University of Minnesota
School of Mathematics

Solving Ill-Posed Inverse Problems with Computerized Reconstruction Approaches: A Preliminary Exploration

Sai Suchir Tyada

Supervisor: Ru-Yu Lai

A thesis submitted in partial fulfillment of the requirements
of
the University of Minnesota for the degree of
Bachelor of Arts in *Mathematics*

Abstract

We conduct a preliminary exploration into computerized reconstruction approaches to solving ill-posed inverse problems with not necessarily linear forward operators. We utilize the Learned Reconstruction, Learned Primal-Dual reconstruction, and the Stochastic Primal-Dual Hybrid Gradient-based reconstruction to compare and contrast the different approaches and make note of any improvements made to performance metrics. The Learned Primal-Dual implementation iterates with a primal and a dual intermittently updating the variables, all while leveraging proximal operator-like methods to reach convergence on an optimal solution. The Learned Reconstruction approach utilizes an iterative gradient-based algorithm that learns the parameters, Θ, using a convolutional neural network (CNN) incorporating prior iterations along the way. The Stochastic Primal-Dual Hybrid Gradient (SPDHG) approach utilizes the SPDHG algorithm to iterate over random subsets of data as opposed to the entire data. Naturally, as an extension of the aforementioned Learned Primal-Dual approach, the rest of the primal-dual iterate process remains much the same. We run our iterative approaches (and some variants) on a non-linear medical tomographic inversion problem using simulated data from the Shepp-Logan phantom to create results. Namely, we notice Peak Signal-to-Noise Ratios (PSNR), measured in dB, close to the mid-40s range. Whereas, past records of Filtered Back-Projection alone were much lower.

Keywords: tomography, deep learning, gradient descent, inverse problems, primal-dual hybrid gradient

Contents

1 Introduction

Inverse problems are a concept that is not new to the world of mathematics with the most generalized definition being: a problem where we want to reconstruct the original state by only using the partially available information in the end state. In fancier terms, inverse problems are problems where one seeks to reconstruct parameters characterizing the system under investigation from indirect observation.

Conceptually, this can be written as

$$A(cause) = effect,$$

and mathematically it can be expressed as

$$m = Af + \varepsilon,$$

where f is a piecewise continuous function defined on a subset of $\mathbb{R}^d$ and $\mathbf{m} \in \mathbb{R}^k$ is a vector of numbers given by a measurement device [1]. Additionally, the ε is used to model noise and error measurements with A being a linear operator in this particular case.

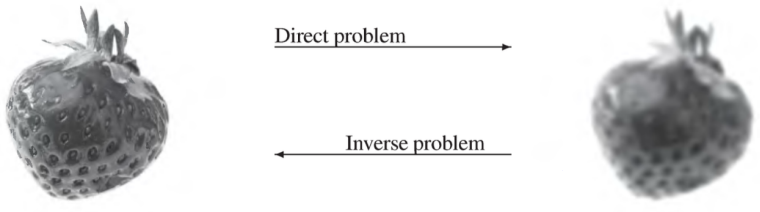

Figure 1: Here the cause is the sharp image and the effect is the blurred image.

To specify this mathematical expression into something that fits our specific use case we have

$$g = T(f_{true}) + \delta g.$$

We now have expressed our inverse problem as a reconstruction of the signal $f_{true} \in X$ from the data $g \in Y$ where X and Y are topological

vector spaces. The forward operator $T : X \to Y$ then models the noise-free mapping from signals in X to data in Y. The noise component of the data is represented by $\delta g \in Y$, a single realization of a Y-valued random variable. To put it in layman's terms, we want to control δg (the noise) while estimating f_{true} (signal) as closely as possible using the g (data) available.

1.1 Well-Posedness vs. the Ill-Posedness of Inverse Problems

Along with the concept of inverse problems, comes the notion of a 'well-posed inverse problem'. As with any case where a specific structure is required to attain the ideal case, there must be a structure that opposes the ideal case, which in our case we call the 'ill-posed inverse problem'. Well-posed inverse problems are defined simply by 3 key properties:

- H_1: Existence. There should be at least one solution.
- H_2: Uniqueness. There should be at most one solution.
- H_3: Stability. The solution must depend continuously on data.

By this, we note that for an inverse problem to be ill-posed, it must break one of these 3 properties. Ill-posed problems are mostly engaging with issues where there are tasks that are extremely sensitive to measurements. This results in an inverse problem where the reconstruction of the original state requires more complex and nuanced approaches. We need not just mathematically to solve the inverse problem but also solve it while controlling the noise in the already extremely sensitive data. All this while also paying attention to being efficient with time complexity and computational resources.

1.2 Problem Statement

In this thesis, we specifically set our attention to the reconstruction of X-ray tomographic imaging, see section 1.4 for details. Namely, we were motivated by the partially learned gradient descent-based approach (learned reconstruction) discussed in [2]. Wherein, there was discussion

made on the sub-optimal nature of the iterative scheme in the face of non-differentiable objective functions. Thus, we explore the improvements to the iterative scheme utilizing the Primal-Dual Hybrid Gradient (PDHG) algorithm (learned primal-dual) and the Stochastic Primal-Dual Hybrid Gradient algorithm, to be able to tackle these non-differentiable objective functions.

1.3 Machine Learning and Neural Networks

Machine learning, particularly neural networks, has revolutionized the way inverse problems are approached. Neural networks are excellent at learning complex mappings from data, making them well-suited for approximating solutions to inverse problems. Convolutional neural networks (CNNs) have been widely used for image reconstruction tasks due to their ability to capture spatial hierarchies and patterns. There are primarily two styles of learning in CNNs: supervised and unsupervised.

1.3.1 Convolutional Neural Networks

Neural networks are impressive in their abilities to pick up patterns and apply those "learned" patterns to different problems ranging from movie recommendation systems to predictive forecasting and beyond. We use Convolutional Neural Networks (CNNs) to pick up patterns in our parameters for the loss functions of each of the respective reconstructions. By minimizing loss we can get closer to our ground truth phantom.

