Neumann Networks for Inverse Problems in Imaging

Davis Gilton, Greg Ongie, Rebecca Willett*

January 13, 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, including in deblurring, tomographic reconstruction, magnetic resonance imaging, radar imaging, image inpainting, compressed sensing, interferometry, and many others [4, 7, 29]. The task of estimating $\boldsymbol{\beta}^\star$ from $\boldsymbol{y}$ is referred to as *image reconstruction*. Classical reconstruction methods assume some prior knowledge about $\boldsymbol{\beta}^\star$ such as smoothness [60], sparsity in some dictionary or basis [26, 43, 63], or geometric properties [54, 61, 17, 45], and attempt to estimate $\widehat{\boldsymbol{\beta}}$ that is both

*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).

a good fit to the observation $\boldsymbol{y}$ and which 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 [32, 33, 38], 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 [9] or learning a denoising autoencoder that can be treated as a regularization step (*i.e.,* proximal operator) within an iterative reconstruction scheme [47].

In this paper, we propose a novel neural network architecture based on the Neumann series expansion [55, 28] 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 training datasets may be smaller.

- Neumann networks naturally yield a block-wise structure with *skip connections* [32] 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 [62]. 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 [21, 39], blind deconvolution [62], and motion deblurring [58], 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 [32].

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 inputs a low-dimensional vector $\boldsymbol{z} \in \mathbb{R}^d$ for $d \ll p$ 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

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

$$\widehat{\boldsymbol{\beta}} = G(\widehat{\boldsymbol{z}}) \tag{2}$$

or, equivalently,

$$\widehat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}=G(\boldsymbol{z}),\boldsymbol{z}\in\mathbb{R}^d}{\arg\min} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2. \tag{3}$$

This approach was described in [9] 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

$$\widehat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\arg\min} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + r(\boldsymbol{\beta}). \tag{4}$$

A proximal gradient algorithm [56, 48] would start with an initial estimate $\boldsymbol{\beta}^{(0)}$ and step size $\eta > 0$ and then compute

for $k = 0, 1, 2, \ldots$

$$\overline{\boldsymbol{\beta}}^{(k)} = \boldsymbol{\beta}^{(k)} + \eta\boldsymbol{X}^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}^{(k)}) \tag{5}$$

$$\boldsymbol{\beta}^{(k+1)} = \underset{\boldsymbol{\beta}}{\arg\min} \frac{1}{2}\|\overline{\boldsymbol{\beta}}^{(k)} - \boldsymbol{\beta}\|_2^2 + r(\boldsymbol{\beta}) \tag{6}$$

Essentially, this algorithm alternates between a (5), gradient descent step that pushes the current estimate towards a better fit to the data, and (6), a *proximal operator* that finds an estimate in the proximity of $\overline{\boldsymbol{\beta}}^{(k)}$ 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 mapping (6) in the form of a denoising autoencoder [47, 10, 31].

The key thing to note 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. However, when we know at training time what $\boldsymbol{X}$ is, then we only need to learn the *conditional* distribution $\phi(\boldsymbol{\beta}|\boldsymbol{X}\boldsymbol{\beta})$ or $\phi(\boldsymbol{\beta}|\boldsymbol{y})$ [24, 6]. 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. Of course, $\phi(\boldsymbol{\beta}|\boldsymbol{X}\boldsymbol{\beta})$ can be calculated from $\phi(\boldsymbol{\beta})$ and $\boldsymbol{X}$ using Bayes' law, but that approach is not always the most sample-efficient.

Specifically, the latter distribution can lie in a much lower-dimensional space, making it easier to learn with a limited amount of data. For instance, suppose the distribution $\phi(\boldsymbol{\beta}) \in \mathcal{B}_\alpha$, where $\mathcal{B}_\alpha$ is the space of all Besov functions in an L_2 norm space with smoothness parameterized by α [19] (larger α implies smoother functions that are easier to estimate). It is well-known how the accuracy of any estimate of $\phi(\boldsymbol{\beta})$ scales with the number of training samples N as a function of α. Specifically, the minimax rate is [23, 18, 37]

$$\min_{\widehat{\phi}} \max_{\phi \in \mathcal{B}_\alpha} \mathbb{E}\|\widehat{\phi} - \phi\|_2 = \mathcal{O}(N^{-\frac{\alpha}{2\alpha+p}}). \tag{7}$$

Recall N is the number of training samples. Since this is the minimax rate, no estimator – regardless of whether it is based on a neural network or not – can achieve better bounds.

In contrast, conditional density estimation can achieve far faster rates when the conditional density only depends on a subset of size p' of the original p coordinates [24, 6]. In such settings, the rate for estimating the conditional density function is $\mathcal{O}(N^{-\frac{\alpha}{2\alpha+p'}})$, meaning that it is possible to achieve a target accuracy with much smaller N in the conditional density setting than when estimating the full density. Consider again the example of image inpainting. Here it is plausible that the distribution of the missing pixels is dependent on the p' observed pixels in the immediate proximity of the missing block but independent of the $p - p'$ pixels further away. Furthermore, the conditional density can be smoother than the full density, *e.g.,* in the space $\mathcal{B}_{\alpha'}$ for some $\alpha' > \alpha$, further accelerating learning rates.

The key point here 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 $\phi(\boldsymbol{\beta}|\boldsymbol{X}\boldsymbol{\beta})$. The latter task can often 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 $\boldsymbol{X}$.

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 [20]. Suppose the desired optimal point $\boldsymbol{\beta}^\star$ satisfies

$$\boldsymbol{\beta}^\star = \arg\min_{\boldsymbol{\beta}} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + r(\boldsymbol{\beta}). \tag{8}$$

This is similar to the starting point in (4), but we proceed in a different direction. Specifically, assume $r(\cdot)$ is differentiable and let $R(\boldsymbol{\beta}) := \nabla r(\boldsymbol{\beta})$ denote the gradient of the regularizer. Then solving (8) can be accomplished using iterative optimization; for instance, gradient descent would result in the iterates

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \eta\left[\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{\beta}^{(k)} - \boldsymbol{y}) + R(\boldsymbol{\beta}^{(k)})\right] \tag{9}$$

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).

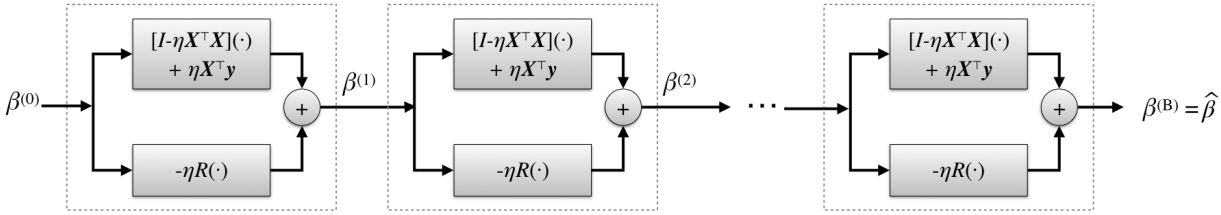


Figure 1: Unrolled gradient descent network. The result of B iterations of gradient descent with a fixed step size η and regularizer with gradient R, as in (5) is equivalent to the output of the above network, with each block corresponding to a single iteration. The network maps a linear function of the measurements, $\boldsymbol{\beta}^{(0)} = \eta\boldsymbol{X}^\top\boldsymbol{y}$ to a reconstruction $\widehat{\boldsymbol{\beta}}$ by successive application of an operator the form $\boldsymbol{I} - \eta\boldsymbol{X}^\top\boldsymbol{X} - \eta R$ and addition of $\eta\boldsymbol{X}^\top\boldsymbol{y}$. Here R is a trained neural network, and the scale parameter η is also trained.

"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 unrolled optimization approach can be applied to a variety of optimization methods beyond gradient descent. The earliest proposed unrolled inverse problem solver was [30], in which the authors proposed unrolling the Iterative Shrinkage and Thresholding Algorithm (ISTA) [5] and

the coordinate descent algorithm. More recent work has illustrated the efficacy of this approach as applied to proximal gradient descent [13, 20, 46], alternating directions method of multipliers [57], primal-dual methods [1], half-quadratic splitting [64], block coordinate descent [51, 14, 52], and alternating minimization [3]. In proximal gradient settings, the learned neural network computes a proximal operator of the form (6), whereas in the gradient descent network, the learned neural network computes the gradient of the regularizer at the input. In other words, different unrolled optimization methods can correspond to different roles played by the learned neural network.

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, [30] 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 [20] achieve promising performance in with $B = 8$ proximal gradient descent iterations.

3 Neumann Networks

Below, we adopt the following strategy. First, we consider the setting in which R is a linear operator and derive a simple Neumann series approximation to the optimal solution. 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.

3.1 Proposed Network Architecture

Our proposed network architecture is motivated by the regularized least squares optimization problem:

$$\min_{\boldsymbol{\beta}} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + r(\boldsymbol{\beta}), \tag{10}$$

where $r : \mathbb{R}^p \to \mathbb{R}$ is a regularizer. Suppose that $r(\boldsymbol{\beta})$ is differentiable. A necessary condition for $\boldsymbol{\beta}^\star$ to be a (local) minimizer of (10) is

$$\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{\beta}^\star - \boldsymbol{y}) + \nabla r(\boldsymbol{\beta}^\star) = 0, \tag{11}$$

or, equivalently,

$$(\boldsymbol{X}^\top\boldsymbol{X} + R)\boldsymbol{\beta}^\star = \boldsymbol{X}^\top\boldsymbol{y}, \tag{12}$$

where we have set $R = \nabla r$.

Assuming the (potentially nonlinear) operator on the left-hand side is invertible, the solution is given by

$$\boldsymbol{\beta}^\star = (\boldsymbol{X}^\top\boldsymbol{X} + R)^{-1}\boldsymbol{X}^\top\boldsymbol{y}. \tag{13}$$

In order to approximate the inverse mapping in (13) we consider a Neumann series expansion of linear operators [55, 28], 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

$$(\boldsymbol{I} - \boldsymbol{A})^{-1} = \sum_{k=0}^{\infty} \boldsymbol{A}^k = \boldsymbol{I} + \boldsymbol{A} + \boldsymbol{A}^2 + \boldsymbol{A}^3 \cdots \tag{14}$$

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:

$$\boldsymbol{B}^{-1} = \eta \sum_{k=0}^{\infty} (\boldsymbol{I} - \eta \boldsymbol{B})^k, \tag{15}$$

which can be obtained through a change of variables. The series in (15) is guaranteed to converge if $\|\boldsymbol{I} - \eta \boldsymbol{B}\| < 1$.

