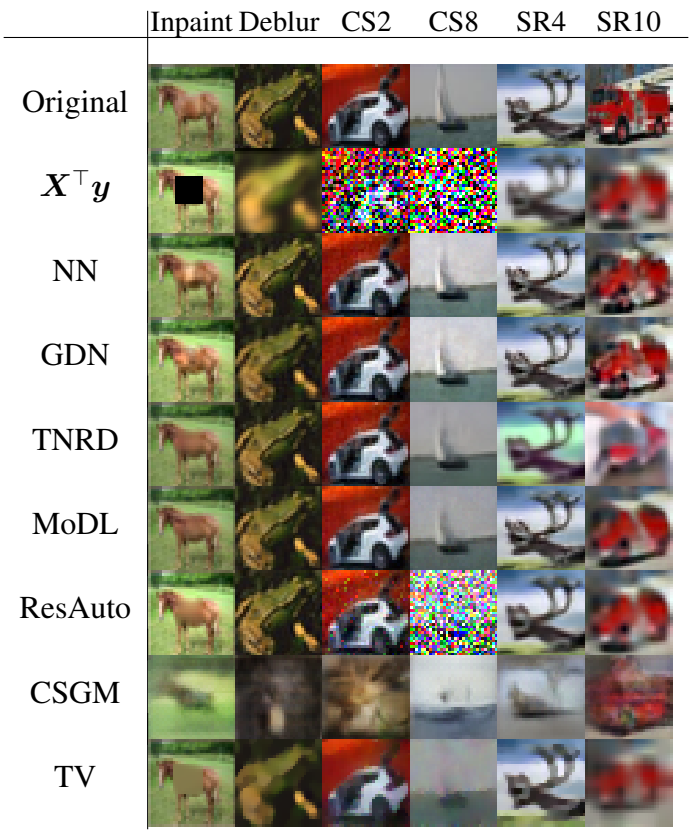


Figure 13: Visual demonstration of inverse problem solutions on the CIFAR-10 dataset. The Network Input row represents $\boldsymbol{X}^\top \boldsymbol{y}$, which is fed into GDN, NN, ResAuto, and TNRD. Inpainting has a 10×10 inpainting region. Deblur has a Gaussian convolutional filter with $\sigma = 2.5$ and filter dimensions 5×5. CS2 has a compression ratio of 2 with a Gaussian sensing matrix, CS8 has a compression ratio of 8 also with a Gaussian sensing matrix. SR (4x) downsamples by a factor of 2 along both dimensions using a linear filter. SR (10x) downsamples by a factor of $\sqrt{10}$ along both dimensions and also uses a linear filter.

Figure 14: Visual demonstration of inverse problem solutions on the CelebA and STL10 dataset. The problems are identical to the CIFAR-10 case.