So what is a CNN? Is it magic? a black-box? The answer is simple but at the same time not so simple. CNNs are neural networks that try to mimic the ability of the brain to pick up patterns specifically when it comes to structured data like images or videos etc. Let's go through the steps together. First, we have the forward pass where we convert our image into a feature map by passing it through our convolutional layers. Here we run a linear function for each node. We define x_i as input and z as the solution to the node's linear function, w_i as the weights and b as the biases. Tying it up we have something like this,

$$z = \sum_i x_i w_i + b.$$

After we find our z, we run the activation function (mostly ReLU in our

case with the exception of learned primal-dual where we use Parametric ReLU).

$$\text{ReLU}(z) = \begin{cases} z, & z \geq 0 \\ 0, & \text{otherwise,} \end{cases}$$

$$\text{Parametric ReLU}(z) = \begin{cases} z, & z > 0 \\ \alpha z, & z \leq 0 \end{cases}$$

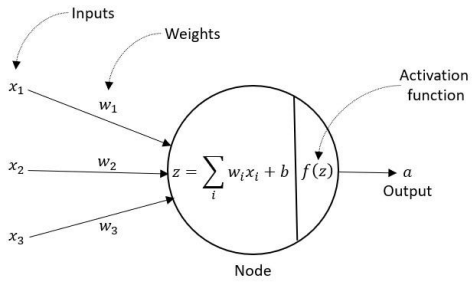

Figure 2: Presented is a visual of how the forward pass works for a single node within a single layer.

Once we go through all the layers repeating the linear function and the activation function, we are presented with the output. This result is then used in the loss function to calculate the error between the predicted output and the ground truth. For the purposes of this thesis, the Mean Squared Error is chosen for most of our loss reconstruction functions:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (f_i - f_{true})^2.$$

The final steps are related to the gradients and partial derivatives. After the loss is calculated, through a process called backpropagation (different from backprojection) the weights for each layer is updated. Backpropagation is a process of repetitive calculation of partial derivatives and addition to the weights. This logic is left for the Tensorflow library to figure out.

1.3.2 Unsupervised Learning

We first start by giving an introduction to unsupervised learning by giving the analogy of a student who learns by reading random notes to pick up patterns without knowing what exam they are going to take.

Unsupervised learning involves the use of unlabeled data wherein there is no distinction on whether or not something is right or wrong. The idea behind training like this is to allow a model or network to naturally pick up patterns within the data without external human guidance. This style of learning is especially useful when there is a complex problem with patterns not yet detected. Unsupervised learning is great at tasks such as clustering (e.g. k-means clustering), association rule mining, and dimensionality reduction (e.g. principal component analysis). All tasks that are common in the world of machine learning.

Unfortunately, for the task of reconstructing medical tomographic imaging, a certain level of preciseness and accuracy is expected. In real-life medical settings, models do not have the leisure to allow such learning lest it lend itself to more 'creative' reconstructions causing misdiagnoses for the patients, malpractice lawsuits for the care provider, and so forth.

1.3.3 Supervised Learning

To give an analogy for supervised learning, we liken it to a student who is learning how to solve quadratic equations, and their teacher gives them a set of examples with both the equations and their solutions provided. Using these examples to understand the process (e.g., factoring, completing the square, etc.), when given a new equation without the solution, the student applies what they've learned to solve it.

Supervised learning is a better option for our task at hand being that we can train the neural network for exactly what we want (parameters) then we can validate the trained model by presenting a similar style of problem in the validation set. This allows for a network that helps solve the reconstruction task more precisely.

In this thesis, we integrate machine learning with traditional optimization techniques to tackle the X-ray tomography problem. The learned reconstruction approach combines the strengths of CNNs and iterative optimization, enabling the network to learn from limited data while lever-

aging the physical model encoded in the forward operator T. This hybrid approach offers a promising avenue for overcoming the challenges posed by ill-posed inverse problems.

1.4 X-ray Tomographic Imaging

X-ray tomographic imaging is a powerful tool for visualizing the internal structure of objects in a non-invasive manner. This technique is widely used in medical applications, such as computed tomography (CT) scans, and in industrial settings for quality control and material analysis. The technique uses X-rays to create detailed cross-sectional images of objects or the human body. Mathematically, the problem involves reconstructing a function f_{true} from a series of projections g, which are obtained by applying the radon transform.

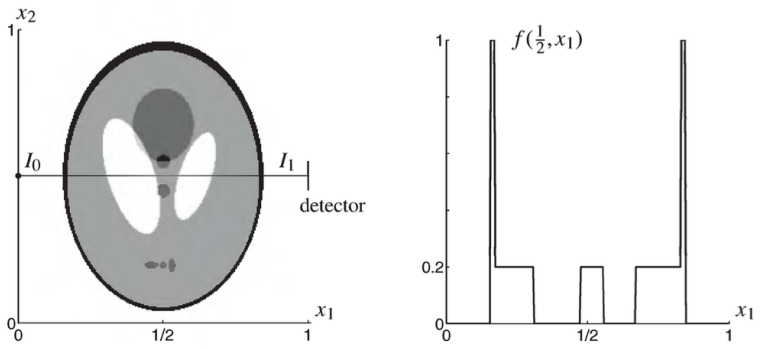

Figure 3: X-ray-based measurement. Left: an X-ray traveling through the Shepp-Logan phantom. Note that higher attenuation is shown here as darker shades of gray and lower attenuation as lighter shades. Right: plot of the attenuation coefficient along the path of the X-ray. See [1]

1.4.1 The Radon Transform

The radon transform is a widely used formula for reconstruction in medical tomography. The work in this thesis makes use of an analogue of the radon transform known as the ray transform. The idea remains much

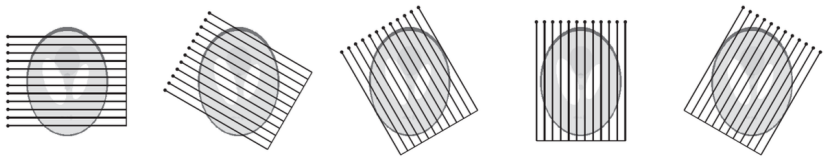

Figure 4: Illustration of how tomographic measurements are made. See [1].

the same, we shoot X-rays with some beam intensity I_0 from multiple angles onto an object we want scanned. As the beam hits the object, some of the lower-intensity photons within the X-ray are absorbed by the object in a manner proportional to the attenuation factor of the object in a process called beam hardening. Think of it as asking how many X-ray photons can this layer in the object absorb.