Now, assume for the moment that R is linear and so $R(\boldsymbol{\beta}) = \boldsymbol{R}\boldsymbol{\beta}$ for some matrix $\boldsymbol{R} \in \mathbb{R}^{p \times p}$. Also assume η is chosen such that $\|\boldsymbol{I} - \eta(\boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{R})\| < 1$, then applying the Neumann series expansion to the inverse in (13), we have

$$\boldsymbol{\beta}^\star = \sum_{j=0}^{\infty} (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R)^j (\eta \boldsymbol{X}^\top \boldsymbol{y}) \tag{16}$$

Truncating the series in (16) to $B + 1$ terms, this motivates an estimator $\widehat{\boldsymbol{\beta}}$ of the form

$$\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) := \sum_{j=0}^{B} (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R)^j (\eta \boldsymbol{X}^\top \boldsymbol{y}), \tag{17}$$

Rather than assuming $R = \nabla r$ for some known regularizer r, we turn (17) into a trainable estimator by replacing R with a (potentially nonlinear) mapping depending on a vector of parameters $\boldsymbol{\theta} \in \mathbb{R}^q$ to be learned from training data. In this work we assume R is a neural network, where $\boldsymbol{\theta}$ is a vectorized set of weights and biases that define the network. We also treat the parameter $\eta > 0$ as trainable. The class of estimators $\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \widehat{\boldsymbol{\beta}}(\boldsymbol{y}; \boldsymbol{\theta}, \eta)$ specified (17) with trainable network R we call *Neumann networks*.

Observe that Neumann networks are motivated by an application of the Neumann series identity in the case where R is linear. However, the Neumann network estimator $\widehat{\boldsymbol{\beta}}(\boldsymbol{y})$ in (17) may not be a good solution to the optimality condition (12) for a general nonlinear R. This is because the Neumann series identity (15) only holds for linear operators. However, as we show in the next section, the Neumann network is still mathematically justified for piecewise linear choices of R under certain modeling assumptions on the data distribution. For now, we simply treat the Neumann network estimator as a heuristic motivated by case where R is linear.

To see how (17) can be formulated as a network, observe that the terms in (17) have the following recursive form: let the input to the network be $\tilde{\boldsymbol{\beta}}^{(0)} := \eta \boldsymbol{X}^\top \boldsymbol{y}$ and define

$$\tilde{\boldsymbol{\beta}}^{(j)} := (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R) \tilde{\boldsymbol{\beta}}^{(j-1)} \tag{18}$$

for all $j = 1, ..., B$. Then we have

$$\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \sum_{j=0}^{B} \tilde{\boldsymbol{\beta}}^{(j)}. \tag{19}$$

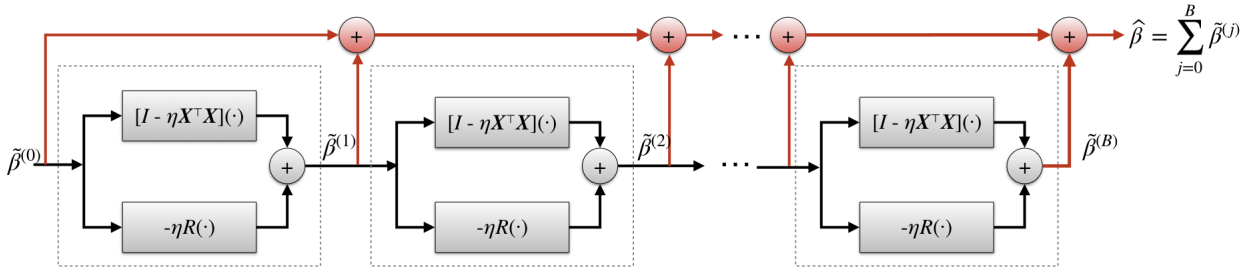


Figure 2: Proposed Neumann network architecture. Inspired by the Neumann series expansion for computing the inverse of an operator, a Neumann network maps a linear function of the measurements, $\widetilde{\boldsymbol{\beta}}^{(0)} = \eta \boldsymbol{X}^\top \boldsymbol{y}$ to a reconstruction $\widehat{\boldsymbol{\beta}}$ by successive application of an operator the form $\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R$ 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 [32] (highlighted in red) that route the output of each dashed block to directly to the output layer.

Figure 2 shows a block diagram for implementing the Neumann network using the recursion (18) and the series structure (19). Each block with a dashed boundary in Figure 2 represents an application of the operator $\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R$. Due to the 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 (ResNets) [32]. These ResNet-like 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 skip connections result in a smoother optimization landscape relative to other unrolling approaches, which allows for easier training via stochastic gradient descent. See Section 5.6 for empirical evidence and more discussion on this point.

3.2 Preconditioning

Efficiently finding a solution to the system (12) using an iterative method is challenging when the operator $\boldsymbol{X}^\top \boldsymbol{X} + R$ is ill-conditioned. This suggests that our Neumann network approach, which is derived from a Neumann series expansion of the system in (12), may benefit from preconditioning of the system. Here we derive a preconditioned Neumann network inspired by Tikhonov regularization.

Starting from (12) we have

$$(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I} - \lambda \boldsymbol{I} + R)\boldsymbol{\beta}^\star = \boldsymbol{X}^\top \boldsymbol{y} \;\text{ for any }\; \lambda > 0, \tag{20}$$

which implies

$$(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})\boldsymbol{\beta}^\star + (R - \lambda \boldsymbol{I})\boldsymbol{\beta}^\star = \boldsymbol{X}^\top \boldsymbol{y}. \tag{21}$$

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

$$(\boldsymbol{I} - [\lambda (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} - \tilde{R}])\boldsymbol{\beta}^\star = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y}. \tag{22}$$

where we have set $\tilde{R} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} R$. Following the same steps used to derive the Neumann network, we arrive at the modified estimator

$$\widehat{\boldsymbol{\beta}}_{pc}(\boldsymbol{y}) = \sum_{j=0}^{B} [\lambda(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} - \tilde{R}]^j (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \tag{23}$$

which we call a *preconditioned Neumann network*. Rather than learning a mappping R we directly learn the mapping $\tilde{R}$. We also treat $\lambda > 0$ as a trainable parameter.

A preconditioned Neumann network has a similar network architecture as the standard Neumann network shown in Figure 2, except the linear unit $[\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X}](\cdot)$ is replaced with $[\lambda(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1}](\cdot)$ and the learned component $[-\eta R](\cdot)$ is replaced with $[-\tilde{R}](\cdot)$. The preconditioned Neumann network also has a different initialization: $\tilde{\boldsymbol{\beta}}^{(0)} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$, which is the solution to the Tikhonov regularized least squares problem

$$\min_{\boldsymbol{\beta}} \frac{1}{2} \|\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}\|^2 + \frac{\lambda}{2} \|\boldsymbol{\beta}\|^2. \tag{24}$$

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 11b) support this observation.

Finally, we note that applying $(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1}$ may be computationally prohibitive for certain large-scale inverse problems in imaging, such as those arising in computed tomography and magnetic resonance imaging. Other preconditioned Neumann networks could be derived by replacing $\lambda \boldsymbol{I}$ in (20) with a general matrix $\boldsymbol{Q}$ such that $\boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{Q}$ is more easily invertible. For simplicity, we focus on the Tikhonov-style preconditioned Neumann network in (23).

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 $\boldsymbol{R} \in \mathbb{R}^{p \times p}$, and consider the loss function (10). The Bth iteration $\boldsymbol{\beta}^{(B)}$ of unrolled gradient descent (9) with step size $\eta > 0$ and initialization $\boldsymbol{\beta}^{(0)} = \eta \boldsymbol{X}^\top \boldsymbol{y}$ can be expanded recursively as follows:

$$\begin{aligned}
\boldsymbol{\beta}^{(B)} &= \boldsymbol{\beta}^{(B-1)} + \eta \boldsymbol{X}^\top (y - \boldsymbol{X}\boldsymbol{\beta}^{(B-1)}) - \eta R \boldsymbol{\beta}^{(B-1)} \\
&= \eta \boldsymbol{X}^\top \boldsymbol{y} + (I - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R) \boldsymbol{\beta}^{(B-1)} \\
&= \eta \boldsymbol{X}^\top \boldsymbol{y} + (I - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R)[\eta \boldsymbol{X}^\top \boldsymbol{y} + (I - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R) \boldsymbol{\beta}^{(B-2)}] \\
&= \eta \boldsymbol{X}^\top \boldsymbol{y} + \eta (I - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R) \boldsymbol{X}^\top \boldsymbol{y} + (I - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R)^2 \boldsymbol{\beta}^{(B-2)} \\
&\quad \vdots \\
&= \eta \sum_{j=0}^{B} (I - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R))^j \boldsymbol{X}^\top \boldsymbol{y},
\end{aligned} \tag{25}$$

which is precisely the form of the Neumann network estimator (17). 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 (12)) only if R is linear. However, restricting R to to be linear severely limits the class of regularizers 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). Our reasons for focusing on UoS models are two-fold: First, they are a natural generalization of linear subspace models and are used widely in many signal and image reconstruction problems [25, 8, 41]. Second, we believe UoS models represent a reasonable trade-off between model complexity/expressiveness and analytic tractability.

To be precise, we consider the class of *Neumann network estimators*

$$\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \sum_{j=0}^{B} (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R)^j (\eta \boldsymbol{X}^\top \boldsymbol{y}). \tag{26}$$

parameterized by a (potentially nonlinear) operator $R : \mathbb{R}^p \to \mathbb{R}^p$, step size $\eta > 0$, and number of blocks B (*i.e.,* the number of terms in the series).