Figure (3) showcases how attenuated radon transform works. We can see the Shepp-Logan phantom with various attenuation values where the intensity of the parallel beams is affected by the object's X-ray photon absorption properties. Figure (4) shows the change in angle θ as the X-rays move around the phantom. The result is the plot on the right-hand side of Figure (3). In this plot, we see the attenuation coefficients for each corresponding layer in the phantom. The simulated skull portion has the highest attenuation compared to the simulated brain matter and muscles etc. After we run the same X-ray scanning across all the angles θ, we are able to generate a sinogram. From this sinogram, we are able to reconstruct our image.

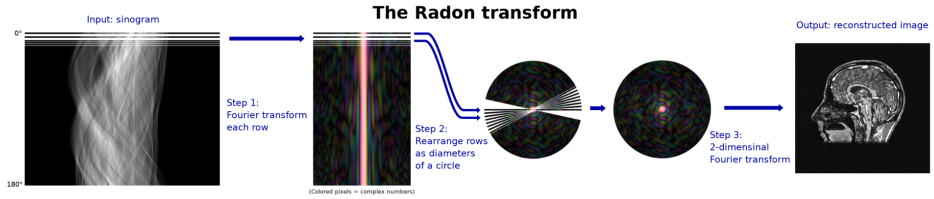

Figure 5: Illustration of the Radon transform using a sinogram and the Fourier transform to reconstruct a brain scan. Image courtesy of https://en.wikipedia.org/wiki/Radon_transform.

Mathematically, we are shooting X-rays with a starting intensity of I_{0,x_2} and measuring the ending intensity I_{1,x_2} after it has passed through the object. We make note that as the X-rays move in layers through the object, we are able to understand that each layer conceptually is a "slice" or cross-section of the object. See Figure (3) for visuals. We then take the line integral of the "line" formed by the X-ray and the cross-section of the object. We can model the absorption of the photons using a non-negative attenuation coefficient function $f(x_1, x_2)$. That is to say that we have $I_{x_1,x_2} = f(x_1, x_2)$. Using this model, we can covert to our line integral like so,

$$\begin{aligned} I_{x_1,x_2} &= -f(x_1, x_2) \\ I(x_i, x_2) &= -f(x_i, x_2). \end{aligned}$$

We want to get the total measured intensity at the end of the object slice. For the discrete case, we hold the x_2 stagnant at some value and sum all of the x_1 values. See figures (3) and (4) for visuals.

$$I(x_1) = \sum_{i=0}^{1} -f(x_i, x_2)$$

We make the obvious observation that for the continuous case, the summation becomes a line integral resulting in a final intensity with all the intensity absorption accounted for a single line.

$$\frac{dI(x_1)}{I(x_1)} = -f(x_1, x_j)dx_1 \tag{1}$$

$$\int_0^1 f(x_1, x_j)dx_1 = -\int_0^1 \frac{I'(x_1)}{I(x_1)}dx_1. \tag{2}$$

But, we are not just doing this for a single line, we want to do this same type of calculation multiple times across multiple angles. We define our angle as $\theta \in \mathbb{R}$ measured in radians. We also give that θ is a vector denoted by the x_1 unit vector,

$$\vec{\theta} := \begin{bmatrix} cos\theta \\ sin\theta \end{bmatrix} \in \mathbb{R}^2.$$

Along with this angular parameter θ, we define a linear parameter $s \in \mathbb{R}$. Also, we make use of the one-dimensional Lebesgue measure along the line defined by $\{X \in \mathbb{R}^2 : x \cdot \theta = s\}$. Combining all of the ideas together with the line integrals we defined before on equation (2). We can generate the Radon transform general formula for the continuous state as,

$$\mathfrak{R}f(s,\theta) = \int_{x\cdot\vec{\theta}=s} f(x)dx^{\perp}.$$

However, due to limited-angle sampling, noise, and other imperfections, this reconstruction problem is highly ill-posed. Traditional methods, such as filtered back projection (FBP), rely on analytical solutions to the radon transform and often fail to produce accurate reconstructions in the presence of noise or incomplete data [1]. To address these challenges, modern approaches incorporate regularization techniques and data-driven methods, which we will explore in this thesis.

1.5 Learned Reconstruction

The learned reconstruction approach introduced in [2] applies the logic from classical gradient descent to an iterative scheme wherein parameters for a neural network are learned. Learned reconstruction uses neural networks to directly learn how to reconstruct images from noisy or incomplete data. The idea is to minimize the difference between the reconstructed image and the original phantom.

1.5.1 Gradient Descent Formula

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta).$$

The concept of gradient descent can be elusively simple all while being confusing to visualize when applied to specific cases with its mathematical representation alone scaring off the uninitiated.

Gradient descent is a type of hill-climbing algorithm commonly used to optimize for a minimization function. The algorithm takes incremental steps following the gradient (slope) of the function to ideally find the

global minimum point. It is efficient, flexible, intuitive, scalable, visualizable, and so forth. But most importantly for our situation, it can handle a wide variety of problems.

1.5.2 Learned Reconstruction

The partially learned approach (learned reconstruction) mentioned in [2] used a gradient descent-based iterative scheme to allow a convolutional neural network (CNN) to train itself on the limited data available and then use the trained parameters to tackle the task of solving the ill-posed inverse problems. It accomplished this by estimating the reconstruction of the X-ray tomographic imaging through the minimization of the loss function. The gradient descent formula used in this case looked something like this:

$$w_i = w_{i-1} - \eta \nabla w_i. \tag{3}$$

To give context, we are updating our weights w_i using the gradient calculation while also factoring in the amount of influence based on the learning rate η.

In the case of the learned reconstruction approach, we choose an error function that measures the difference between the reconstructed image and the ground truth phantom. Our main goal is to estimate the parameters $\mathbf{\Theta} \in Z$, where Z is an appropriate parameter space while minimizing the loss function $\mathbf{L}(\mathbf{\Theta})$.

Due to the nature of ill-posed inverse problems, small changes in input may vary the output by a very large scale. To minimize this variance and to allow the network models a chance to learn the patterns, we regularize our objective function. We define $\mathcal{S} : X \to \mathbb{R}$, a regularization functional with prior information. We also define λ as a regularization parameter quantifying how much of an effect the regularization function has on the final minimization problem. Resulting in

$$\min_{f \in x} [\mathcal{L}(\mathcal{T}(f), g) + \lambda \mathcal{S}(f)] \text{ for a fixed } \lambda \geq 0.$$

Minimizing this equation will allow us to reconstruct our signal properly. We pose the problem then as this: f

$$\arg\min_{f \in X} E(f) \approx \arg\min_{f \in X} [\mathcal{L}(\mathcal{T}(f), g) + \lambda \mathcal{S}(f)]. \tag{4}$$

As we generalize this approach to teach ourselves the parameters $\boldsymbol{\Theta}$, we convert the equation into a format that coincides with the gradient descent equation on (3). By continuously calculating our loss functional and having that value guide our gradient descent, we get something like this:

$$f_i := f_{i-1} - \eta(\nabla[\mathcal{L}(\mathcal{T}(.), g)](f_{i-1}) + \lambda[\nabla\mathcal{S}](f_{i-1})).$$

Using (4), we can reword this equation in terms of the error functional as:

$$f_{j+1} := f_j - \eta[\nabla E](f_j).$$

In this iterative process, the gradients provide the necessary direction for updating the weights, shifting the reconstruction in each step. With each iteration, the model's parameters are adjusted, gradually improving the accuracy. The updates continue until the error functional stabilizes and the step size for the gradient is below too small to notice, or a threshold for a maximum number of iterations is met.

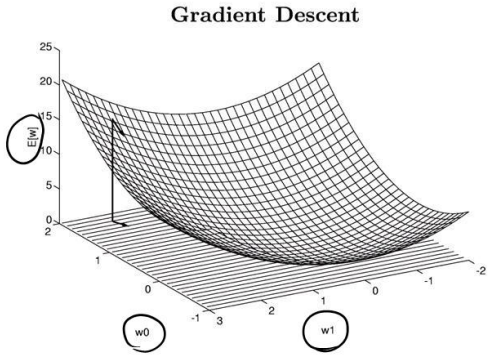

Figure 6: Illustration of gradient descent with weights and error functional. Image courtesy of CSEP546: Machine Learning - University of Washington

1.6 Learned Primal-Dual Reconstruction

The learned primal-dual reconstruction introduced in [3] is another tomographic reconstruction method. The approach is a logical cross-section

between the Primal-Dual Hybrid Gradient (PDHG) algorithm introduced in [4] and the aforementioned learned reconstruction approach. The researchers of [2] noted that while learned reconstruction was powerful there is an inherent inability for gradient descent methods to handle non-differentiable objective functions. Logically this makes sense as the idea of a gradient descent is based on taking small incremental steps towards a global minimum. The learned primal-dual reconstruction takes into account non-differentiable objective functions and replaces the gradient step with a parameterized operator to essentially *approximate* the steps artificially towards a global minimum.

1.6.1 Primal-Dual Hybrid Gradient Algorithm

The PDHG algorithm [4] while sounding daunting can be boiled down simplistically as a systematic guessing and checking. The primal variable is in charge of making the guess, while the dual variable is in charge of checking the guess and seeing if it fits into the big picture and gets us closer to the end goal. This back-and-forth allows for efficient convergence. This efficiency comes from the ability of the algorithm to make a guess without a differentiable function and for the dual to push the guess in the right direction.

While the applications of the PDHG algorithm (also known as the Chambolle-Pock algorithm) are many [5, 6, 4, 7, 8], we use the algorithm for the task of image reconstruction. The learned primal-dual reconstruction applies the logic from the PDHG algorithm by replacing the proximal operators with parameterized operators where the parameters are learned from the training data.

The PDHG algorithm is useful for solving saddle-point problems of the form:

$$[x^*, y^*] = \min_x \max_y \{r(x) + \langle Ax, y\rangle - f_b^*(y)\},$$

where $f_b^*(y) := \sup\{\langle h, y\rangle - f_b(h)\}$ is the Fenchel conjugate of the convex function $f_b(.)$, r is also a convex function, and A is a forward measurement operator.
The algorithm iteratively updates the primal variable x and the dual variable y as follows:

Primal Update:

$$x_{k+1} = \text{prox}_{\tau r}(x_k - \tau A^T y_k)$$

Dual Update:

$$y_{k+1} = \text{prox}_{\sigma f_b^*}(y_k + \sigma A\overline{x}_{k+1})$$

where τ and σ are step sizes, and $\text{prox}_h(z)$ denotes the proximal operator of function h at point z.

We can reformulate the function by considering a traditional form of the composite optimization function for image reconstruction:

$$x^* \in \arg\min_{x \in \mathbb{R}^d} f_b(Ax) + r(x)$$

The reformulated objective function can be appropriately solved by the PDHG algorithm in quite an efficient manner. With this understanding of the PDHG algorithm we have that the algorithm obeys the following updating rule:

Algorithm 1 Primal-Dual Hybrid Gradient (PDHG)

1: **Initialize** $x_0, \overline{x}_0 \in \mathbb{R}^d y_0 \in \mathbb{R}^p$
2: **for** $k = 0, 1, 2, ..., K$ **do**
3: $\quad y_{k+1} = \text{prox}_{\sigma f_b^*}(y_k + \sigma A\overline{x}_k)$
4: $\quad x_{k+1} = \text{prox}_{\tau r}(x_k - \tau A^T y_{k+1})$
5: $\quad \overline{x}_{k+1} = x_{k+1} + \beta(x_{k+1} - x_k)$
6: **end for**

To summarize, the PDHG algorithm works by taking turns to adjust two sets of variables: primal x and dual y. It keeps updating these variables until it finds the best solution. Often, it is better to change the original problem into a special form called the primal-dual form, especially when the problem is tricky or has lots of sharp corners (non-smooth) or when the rate of change is very high (large Lipschitz constant). Hence we have the learned primal-dual approach laid out by [3].

1.6.2 Learned Primal-Dual Reconstruction

As mentioned, the learned primal-dual reconstruction method takes inspiration from the PDHG algorithm and applies it to the learning of

Algorithm 2 Learned Primal-Dual

Initialize $f_0 \in X^{N_{primal}}, h_0 \in U^{N_{dual}}$
for $i = 1, \ldots, I$ **do**
    $h_i \leftarrow \Gamma_{\theta_i^d}(h_{i-1}, \mathcal{K}(f_{i-1}^{(2)}), g)$
    $f_i \leftarrow \Lambda_{\theta_i^p}(f_{i-1}, [\partial\mathcal{K}(f_{i-1}^{(1)})] * (h_i^{(1)}))$
end for
return $f_I^{(1)}$

parameters for a neural network through the replacement of the proximal operator with a parameterized operator. The final algorithm we utilized stems from [3] and is as follows:

To understand algorithm 2, we will break down the terms. The f terms are related to the primal variable and the h terms are related to the dual variable. Compared to the PDHG algorithm, the primal proximal has been replaced by a learned proximal, Γ_{θ^d} and the dual proximal by a learned proximal Λ_{θ^p}. We also have that $\mathcal{K} : X \to U$ is a (possibly non-linear) operator, U is a Hilbert space, and $[\partial\mathcal{K}(f_i)] : U \to X$ is the adjoint of the (Fréchet) derivative of $\mathcal{K}$ in point f_i [3].