We prove two main existence results: (1) we prove there exists a Neumann network estimator with a *linear* choice of R that recovers images belonging to one subspace with arbitrarily small reconstruction error, and (2) in the case of images belonging to a UoS, we show there exists a Neumann network estimator with a *piecewise linear* choice of R that gives arbitrarily small reconstruction error under mild assumptions on the subspaces and their interaction with the measurement operator. Additionally, we show these results extend to unrolled gradient descent estimators due to their equivalence with Neumann networks for linear R established in Section 3.3.

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. 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

$$\widehat{\boldsymbol{\beta}}_o(\boldsymbol{y}) = \boldsymbol{U}(\boldsymbol{U}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{U})^{-1} \boldsymbol{U}^\top \boldsymbol{X}^\top \boldsymbol{y} \tag{27}$$

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 (26) (*i.e.,* a linear choice of R in (26)) 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 $\boldsymbol{XU} \in \mathbb{R}^{m\times r}$ 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*

$$\boldsymbol{R} = -c_{\eta,B}\,(\boldsymbol{I} - \boldsymbol{X}^\top\boldsymbol{X})\boldsymbol{U}(\boldsymbol{U}^\top\boldsymbol{X}^\top\boldsymbol{XU})^{-1}\boldsymbol{U}^\top\boldsymbol{X}^\top\boldsymbol{X} \tag{28}$$

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

$$\|\widehat{\boldsymbol{\beta}}(\boldsymbol{X}\boldsymbol{\beta}^\star) - \boldsymbol{\beta}^\star\| \leq (1-\eta)^{B+1}\|\boldsymbol{X}\boldsymbol{\beta}^\star\|. \tag{29}$$

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

The proof of Lemma 1 is given in Appendix A. The main idea behind the proof is that with this choice of R the Neumann network terms $\tilde{\boldsymbol{\beta}}^{(j)}$, $j = 0, 1, ..., B$, simplify to

$$\tilde{\boldsymbol{\beta}}^{(j)} = a_j\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\beta}^\star + b_j(\boldsymbol{I} - \boldsymbol{X}^\top\boldsymbol{X})\boldsymbol{\beta}^\star \tag{30}$$

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 $\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \sum_{j=0}^{B}\tilde{\boldsymbol{\beta}}^{(j)} \approx \boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\beta}^\star + (\boldsymbol{I} - \boldsymbol{X}^\top\boldsymbol{X})\boldsymbol{\beta}^\star = \boldsymbol{\beta}^\star$.

Since the Neumann network and unrolled gradient descent estimators coincide in the case where R is linear (see Section 3.3), Lemma 1 also holds for the equivalent unrolled gradient descent network with the same choice of R, η, and B.

4.2 Images Belonging to a Union of Subspaces

Now we suppose that the images belong to a union of subspaces $\cup_{k=1}^{K}\mathcal{S}_k \subset \mathbb{R}^p$ 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 $m \geq r$ and $\boldsymbol{XU}_k \in \mathbb{R}^{m\times r}$ is full rank for every $k = 1, ..., K$. This assumption is met in many practical settings, including inpainting when pixels are missing uniformly at random and the subspaces are chosen at random.

Let $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^\star$ be the measurements of any point $\boldsymbol{\beta}^\star$ belonging to the union of subspaces. 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

$$\widehat{\boldsymbol{\beta}}_o(\boldsymbol{y}; k) := \boldsymbol{U}_k(\boldsymbol{U}_k^\top\boldsymbol{X}^\top\boldsymbol{XU}_k)^{-1}\boldsymbol{U}_k^\top\boldsymbol{X}^\top\boldsymbol{y} \tag{31}$$

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 (26) 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

$$R^*(\boldsymbol{\beta}) = \begin{cases} \boldsymbol{R}_1\boldsymbol{\beta} & \text{if } \boldsymbol{\beta} \in \mathcal{C}_1 \\ \boldsymbol{R}_2\boldsymbol{\beta} & \text{if } \boldsymbol{\beta} \in \mathcal{C}_2 \\ \vdots & \vdots \\ \boldsymbol{R}_K\boldsymbol{\beta} & \text{if } \boldsymbol{\beta} \in \mathcal{C}_K \end{cases} \tag{32}$$

where each $\boldsymbol{R}_k$ is a $p \times p$ matrix, and the regions $\mathcal{C}_k$ are disjoint with $\mathbb{R}^p = \cup_{k=1}^K \mathcal{C}_k$. The main idea behind our analysis is this: If the ground truth point $\boldsymbol{\beta}^\star$ belongs the kth subspace, 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

$$\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \sum_{j=0}^{B} \tilde{\boldsymbol{\beta}}^{(j)} = \sum_{j=0}^{B} (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta \boldsymbol{R}_k)^j (\eta \boldsymbol{X}^\top \boldsymbol{y}). \tag{33}$$

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

$$\boldsymbol{R}_k(\boldsymbol{\beta}) = -c_{\eta,B}\, (\boldsymbol{I} - \boldsymbol{X}^\top \boldsymbol{X}) \boldsymbol{U}_k (\boldsymbol{U}_k^\top \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{U}_k)^{-1} \boldsymbol{U}_k^\top \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{\beta}, \tag{34}$$

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

$$\mathcal{C}_k = \{\boldsymbol{\beta} \in \mathbb{R}^p : d_{\boldsymbol{X},k}(\boldsymbol{\beta}) < d_{\boldsymbol{X},\ell}(\boldsymbol{\beta}) \text{ for all } \ell \neq k\} \tag{35}$$

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 prove this choice of R gives the following 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* (32) *satisfies*

$$\|\widehat{\boldsymbol{\beta}}(\boldsymbol{X}\boldsymbol{\beta}^\star) - \boldsymbol{\beta}^\star\| \leq (1 - \eta)^{B+1} \|\boldsymbol{X}\boldsymbol{\beta}^\star\|. \tag{36}$$

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. In fact, this condition holds for a generic union of r-dimensional subspaces provided $m \geq 2r$, regardless of the number of subspaces in the union. 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.

Finally, using the equivalence of Neumann networks and unrolled gradient descent networks in the case where R is linear, we show that an unrolled gradient descent network with the same piecewise linear R^* satisfies the same 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* (32) *satisfies*

$$\|\widehat{\boldsymbol{\beta}}'(\boldsymbol{X}\boldsymbol{\beta}^\star) - \boldsymbol{\beta}^\star\| \leq (1-\eta)^{B+1}\|\boldsymbol{X}\boldsymbol{\beta}^\star\|. \tag{37}$$

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

4.3 Empirical Support for Union of Subspaces Theory

Theorem 1 shows that 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. This does not necessarily mean that a Neumann network trained on images belonging to a union of subspaces will recover the form of R^* specified in Theorem 1. For one, there may be other R' that yield similar training loss as R^*. Alternatively, the learned component may be under-parameterized in such a way that it cannot well-approximate R^*. Nevertheless, one would hope that given sufficient training data and a sufficiently expressive architecture for the learned component, it would be possible to learn R^* as specified in Theorem 1. Here we verify that this is indeed the case for a Neumann network trained on images belonging to synthetic union of subspaces for a 1-D inpainting task.

Specifically, 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 3 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 $6-10$.

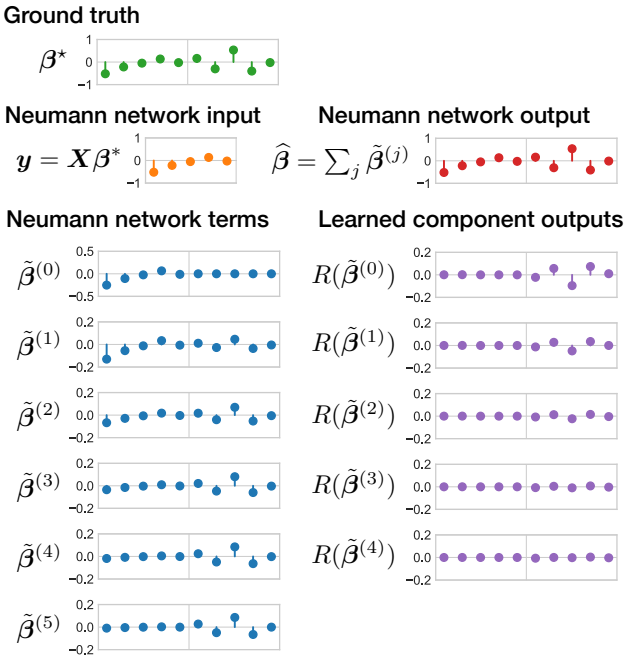
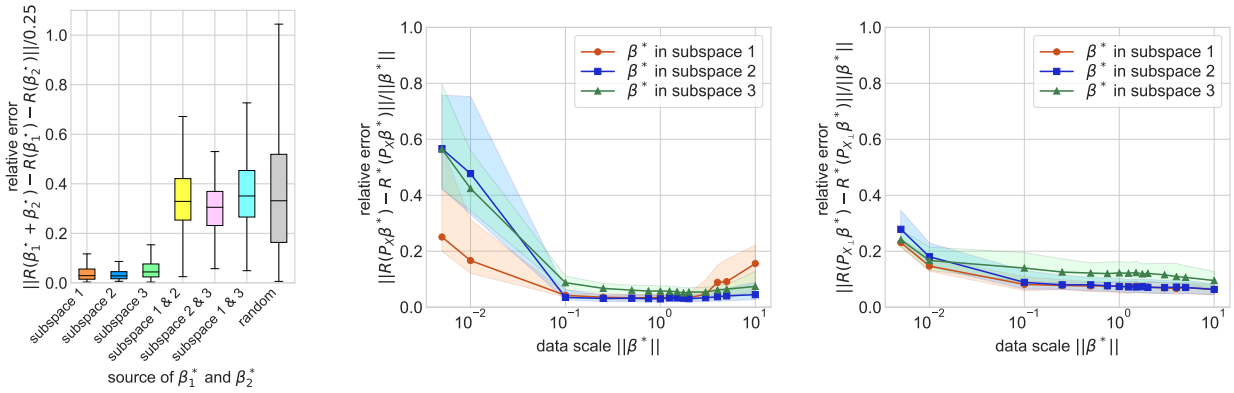


Figure 3: Example output of Neumann network trained on synthetic union of subspaces data for a 1-D inpainting task. Here a vector $\boldsymbol{\beta}^\star \in \mathbb{R}^{10}$ is drawn from one of the subspaces, and its measurements $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^*$ (restriction to first five coordinates) are input into the Neumann network, which faithfully restores the missing coordinates. The output $\widehat{\boldsymbol{\beta}}$ of the Neumann network is a sum of terms $\tilde{\boldsymbol{\beta}}^{(j)}$ (shown in bottom left) defined recursively as $\tilde{\boldsymbol{\beta}}^{(j)} = (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R)\tilde{\boldsymbol{\beta}}^{(j-1)}$, where $R(\cdot)$ is a learned neural network. As predicted by Theorem 1, the terms $\tilde{\boldsymbol{\beta}}^{(j)}$ are weighted linear combinations the projections of $\boldsymbol{\beta}^\star$ onto the observed and unobserved coordinates. Also, predicted by Theorem 1, the outputs of the learned component $R(\tilde{\boldsymbol{\beta}}^{(j)})$ (shown in bottom right) are zero in the observed coordinates are scaled projections of $\boldsymbol{\beta}^\star$ in the missing coordinates.