We use the primal and dual variables to go back and forth for a set number of iterations I while updating the primal with information from the dual and dual with the primal exactly the same as the original PDHG algorithm. Finally, after the iterations are complete, we return the primal variable $f_I^{(1)}$ which will contain the solution guess or, in the final iteration, our reconstruction attempt.

In this thesis, we compare the learned primal-dual approach to solve the X-ray tomography reconstruction problem. The approach's ability to self-guide itself on incomplete data demonstrates measurable improvement compared to the learned reconstruction approach.

1.7 Stochastic Primal-Dual Hybrid Gradient Algorithm

The Stochastic Primal-Dual Hybrid Gradient (SPDHG) algorithm is introduced as an extended version of the PDHG algorithm. By processing only small random chunks at a time, SPDHG stands out in its ability to handle larger datasets [9].

The idea behind SPDHG is that, while PDHG is an impressive algorithm in its own right, the time to compute a solution can be significantly affected by the need to sample from the entire dataset, especially for very large datasets. SPDHG addresses this issue by randomly selecting a small subset (or block) of the data at each iteration, often without replacement, and performing computations on this subset. While this introduces stochastic noise into the process, it substantially reduces computation time.

Our experiments show that the SPDHG algorithm is significantly faster than both the LR and the LPD reconstruction methods. However, this speed gain comes with a trade-off. We noticed that the quality of the reconstructions from the SPDHG algorithm was often times less robust in comparison. (Further details and analysis are provided in later sections.)

Algorithm 3 Stochastic Primal-Dual Hybrid Gradient (SPDHG)

1: **Input:** x_0, $y_0 = 0$, A_i, f_i, g, τ, σ_i, θ, *niter*
2: **for** $k = 1$ **to** *niter* **do**
3: Select block: $i_k = \text{random}(\text{len(A)}, 1)$
4: $z_{\text{relax}} = x_k - \tau \cdot z_k$
5: $x_{k+1} = \text{prox}_g(z_{\text{relax}})$
6: **for** each $i \in i_k$ **do**
7: $y[i] = \text{prox}_{f_i}(y[i] + \sigma_i \cdot A_i(x))$
8: **end for**
9: $z_{k+1} = z_k + A_i^*(y[i] - y_{\text{old}}[i])$
10: $z_{\text{relax}} = z_{\text{relax}} + (1 + \theta \cdot \text{extra}[i]) \cdot (z_k - z_{\text{relax}})$
11: $\tau_{k+1} = \theta \cdot \tau_k$
12: $\sigma_i = \frac{\sigma_i}{\theta}$
13: **end for**

We show the respective algorithm in algorithm 3 wherein there are four key steps to highlight:

1. **Primal Update** (line 4-5): Updates the primal variable x using a proximal operator to handle $g(x)$, which is often a non-smooth term.

2. **Dual Update** (line 7): Updates the dual variable y by considering a stochastic subset of the constraints.

3. **Adjoint Update** (line 9): Adjusts the dual variable based on the adjoint A_i^*, ensuring consistency with the current estimates.

4. **Extrapolation Step** (line 10): Enhances convergence efficiency and speed by mixing the current and past guesses.

The SPDHG algorithm is designed to minimize composite optimization problems of the form:

$$\min_{x \in X} \max_{y \in Y} g(x) + \langle Ax, y \rangle - f^*(y),$$

where $g(x)$ is a convex function, $f^*(y)$ is the Fenchel conjugate of a convex function f, and A is a linear operator. This should feel familiar and natural as the algorithm is an extension of the PDHG algorithm.

The proximal operator, denoted as $\text{prox}_h(z)$, is traditionally defined as:

$$\text{prox}_h(z) = \arg\min_u \left\{ h(u) + \frac{1}{2}\|u - z\|^2 \right\}.$$

The convergence of the SPDHG algorithm depends on the choices for the step sizes τ and σ_i, which need to satisfy the following:

$$\tau\sigma\|A_i\|^2 < 1, \quad \forall i.$$

Additionally, the relaxation and extrapolation steps are important for ensuring convergence and controlling the discrepancies introduced by the random sampling.

In this thesis, we will utilize a straightforward application of the SPDHG algorithm to observe the improvements present with the implementation of a neural network as opposed to without them. The random sampling ability of the algorithm should lend itself to be several orders of measures faster than the LPD or the LR reconstruction approaches and their variants.

2 Implementation

A note prior to the rest of the implementation section. The code that was utilized was from code bases created by the authors of the respective papers. Most of these code bases are several years old and most of the libraries and tools employed are no longer actively maintained or have

been deprecated. There is currently work ongoing at various stages to update these libraries but until then a one-stop reliably reproducible code base will not be available. That being said the Python notebooks used for the results of this thesis will be available on Github.

2.1 Operator Discretization Library

For the implementation of the code for this thesis we extensively make use of the Operator Discretization Library (ODL) [10]. The ODL library is a Python library that enables research in inverse problems on realistic or real data. The framework allows users to create operators that can encapsulate physical models. These operators can then be used like a mathematical object in, e.g., optimization methods [10].

We used the tomographic functions for creating operators that perform parallel beam analysis using the Ray Transform. We also made use of the phantom generation functionality to create an essentially endless supply of training data for the neural networks. More specifically, we utilized random phantoms for training shown in Figure (7) and used Shepp-Logan phantoms with their contrast values boosted for testing, shown in Figure (8).

2.2 Learned Reconstruction Implementation

The learned reconstruction was implemented using an example code base provided on the learned gradient tomography github. We specifically trained and tested runs using the files for partially learned gradient descent.

2.2.1 Training

For training, we trained the model for Θ parameters using the inverse time decay with a learning rate of 10^{-5}. We used the **RMSPropOptimizer** to optimize our model and ran the model for 10^5 iterations to mimic as closely as possible the conditions met in [2]. We ran the model with and without creating new parameters (note: due to deprecation, we used a **GlorotNormal** initializer as opposed to the **xavier initializer conv2d** initializer.)

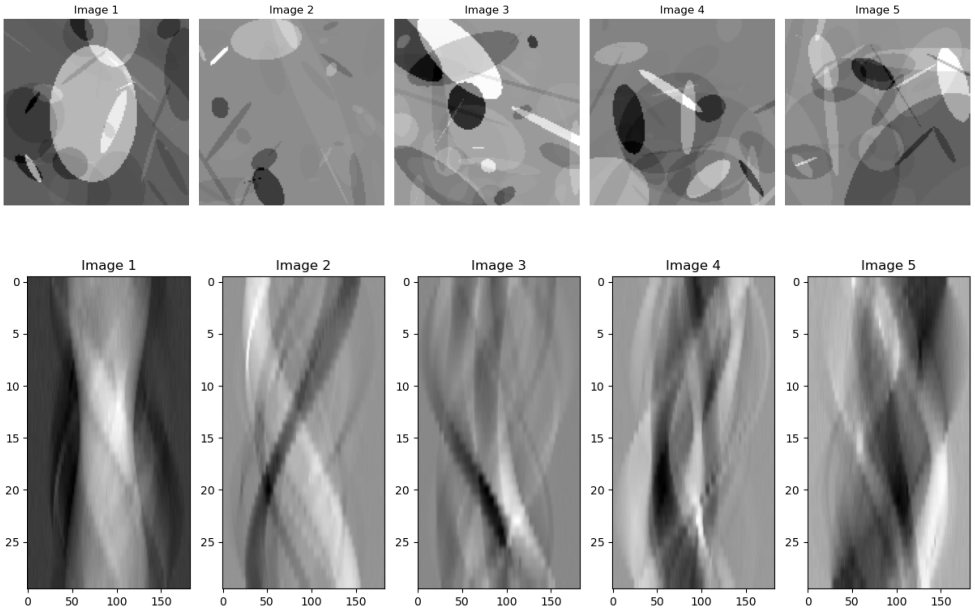

Figure 7: Example of the training ellipses and training sinograms.

As shown in Figure (7), and mentioned in the discussion on the ODL library, we dynamically generated random ellipses phantoms for the model training set. This allows for the neural network to not overfit the parameters Θ for any specific case of the Shepp-Logan phantom. This training phase took 1.73 days to run with GPU acceleration using the integrated GPU on the Apple Silicon M2 Pro chip. We used the Apple proprietary Metal Performance Shaders (MPS) to tie in with Tensorflow-metal to achieve this. Due to the computational intensity of the task at hand, we were unable to get as much training data on how variations affect the end result.

2.2.2 Testing

For testing, we did not change much to the code we pulled from GitHub for [2]. Namely, the one difference between the training and the testing phase was a change in the validation phantom. For testing purposes,

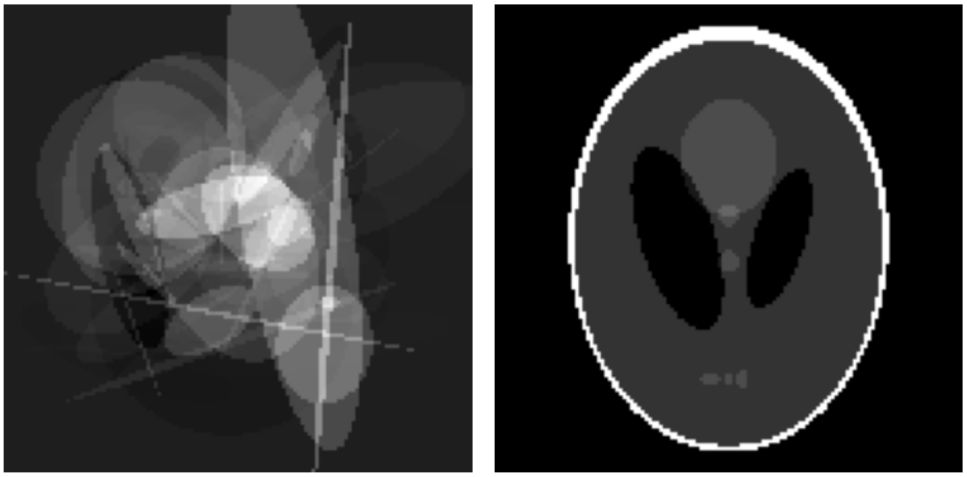

Figure 8: Side-by-side of training ellipsoid phantom and testing Shepp-Logan phantom.

we present a Shepp-Logan phantom as opposed to the training ellipses. Then there was Gaussian white noise added to the phantom before it was sent into the neural network to be reconstructed. Similar to Figure (9). After this, the reconstruction was compared against the ground truth phantom resulting in the validation loss calculation. Similar to the training phase we used the Apple integrated GPU to run the tests on the model.

Throughout the entire process, we were feeding the validation loss values per iteration to a separate Excel file to observe the trends later on. Additionally, we used the numpy and the matplotlib libraries to plot out the ending results.

We compared the results on 5 types of reconstruction we attempted:

- learned reconstruction (section 1.5)
- learned primal-dual reconstruction (section 1.6)
- learned reconstruction with no gradients
- learned primal-dual reconstruction with additional convolutional layers
- SPDHG (section 1.7).