Figure 4 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 4(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 where $\boldsymbol{P}_{\boldsymbol{X}} := \boldsymbol{X}^\top \boldsymbol{X}$ 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 specified in (47). In the present setting ($B = 6$ blocks, and learned $\eta = 0.482$) we have $c_{\eta,B} = 0.349$. In Figure 4(b), we plot the



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

Figure 4: 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 $\boldsymbol{P_{X_\perp}}\boldsymbol{\beta}^\star$ (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.

median of the relative error $\|R(\boldsymbol{P_X}\boldsymbol{\beta}^\star) - R^*(\boldsymbol{P_X}\boldsymbol{\beta}^\star)\|/\|\boldsymbol{\beta}^\star\|$ where $\boldsymbol{\beta}^\star$ is normalized to different scales $10^{-3} \leq \|\boldsymbol{\beta}^\star\| \leq 10$. 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 $\boldsymbol{P_{X_\perp}}\boldsymbol{\beta}^\star$ where $\boldsymbol{P_{X_\perp}} = \boldsymbol{I} - \boldsymbol{X}^\top\boldsymbol{X}$ denotes projection onto the nullspace of $\boldsymbol{X}$. The predicted output in this case is $R^*(\boldsymbol{P_{X_\perp}}\boldsymbol{\beta}^\star) = 0$. In Figure 4(c), we plot the median of the relative error $\|R(\boldsymbol{P_{X_\perp}}\boldsymbol{\beta}^\star) - R^*(\boldsymbol{P_{X_\perp}}\boldsymbol{\beta}^\star)|/\|\boldsymbol{\beta}^\star\|$ 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.

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 the 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, permitting smaller networks and allowing for faster training and implementation. Finally, we explore the optimization landscape of Neumann networks relative to unrolled gradient descent, illustrating that Neumann networks are more robust to parameter (and hence training data) perturbations, while also achieving lower test set errors.

5.1 Datasets and Comparison Methods

In our experiments, we consider three different training sets.

- **CIFAR10:** The CIFAR10 dataset is a machine learning standard, consisting of real-world images of both man-made and natural scenes [36]. The overall dataset has 50,000 training color images, which are all labeled with one of 10 labels: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Here we use a resized, full-color version of the dataset. The dataset has been resized to be 32×32 pixels, and we have limited the size of the training set to be a single random subset of the dataset of size 30,000.

- **CelebA:** We use a subset of the aligned Celebrity Faces With Attributes (CelebA) dataset [42], where the images have been resized to be 64×64, and a uniformly random subset of size 30,000 has been chosen as a training set. 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.

- **STL10:** The STL10 dataset [16] is a curated subset of the ImageNet dataset, and was originally intended to be used with semisupervised learning problems. It consists of over 100,000 color images containing natural scenes belonging to 10 classes: airplane, bird, car, cat, deer, dog, horse, monkey, ship, and truck. In our experiments, we have resized all STL10 images to be 64×64 pixels, as with the CelebA dataset, and use a single uniformly at random chosen set of size 30,000 as the training set.

We use these training sets in six different inverse problems in imaging: Block inpainting, deblurring, 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).

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

1. **Gradient Descent Network (GDN).** We first compare to the gradient descent network, an *unrolled optimization* algorithm that is trained end-to-end. While the 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 set the nonlinear learned component in the gradient descent network to be identical to the nonlinear learned component in the Neumann network. This nonlinear learned component is a 7-layer convolutional-deconvolutional architecture with a channel-wise fully connected layer.

2. **The Residual Autoencoder (ResAuto).** The residual autoencoder (ResAuto), first proposed in [44], 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 $\boldsymbol{X}$. We do this because all other comparison methods use $\boldsymbol{X}^\top \boldsymbol{y}$ as an input. Specifically, we construct a 12-layer convolutional-deconvolutional neural network (almost twice as many layers as the network used in the Neumann network), with a channel-wise fully connected layer at the center. If this layer is omitted, our reconstruction quality is no longer competitive.

3. **Compressed Sensing using Generative Models (CSGM).** CSGM [9] 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 [9], in our experiments we train three DCGANs [50], one for each dataset. Our DCGAN architecture is identical to theirs, but was trained anew because of different training set sizes.

4. **LASSO.** Our final method does not incorporate traning data at all into the solution of the inverse problem. We use the LASSO [59], assuming sparsity in the blockwise Discrete Cosine Transform domain on blocks of size 16×16 for our reconstruction. Results reported are for the best parameter values, determined by cross-validation.

5.2 Training and Implementation Details

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.,* $R(\boldsymbol{\beta}) = R(\boldsymbol{\beta}; \boldsymbol{\theta})$ and the partial derivatives $\partial_{\boldsymbol{\theta}} R(\boldsymbol{\beta}; \boldsymbol{\theta})$ exist, we train a Neumann network $\widehat{\boldsymbol{\beta}}(\boldsymbol{y}; \boldsymbol{\theta})$ by minimizing the empirical risk:

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \|\widehat{\boldsymbol{\beta}}(\boldsymbol{y}_i; \boldsymbol{\theta}) - \boldsymbol{\beta}_i\|^2. \tag{38}$$

In Appendix B we derive the backpropagation updates for minimizing (38) in the case where $\widehat{\boldsymbol{\beta}}$ is a Neumann network or a gradient descent network.

The learned components of the Neumann and gradient descent networks have identical architectures, which is described here. Based on architectural choices in [22, 44, 35], we used 7-layer convolutional-deconvolutional neural networks as the learned components of the Neumann and gradient descent networks presented below. We found the addition of a single channel-wise fully-connected layer [49] greatly improved the performance of both NN and GDN.

For NN and GDN we used architectures with $B = 6$ blocks. The learned component was fixed per network, *i.e.,* the learned component in the first block had identical weights to the learned component in all other blocks for a given architecture. 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 [44], an approach that highly resembles the U-Net [53], 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. The first 6 layers are convolutional, followed by a channelwise fully-connected layer and 5 convolution-transpose layers. 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.

Further implementation details can be found in Appendix C.

5.3 Relative Empirical Performance of Inverse Problem Solvers

In this section, a variety of methods are used to solve the previously-described inverse problems on three datasets. The methods used for comparison are intended to provide an illustration of different inverse problem solution paradigms.

A qualitative comparison on sample CIFAR-10 images for the proposed inverse problems is presented in Figure 5, and similar qualitative comparisons for CelebA and STL10 are presented in Figure 6. The results in terms of PSNR are described in Table 1.

We observe that NN and GDN are competitive across all inverse problems and datasets. ResAuto does especially well across datasets for inpainting and superresolution as expected, while failing to produce accurate reconstructions for compressed sensing. CSGM appears to suffer because of the lack of training data.

		Inpaint	Deblur	CS2	CS8	SR4	SR10
CIFAR10	NN	28.20	**36.55**	33.83	**25.15**	24.48	**23.09**
	GDN	27.76	31.25	**34.99**	25.00	24.49	20.47
	ResAuto	**29.05**	31.04	18.51	9.29	**24.84**	21.92
	CSGM	17.88	15.20	17.99	19.33	16.87	16.66
	LASSO	19.34	23.70	22.74	16.37	20.03	19.93
CelebA	NN	**31.06**	**31.01**	**35.12**	**28.38**	**27.31**	23.57
	GDN	30.99	30.19	34.93	28.33	27.14	23.46
	ResAuto	29.66	25.65	19.41	9.16	25.62	**24.92**
	CSGM	17.75	15.68	17.99	18.21	18.11	17.88
	LASSO	15.99	14.82	24.37	17.61	16.56	22.74
STL10	NN	27.47	29.43	**31.98**	**26.65**	**24.88**	**21.80**
	GDN	**28.07**	**30.19**	31.11	26.19	**24.88**	21.46
	ResAuto	27.28	25.42	19.48	9.30	24.12	21.13
	CSGM	16.50	14.04	16.67	16.39	16.58	16.47
	LASSO	18.70	16.54	23.14	17.46	18.79	21.36

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

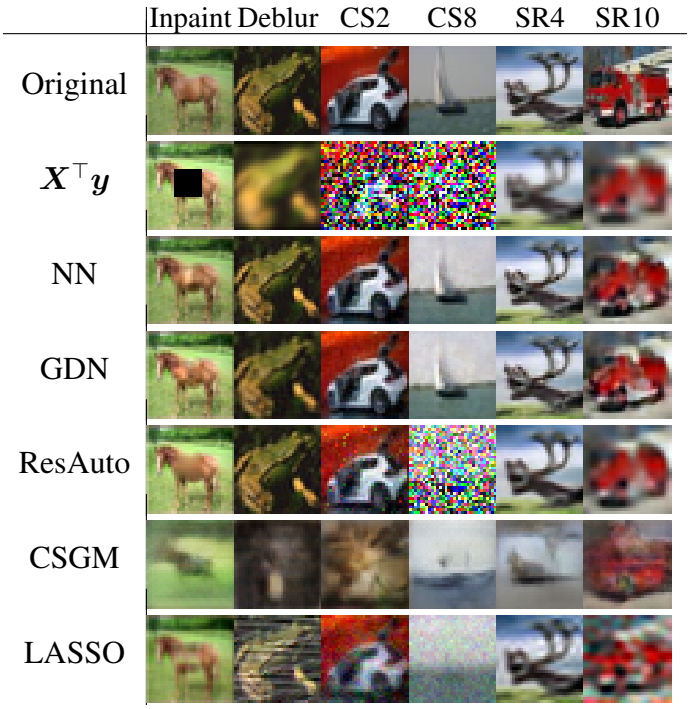


Figure 5: Visual demonstration of inverse problem solutions on the CIFAR-10 dataset. The Network Input row represents $\boldsymbol{X}^\top \boldsymbol{y}$, which is fed into both GDN and NN. Inpainting has a 10×10 inpainting region. Deblur has a Gaussian convolutional filter with $\sigma = 2.5$ and filter 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.

We observe that Neumann networks perform well across all baselines on all tested datasets. While the LASSO's performance is consistent across problems and data, we observe significant variability in the performance of the residual autoencoder. Recall the motivation for the residual autoencoder: the closer $\boldsymbol{X}^\top \boldsymbol{y}$ is to the ground truth $\boldsymbol{\beta}^\star$, the smaller 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 deblurring, 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 $\boldsymbol{\beta}^\star$.

Figures 7, 8, 9 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 $\boldsymbol{\beta}^\star - \widehat{\boldsymbol{\beta}}$ for $\boldsymbol{\beta}^\star$ 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. In all figures hereafter, the same formula is used for residuals.

5.4 Effect of Sample Size on Methods

A core hypothesis presented earlier was that incorporating information about the geometry imposed by the forward operator would have implications for the sample sizes required to achieve

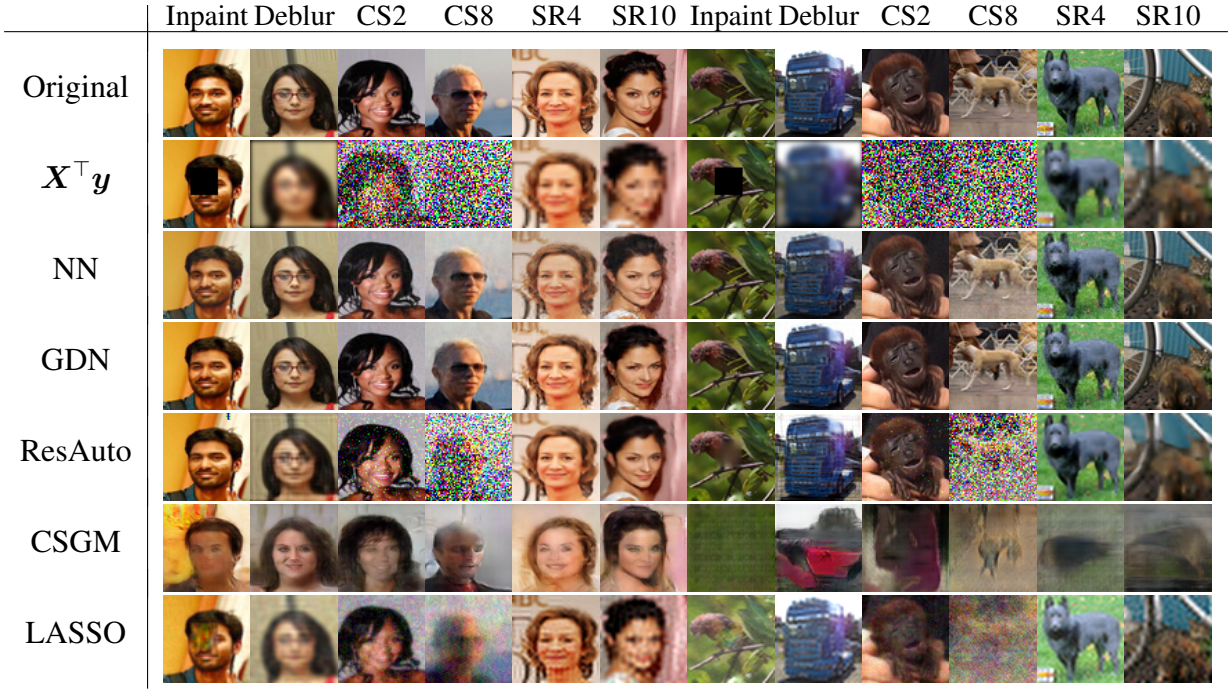


Figure 6: Visual demonstration of inverse problem solutions on the CelebA and STL10 dataset. The problems are identical to the CIFAR-10 case, as are the comparison methods.

particular error rates.

A coarse comparison of the presented learning-based methods at different sample sizes is provided in Figure 11a. 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 enjoys the largest performance increase 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 10.

Methods that do not incorporate the forward model, like ResAuto and CSGM, appear not to perform well in the low-sample regime, as discussed in Section 2.2. While ResAuto performs competitively at 30k and 50k iterations, there is little change between image qualities produced at these sample sizes, and 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 11b 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 from the 4-block case are included in 12

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 $(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1}$.

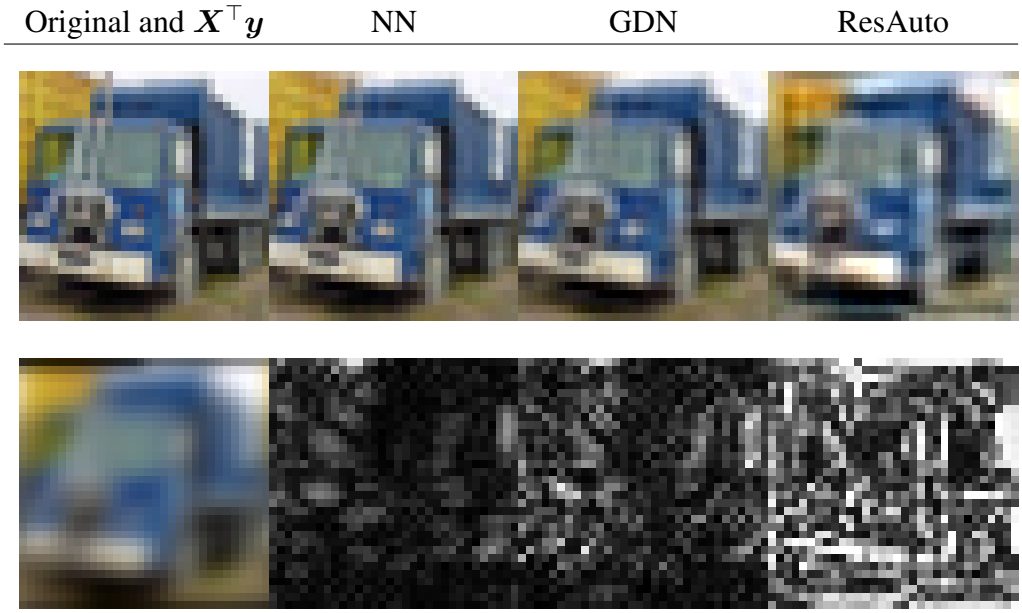


Figure 7: Deblurring reconstruction comparison on CIFAR-10. While the Neumann networks (NN) and gradient descent networks (GDN) perform well, the differences are most apparent in the residual images in the second row. NN produces sharper edges, and while GDN captures high-level features, it and the ResAuto method fail to reconstruct edges well. Residuals are formed by displaying the norm across color channels of the error at each pixel, scaled by a factor of 6.

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 Optimization Landscapes

The performance of the Neumann and gradient descent networks are very similar across a range of problems and datasets, but the Neumann network architecture appears to slightly outperform the gradient descent architecture consistently.

Both proposed architectures contain connections across blocks, but differ mainly in their *direction*. While GDN's addition of $\eta \boldsymbol{X}^\top \boldsymbol{y}$ at every stage is a forward connection, the NN approach of summing the outputs of every layer strongly resembles residual connections [32]. Recent work [40] 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 Figure 13 we illustrate the optimization landscapes using the method of [40]. Suppose that the fully-trained network has a set of kernels and weights which is vectorized as $\widehat{\boldsymbol{\theta}} \in \mathcal{R}^K$. Draw two independent standard Gaussian vectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2$ with dimension K, and normalize in the manner described in [40]. Then find the test and training set error at the points $\widehat{\boldsymbol{\theta}} + \tau(i\boldsymbol{v}_1 + j\boldsymbol{v}_2)$ for step

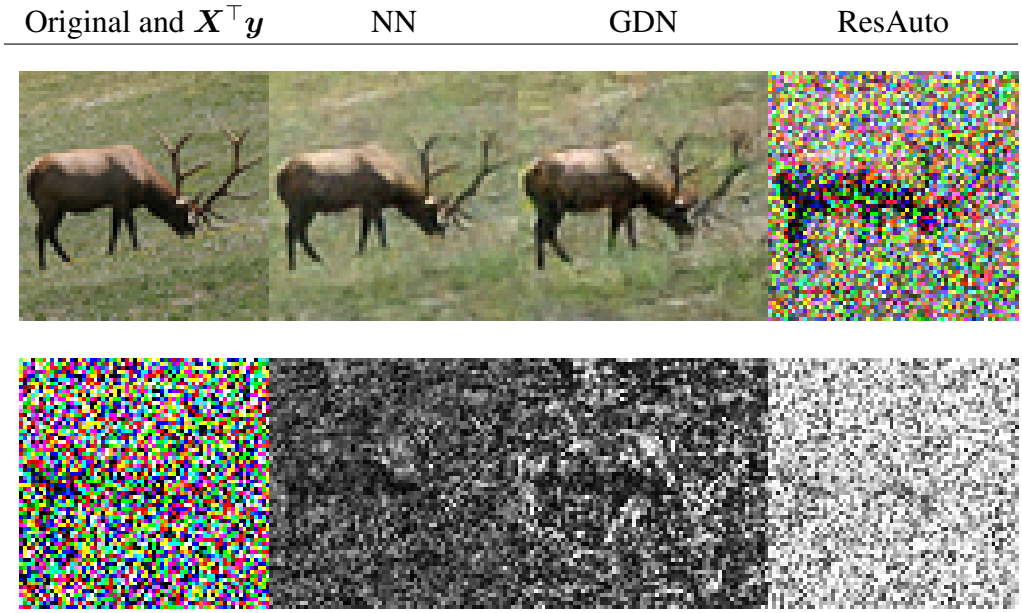


Figure 8: 8x compressed sensing reconstruction comparison on STL10. ResAuto fails to invert the compressed sensing problem adequately. The gradient descent network (GDN) reconstructs accurately but gives more artifacts. The Neumann network (NN) produces a good reconstruction at the cost of slight oversmoothing.

size $\tau > 0$ and integers i, j. The plots above are generated for $i, j \in \{-125, ..., 125\}$ and $\tau = 0.01$.