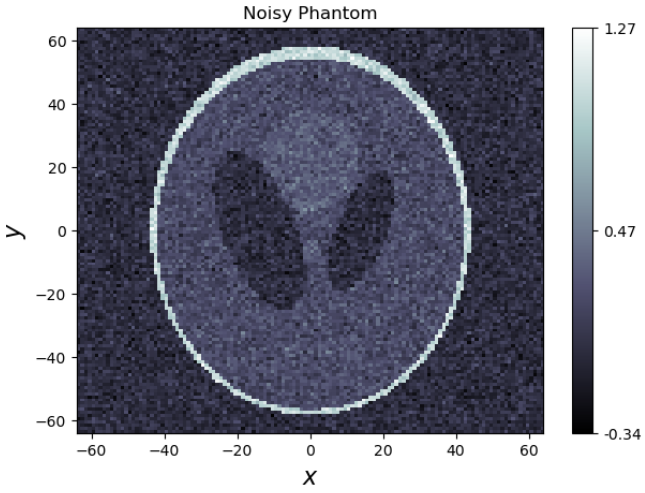

Figure 9: Illustration of noisy Shepp-Logan phantom used

2.3 Learned Primal-Dual Implementation

The learned primal-dual reconstruction was also implemented using an example code base provided on the learned primal-dual github. We noticed the code base had a reliance built on a specific folder with deprecated code for GPU acceleration. We removed all references to this code and implemented our own GPU acceleration using Apple's MPS as mentioned in section 2.2. We also noticed that the original code favored a monolithic approach which made debugging difficult. So we broke down the file into a notebook to allow for modularity, flexibility, and easier debugging. We used learned proximal operators of the form

$$Id + \mathcal{W}_{w3,b3} \circ \mathcal{A}_{c2} \circ \mathcal{W}_{w2,b2} \circ \mathcal{A}_{c1} \circ \mathcal{W}_{w1,b1}$$

where Id is the identity operator that makes the network a residual network. We use the proximal operators to make a heuristically simpler update mechanism. This way each update does not need to learn the whole update, but only a small offset from the identity [3]. The affine operators, $\mathcal{W}_{wi,bi}$ used are parameterized by the weights and biases defined by convolutional operators. more specifically:

$$\mathcal{W}_{wj,bj} : X^n \to X^m,$$

where the k:th component is given by

$$[\mathcal{W}_{wj,bj}([f^{(1)},\dots,f^{(n)}])]^{(k)} = b_j^{(k)} + \sum_{l=1}^{n} w_j^{(l,k)} * f^{(l)},$$

where $b_j \in \mathbb{R}$ and $w_j \in X^{n\times m}$.

2.3.1 Training

For the training of the LPD reconstruction, we trained a model with primal and dual updates using the training ellipses shown in Figure (7). The projection geometry was a sparse 30-view parallel beam geometry with 182 detector pixels. 5% additive Gaussian noise was added to the projections. Since the forward operator is linear, the adjoint of the derivative is simply the adjoint, which for the ray transform is the back projection.

$$[\partial\mathcal{T}(f)]^* = \mathcal{P}^*.$$

In particular, we ran the LPD reconstruction for 1000, 10,000, and 100,000 iterations. We only discuss the 100,000 iteration case but make note of the trends observed in the lower iterations.

2.3.2 Testing

For testing the LPD reconstruction, we followed a similar pattern to section 2.2.2 by feeding in the Shepp-Logan phantom with the additive Gaussian white noise to the model. The **AdamOptimizer** was used as the optimizer and modified leaky ReLU activation functions were created called **PReLU (Parameterized Rectified Linear Unit)**.

$$A_{c_j}(x) = \begin{cases} x & \text{if } x \geq 0 \\ -c_j x & \text{else.} \end{cases}$$

We utilize a TensorBoard to save the images and scalar values from the LPD reconstruction. Additionally, we also save our validation loss details to a separate Excel file and create checkpoints along the way to be able to start the network from the checkpoint instead of completely restarting the model training.

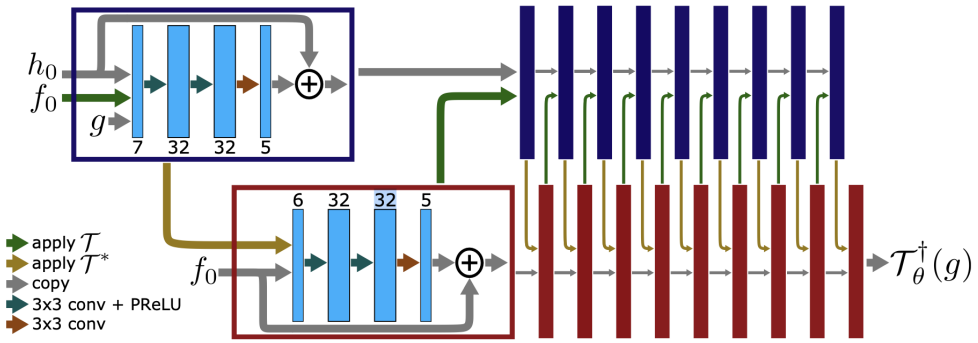

Figure 10: The network architecture addresses the tomography problem, with dual iterates represented by blue boxes and primal iterates by red boxes. Both types follow the same structure, as shown in the larger boxes. Initial guesses are input from the left, while the data is provided to the dual iterates. Arrows indicate where outputs are concatenated.

2.4 Stochastic Primal-Dual Hybrid Gradient Implementation

With the SPDHG algorithm implementation, we wanted to observe the differences between a neural network approach and a single-pass approach. We chose SPDHG for its ease of access through the ODL library (see section 2.1) and for its speedy computational speeds. To that point, we don't have a training phase for our implementation since we make direct use of algorithm (3). The process therefore is quite straightforward. We created a phantom and added approximately 5% Gaussian noise. We help the **spdhg** solver from the ODL library by creating our own function for the selection of the random subset. We wait for the solver to compute after which we calculate the Peak-Signal-to-Noise-Ratio (PSNR) and print out the results. Overall this is the simplest reconstruction to build out and certainly the most time effective. That being said, for the purpose of the accuracy required of medical tomographic imaging reconstruction, this version of the SPDHG would not be a good match. We recommend the approach laid out in [9].

3 Results & Discussion

We present the quantitative results in Table 1 while showing the reconstructions and sinograms in the figures below. Our findings indicate that the Learned Primal-Dual reconstruction consistently yields the best results based on PSNR values. However, we do make note that due to time constraints and technical limitations, we were unable to fully replicate the results found in the original thesis. To facilitate a smoother workflow, we introduced certain modifications, which may contribute to some discrepancies in the output values.