Figure 13 illustrates several attractive properties of the Neumann and gradient descent networks. Fortunately for both, local minima appear to be rare in the neighborhood of the trained minimum. While neither is convex, it is interesting to note that the Neumann network landscape has a much wider basin around the minimum 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 addition, experimental evidence and some theory indicates that wider local minima have better generalization properties [34]. This does not indicate that one architecture should perform better than another, but if both networks achieve similar training error, we may hope wider local minima would 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 derives from the Neumann series expansion for inverting linear operators and has several key features. First, Neumann networks naturally result in "skip connections" [32] that are absent from previously proposed network architectures. These skip connections appear to yield optimization landscapes that facilitate more efficient training.

Our theoretical analysis reveals that when the training data lie in a union of subspaces, then the optimal *oracle* estimator that has prior knowledge of both the subspaces in the union and

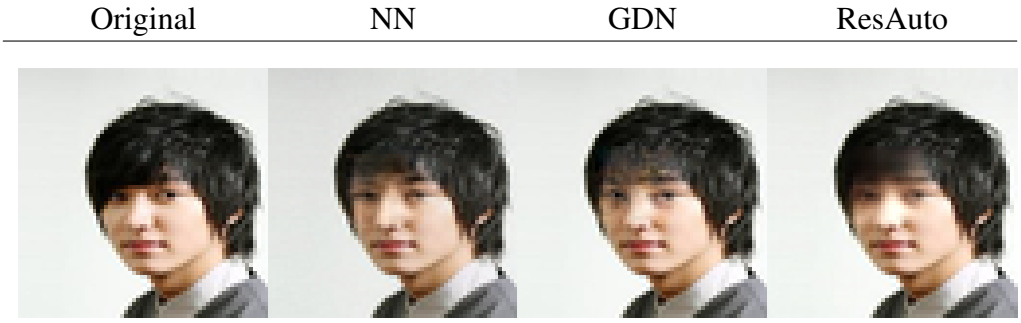


Figure 9: Inpainting reconstruction comparison on CelebA. While outside the inpainting region both gradient descent network (GDN) and Neumann network (NN) perform identically, in the inpainting region the differences are most clear in the eyes and hairline. NN generates a more realistic left eye, and generates two separated eyebrows, rather than a single linked eyebrow line. While a single eyebrow is more accurate in this case from a PSNR perspective, as it more closely resembles the hair covering that portion of the face in the original image, a split eyebrow is a more accurate model in general. Again, for ResAuto the inpainting region is much more smooth than the other two approaches.

the identity of the subspace to which true image belongs is easily represented by the proposed Neumann network architecture when ReLU or ELU [35] activations are used. Furthermore, we observe empirically on simulated union of subspaces data that the learned nonlinear components in the trained Neumann network well-approximate 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 [27, 2],

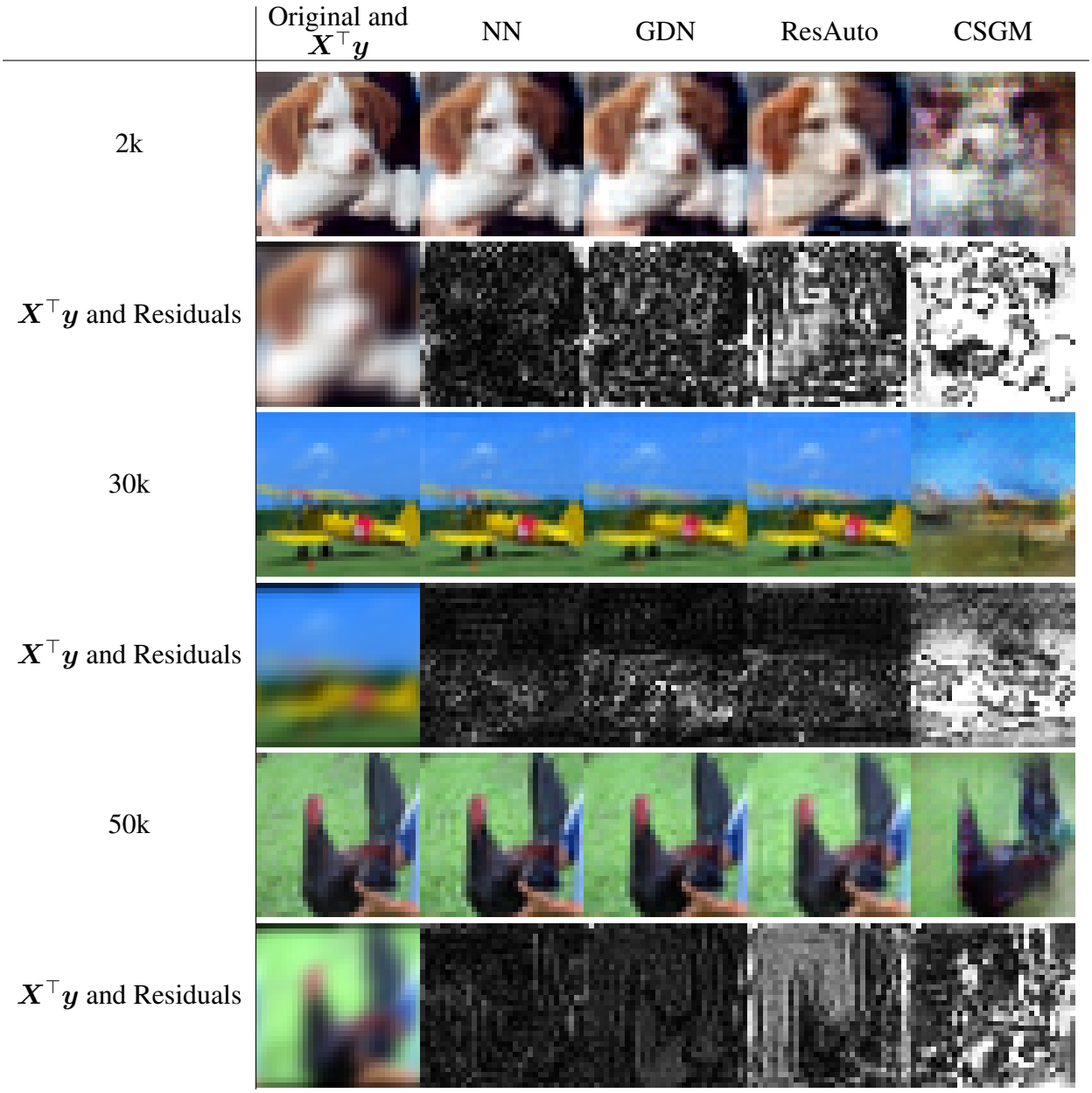
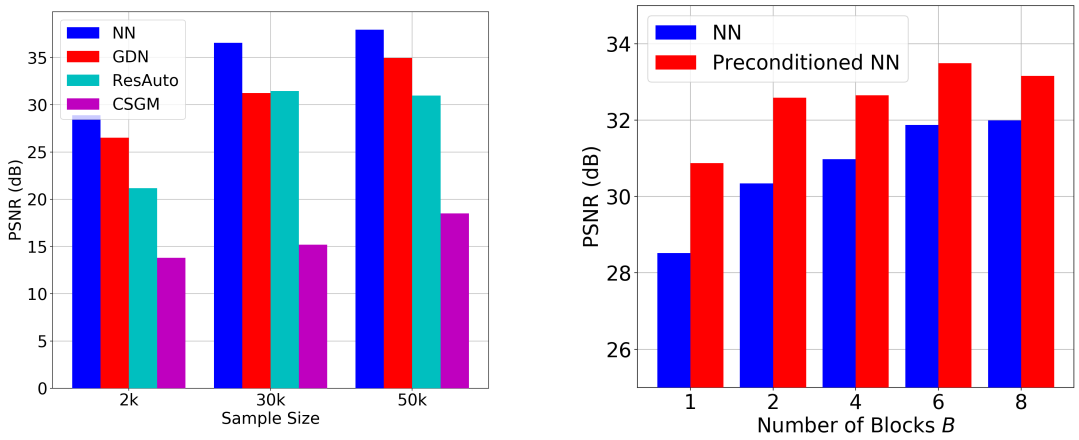


Figure 10: A qualitative comparison of the reconstructions produced for the deblurring problem at different training set sizes, along with the associated residual images.



(a) Sample Complexity

(b) Preconditioning

Figure 11: Performance comparisons. (a) Median PSNR of methods trained with different sample sizes of 2,000, 30,000, and 50,000. Neumann networks (NN) scale very well with training set size, with smaller marginal gains as training sizes increase. With very few samples the residual autoencoder (ResAuto) performs poorly, while enjoying good performance at a training size of 30k. 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 $\sigma = 5.0$.

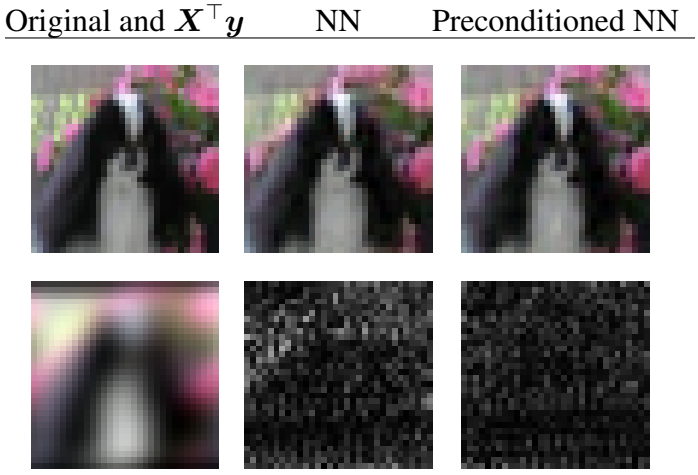
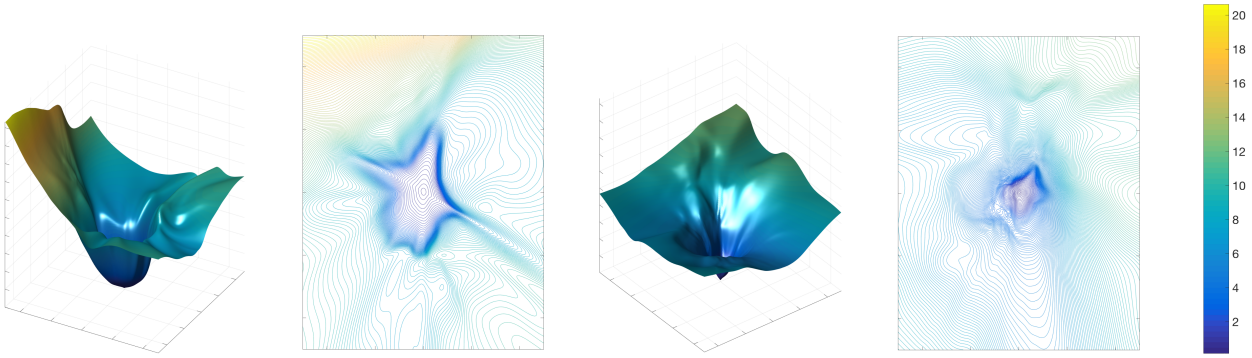


Figure 12: A qualitative comparison of the reconstructions produced for the deblurring problem using a Neumann network with preconditioning (Preconditioned NN) and without preconditioning (NN). The forward operator is a 5×5 Gaussian blur with $\sigma = 5$. While both are high quality, we observe highly structured errors without preconditioning, which are most clear in the residual image.



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

Figure 13: 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. Values displayed are the mean squared error over a set of training images. The Neumann network's landscape is more resilient to step size choices because of the wide basin around the minimizer, and is much steeper outside the basin of the minimizer, which are both more favorable to practical optimization by SGD. Figures use the CIFAR-10 dataset and deblurring inverse problem.

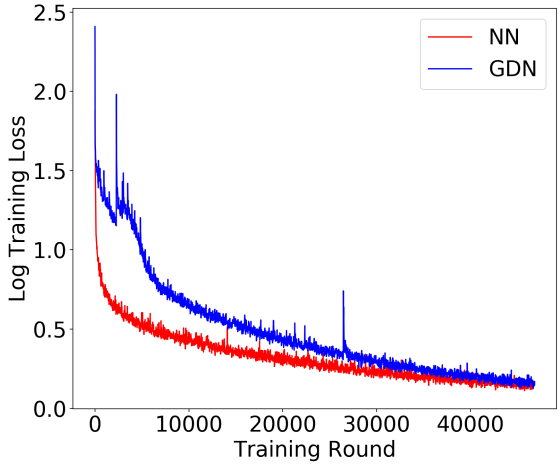


Figure 14: Log training loss for GDN and NN architectures for the deblurring inverse problem on the CIFAR-10 dataset. Loss recorded once per 10 training rounds. NN's loss is much smoother over the course of training, while GDN's loss is much more variable. In addition, we observe that while the final training loss value is approximately equal, the test losses are different in Table 1.

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" [12] 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

6.1 Proofs

For the following proofs 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. When $\boldsymbol{X}$ has orthonormal rows, as we assume in Lemma 1 and Theorem 1, we have $\boldsymbol{P_X} = \boldsymbol{X}^\top \boldsymbol{X}$ and $\boldsymbol{P_{X_\perp}} = \boldsymbol{I} - \boldsymbol{X}^\top \boldsymbol{X}$.

6.1.1 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

$$\tilde{\boldsymbol{\beta}}^{(j)} = (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R)\tilde{\boldsymbol{\beta}}^{(j-1)} \tag{39}$$

$$= (\boldsymbol{P_{X_\perp}} + (1-\eta)\boldsymbol{P_X} - \eta R)\tilde{\boldsymbol{\beta}}^{(j-1)} \tag{40}$$

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 $\boldsymbol{R}$ defined in (28) satisfies the desired error bounds. With this choice of $\boldsymbol{R}$ we have $\boldsymbol{P_X}\boldsymbol{R} = 0$, and an easy induction gives

$$\tilde{\boldsymbol{\beta}}^{(j)} = \eta(1-\eta)^j \boldsymbol{X}^\top \boldsymbol{y} - \eta \sum_{k=0}^{j-1} \boldsymbol{R}\tilde{\boldsymbol{\beta}}^{(k)} \text{ for all } j \geq 1. \tag{41}$$

Hence, we have

$$\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \sum_{j=0}^{B} \tilde{\boldsymbol{\beta}}^{(j)} = \sum_{j=0}^{B} \eta(1-\eta)^j \boldsymbol{X}^\top \boldsymbol{y} - \eta \sum_{j=1}^{B} \sum_{k=0}^{j-1} \boldsymbol{R}\tilde{\boldsymbol{\beta}}^{(k)} \tag{42}$$

$$= \sum_{j=0}^{B} \eta(1-\eta)^j \boldsymbol{X}^\top \boldsymbol{y} - \eta \sum_{j=0}^{B-1} (B-j)\boldsymbol{R}\tilde{\boldsymbol{\beta}}^{(k)} \tag{43}$$

Next, we show we can choose the constant c in (28) such that

$$\boldsymbol{P_{X_\perp}}\boldsymbol{U}(\boldsymbol{U}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{U})^{-1}\boldsymbol{U}^\top \boldsymbol{X}^\top \boldsymbol{y} = -\eta \sum_{j=0}^{B-1} (B-j)\boldsymbol{R}\tilde{\boldsymbol{\beta}}^{(j)}. \tag{44}$$

Observe that $\boldsymbol{R}^2\tilde{\boldsymbol{\beta}}^{(k)} = 0$ for all $k = 0, ..., j-1$, and so from (41) we have

$$\boldsymbol{R}\tilde{\boldsymbol{\beta}}^{(j)} = \eta(1-\eta)^j \boldsymbol{R}\boldsymbol{X}^\top \boldsymbol{y}. \tag{45}$$

This gives

$$-\eta \sum_{j=0}^{B-1} (B-j)\boldsymbol{R}\tilde{\boldsymbol{\beta}}^{(j)} = \left(-\eta^2 \sum_{j=0}^{B-1} (B-j)(1-\eta)^j\right) \boldsymbol{R}\boldsymbol{X}^\top \boldsymbol{y}. \tag{46}$$

Letting

$$c_{\eta,B} = \left(\eta^2 \sum_{j=0}^{B-1} (B-j)(1-\eta)^j\right)^{-1} \tag{47}$$

we obtain (44).

Therefore, altogether we have

$$\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \left(\eta \sum_{j=0}^{B} (1-\eta)^j\right) \boldsymbol{X}^\top \boldsymbol{y} + \boldsymbol{P}_{\boldsymbol{X}_\perp} \boldsymbol{U}(\boldsymbol{U}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{U})^{-1} \boldsymbol{U}^\top \boldsymbol{X}^\top \boldsymbol{y}. \tag{48}$$

Finally, using the fact that $\eta \sum_{j=0}^{B} (1-\eta)^j = 1 - (1-\eta)^{B+1}$ and $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^\star$ we have

$$\widehat{\boldsymbol{\beta}}(\boldsymbol{y}) = \boldsymbol{\beta}^\star - (1-\eta)^{B+1} \boldsymbol{P}_{\boldsymbol{X}} \boldsymbol{\beta}^* \tag{49}$$

which gives the desired error bound (29).

6.1.2 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 $\tilde{\boldsymbol{\beta}}^{(j)} = (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X} - \eta R^*)\tilde{\boldsymbol{\beta}}^{(j-1)}$ for $j = 1, ..., B$. The desired error bounds then follow by direct application of Lemma 1.

Similar to the expression (41) for $\tilde{\boldsymbol{\beta}}^{(j)}$ obtained in the proof of Lemma 1, an easy induction shows that

$$\tilde{\boldsymbol{\beta}}^{(j)} = \eta(1-\eta)^j \boldsymbol{X}^\top \boldsymbol{y} - \eta \sum_{i=0}^{j-1} R^*(\tilde{\boldsymbol{\beta}}^{(k)}) \tag{50}$$

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

$$\boldsymbol{P}_{\boldsymbol{X}} \tilde{\boldsymbol{\beta}}^{(j)} = \eta(1-\eta)^j \boldsymbol{P}_{\boldsymbol{X}} \boldsymbol{\beta}^\star \tag{51}$$

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,

$$\|(\boldsymbol{I} - \boldsymbol{X}\boldsymbol{U}_k(\boldsymbol{X}\boldsymbol{U}_k)^+)\boldsymbol{X}\boldsymbol{\beta}^\star\| < \|(\boldsymbol{I} - \boldsymbol{X}\boldsymbol{U}_\ell(\boldsymbol{X}\boldsymbol{U}_\ell)^+)\boldsymbol{X}\boldsymbol{\beta}^\star\| \tag{52}$$

for all $\ell \neq k$. Since $\boldsymbol{X}\boldsymbol{U}_k(\boldsymbol{X}\boldsymbol{U}_k)^+$ is projection onto span$(\boldsymbol{X}\boldsymbol{U}_k)$, and $\boldsymbol{X}\boldsymbol{\beta}^\star \in$ 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

$$\|(\boldsymbol{I} - \boldsymbol{X}\boldsymbol{U}_k(\boldsymbol{X}\boldsymbol{U}_k)^+)\boldsymbol{X}\boldsymbol{\beta}^\star\| = 0. \tag{53}$$

Furthermore, since by assumption $\boldsymbol{X}\boldsymbol{\beta}^\star \notin$ 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

$$\|(\boldsymbol{I} - \boldsymbol{X}\boldsymbol{U}_\ell(\boldsymbol{X}\boldsymbol{U}_\ell)^+)\boldsymbol{X}\boldsymbol{\beta}^\star\| > 0, \tag{54}$$

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 $\boldsymbol{\beta}^{(j)} = (\boldsymbol{I} - \eta\boldsymbol{X}^\top\boldsymbol{X} - \eta R)\boldsymbol{\beta}^{(j-1)} + \boldsymbol{\beta}^{(0)}$. Since $\boldsymbol{P}_{\boldsymbol{X}} R^*(\boldsymbol{\beta}) = 0$ for all $\boldsymbol{\beta} \in \mathbb{R}^p$, an easy induction shows that

$$\boldsymbol{P}_{\boldsymbol{X}}\boldsymbol{\beta}^{(j)} = \eta \sum_{i=0}^{j} (1-\eta)^i \boldsymbol{P}_{\boldsymbol{X}}\boldsymbol{\beta}^\star. \tag{55}$$

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.

6.2 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 (38) have the form

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{i=1}^{N} \frac{\partial \widehat{\boldsymbol{\beta}}(\boldsymbol{y}_i; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}^\top \left(\widehat{\boldsymbol{\beta}}(\boldsymbol{y}_i; \boldsymbol{\theta}) - \boldsymbol{\beta}_i \right) \tag{56}$$

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

$$\tilde{\boldsymbol{\beta}}^{(j+1)}(\boldsymbol{\theta}) = (\boldsymbol{I} - \eta\boldsymbol{X}^\top\boldsymbol{X})\tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta}) - \eta R(\tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta}); \boldsymbol{\theta}) \tag{57}$$

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

$$\frac{\partial \widehat{\boldsymbol{\beta}}(\boldsymbol{y}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{j=0}^{B} \frac{\partial \tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{58}$$

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 (57) and the chain rule to get:

$$\frac{\partial \tilde{\boldsymbol{\beta}}^{(j+1)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = (\boldsymbol{I} - \eta\boldsymbol{X}^\top\boldsymbol{X})\frac{\partial \tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} - \eta \left(\frac{\partial R(\tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta}); \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} + \frac{\partial R(\tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta}); \boldsymbol{\theta})}{\partial \boldsymbol{\beta}} \frac{\partial \tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right) \tag{59}$$

$$= \left(\boldsymbol{I} - \eta\boldsymbol{X}^\top\boldsymbol{X} - \eta \frac{\partial R(\tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta}); \boldsymbol{\theta})}{\partial \boldsymbol{\beta}} \right) \frac{\partial \tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} - \eta \frac{\partial R(\tilde{\boldsymbol{\beta}}^{(j)}(\boldsymbol{\theta}); \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{60}$$

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:

$$\partial_{\boldsymbol{\theta}} \tilde{\boldsymbol{\beta}}^{(j+1)} = \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j)}} \cdot \partial_{\boldsymbol{\theta}} \tilde{\boldsymbol{\beta}}^{(j)} - \eta\, \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j)}} \tag{61}$$

For example,

$$\partial_{\boldsymbol{\theta}} \tilde{\boldsymbol{\beta}}^{(1)} = -\eta\, \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(0)}} \tag{62}$$

and

$$\partial_{\boldsymbol{\theta}} \tilde{\boldsymbol{\beta}}^{(2)} = -\eta \left(\partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(0)}} + \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} \right) \tag{63}$$

and

$$\partial_{\boldsymbol{\theta}} \boldsymbol{\beta}_3 = -\eta \left(\partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(2)}} \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(0)}} + \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(2)}} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} + \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(2)}} \right) \tag{64}$$

and so in general

$$\partial_{\boldsymbol{\theta}} \boldsymbol{\beta}_k = -\eta \left(\sum_{j'=0}^{j-1} \left(\partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j-1)}} \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j-2)}} \cdot \cdots \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j'+1)}} \right) \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j')}} \right). \tag{65}$$

Therefore,

$$\partial_{\boldsymbol{\theta}} \widehat{\boldsymbol{\beta}} = -\eta \sum_{j=0}^{B} \left(\sum_{j'=0}^{j-1} \left(\partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j-1)}} \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j-2)}} \cdot \cdots \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j'+1)}} \right) \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(j')}} \right). \tag{66}$$

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

$$\boldsymbol{\beta}^{(j+1)}(\boldsymbol{\theta}) = (\boldsymbol{I} - \eta \boldsymbol{X}^\top \boldsymbol{X}) \boldsymbol{\beta}^{(j)}(\boldsymbol{\theta}) - \eta R(\boldsymbol{\beta}^{(j)}(\boldsymbol{\theta}); \boldsymbol{\theta}) + \boldsymbol{\beta}^{(0)} \tag{67}$$

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:

$$\partial_{\boldsymbol{\theta}} \boldsymbol{\beta}^{(j+1)} = \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(j)}} \cdot \partial_{\boldsymbol{\theta}} \boldsymbol{\beta}^{(j)} - \eta\, \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(j)}} \tag{68}$$

And so

$$\partial_{\boldsymbol{\theta}} \widehat{\boldsymbol{\beta}}' = -\eta \left(\sum_{j'=0}^{B-1} \left(\partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(j-1)}} \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(j-2)}} \cdot \cdots \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(j'+1)}} \right) \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(j')}} \right). \tag{69}$$

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

$$\begin{aligned} \partial_{\boldsymbol{\theta}} \widehat{\boldsymbol{\beta}} = -\eta \Big(& \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(0)}} + \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} + \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(2)}} \\ & + \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(0)}} + \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}_2} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} \\ & + \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(2)}} \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(1)}} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\tilde{\boldsymbol{\beta}}^{(0)}} \Big). \end{aligned} \tag{70}$$

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

$$\partial_{\boldsymbol{\theta}} \widehat{\boldsymbol{\beta}}' = -\eta \left(\partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(2)}} + \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(2)}} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(1)}} + \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(2)}} \cdot \partial_{\boldsymbol{\beta}} F|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(1)}} \cdot \partial_{\boldsymbol{\theta}} R|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(0)}} \right) \tag{71}$$

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

Implementation details are described here. High-level choices are described in Section 5.2.

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 [11, 10]. 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 [15].

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 $32, 64, 128, 256, 512, 512, 512, 256, 128, 64, 3$.

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.

References

[1] J. Adler and O. Öktem. Learned primal-dual reconstruction. *IEEE Transactions on Medical Imaging*, 37(6):1322–1332, 2018.

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

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

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

[5] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[6] K. Bertin, C. Lacour, and V. Rivoirard. Adaptive pointwise estimation of conditional density function. In *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, volume 52, pages 939–980. Institut Henri Poincaré, 2016.

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

[8] T. Blumensath. Sampling and reconstructing signals from a union of linear subspaces. *IEEE Transactions on Information Theory*, 57(7):4660–4671, 2011.

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

[10] 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)*, pages 5889–5898, 2017.

[11] 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*, 36(12):2524–2535, 2017.

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

[13] 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*, 39(6):1256–1272, 2017.

[14] I. Y. Chun and J. A. Fessler. Deep BCD-net using identical encoding-decoding CNN structures for iterative image recovery. *arXiv preprint arXiv:1802.07129*, 2018.

[15] 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.

[16] 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)*, pages 215–223, 2011.

[17] A. Danielyan, V. Katkovnik, and K. Egiazarian. BM3D frames and variational image deblurring. *IEEE Transactions on Image Processing*, 21(4):1715–1728, 2012.

[18] B. Delyon and A. Juditsky. On minimax wavelet estimators. *Applied and Computational Harmonic Analysis*, 3(3):215–228, 1996.

[19] R. A. DeVore and G. G. Lorentz. *Constructive approximation*, volume 303. Springer Science & Business Media, 1993.

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

[21] 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)*, pages 184–199. Springer, 2014.

[22] 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*, 38(2):295–307, 2016.

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

[24] S. Efromovich. Conditional density estimation in a regression setting. *The Annals of Statistics*, 35(6):2504–2535, 2007.

[25] E. Elhamifar and R. Vidal. Sparse subspace clustering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2790–2797, 2009.

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

[27] L. Gabet. The theoretical foundation of the Adomian method. *Computers & Mathematics with Applications*, 27(12):41–52, 1994.

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

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

[30] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *International Conference on Machine Learning (ICML)*, pages 399–406, 2010.

[31] 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*, 37(6):1440–1453, 2018.

[32] 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)*, pages 770–778, 2016.

[33] 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)*, volume 1, page 3, 2017.

[34] 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.

[35] 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)*, pages 1646–1654, 2016.

[36] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

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

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

[39] 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)*, pages 4681–4690, 2017.

[40] 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)*, pages 6389–6399, 2018.

[41] 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*, 35(1):171–184, 2013.

[42] 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.

[43] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pages 689–696. ACM, 2009.

[44] 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)*, pages 2802–2810, 2016.

[45] 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)*, pages 1–5, 2017.

[46] 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.

[47] 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)*, pages 1799–1808, 2017.

[48] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[49] 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*, pages 2536–2544, 2016.

[50] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[51] 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*, pages 1859–1863, 2017.

[52] S. Ravishankar, A. Lahiri, C. Blocker, and J. A. Fessler. Deep dictionary-transform learning for image reconstruction. In *IEEE International Symposium on Biomedical Imaging*, pages 1208–1212, 2018.

[53] 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)*, pages 234–241. Springer, 2015.

[54] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992.

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

[56] 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)*, pages 1458–1466, 2011.

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

[58] 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*, pages 623–635. Springer, 2017.

[59] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

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

[61] 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*, 22(3):332–350, 2003.

[62] 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)*, pages 1790–1798, 2014.

[63] 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*, 21(5):2481–2499, 2012.

[64] 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*, volume 2, 2017.