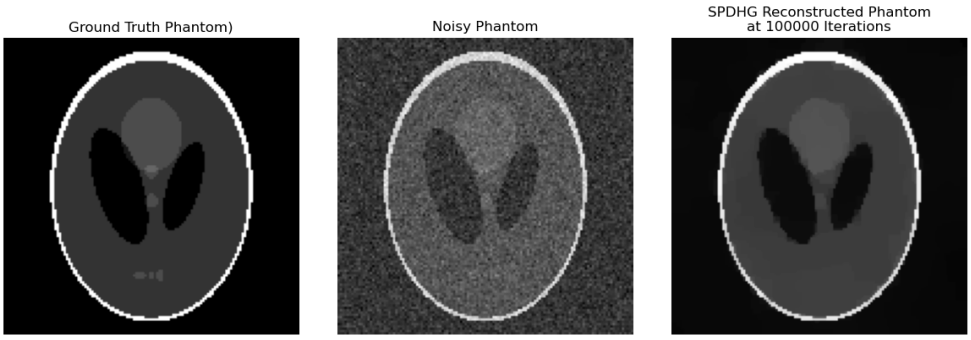

Figure 11: SPDHG reconstruction phantoms at various stages.

3.1 SPDHG

Running the SPDHG algorithm, we can compute 1,000,000 iterations in approximately 9-10 minutes, but with the caveat of the smaller details getting smudged. We can see in the SPDHG reconstruction above in Figure (11) how during the reconstruction process for a single forward pass operator not reliant on a neural network, it suffers from a loss of finer details. The final reconstruction completely obscures the three markings at the front and nearly eliminates the two lighter ellipses in the middle. Whereas, when we compare it with the LPD reconstruction, as seen in Figure (12), we preserve the presence of the markings in the front part of the phantom, albeit not completely clearly. This is still an improvement upon the SPDHG reconstruction qualitatively.

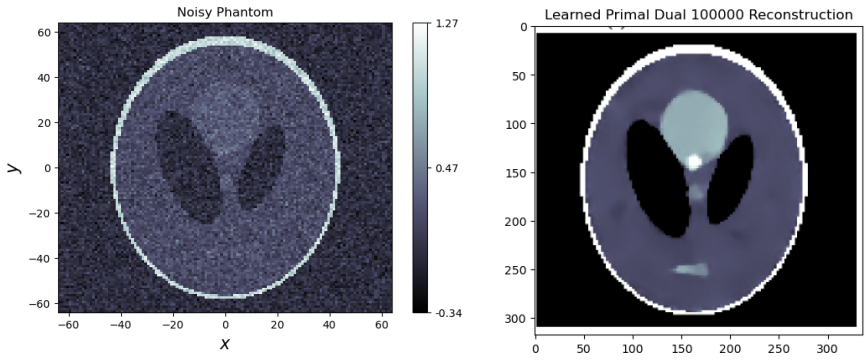

Figure 12: Going from left to right: we have the LPD reconstruction process.

3.2 Learned Reconstruction

Inspecting visually and comparing the leaned primal-dual reconstruction to the learned reconstruction from Figure (13), we can see that the reconstruction for LR loses some of the smaller details as well. Admittedly, there is some loss of details even within the LPD approach especially looking at the nodules near the middle of the phantom we see an almost smearing-like artifact. Also looking at the gray space surrounding the internal ellipses we see smearing again and streaks forming in a cyclical form potentially present because of the sparse angle approach we implemented with only 30 angles being considered as opposed to a full angle approach.

3.3 Learned Reconstruction Variants

We conducted preliminary experiments with the learned reconstruction without gradients and a further variant with a modified loss functional, resulting in rather obscure images as depicted in Figure (14). The purpose of this experiment was to see the impact of gradients on reconstruction quality, if any. As expected, the resulting phantom demonstrates the significant contribution of gradients to the overall image quality.

Shown in Table 1 are the best-performing PSNR values and validation loss values for each approach we experimented with. We showcase the graphs separately for the trends of the values as the model iterates in the

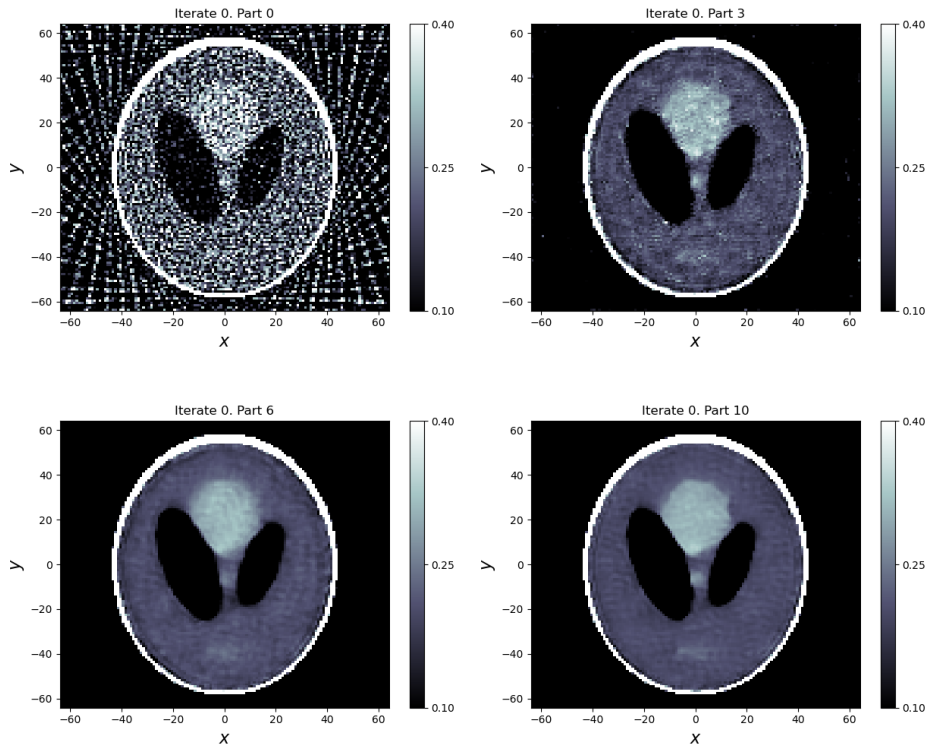

Figure 13: Going from left to right: we have the Learned reconstruction process.

appendix section. Except for the no-gradient and SPDHG reconstruction approaches, we observed increasing PSNR values and decreasing validation losses as the iterations went on.

3.4 A Note on Experimental Discrepancies

As we experimented with the approaches, we encountered an interesting case where our validation losses started increasing and PSNR kept decreasing. This was with the inclusion of the gradient-based approach specifically the Learned Reconstruction approach. In an experiment where we modified the learning rate to be much higher and a modified loss functional which ditched the MSE. We noticed at the 100,000th

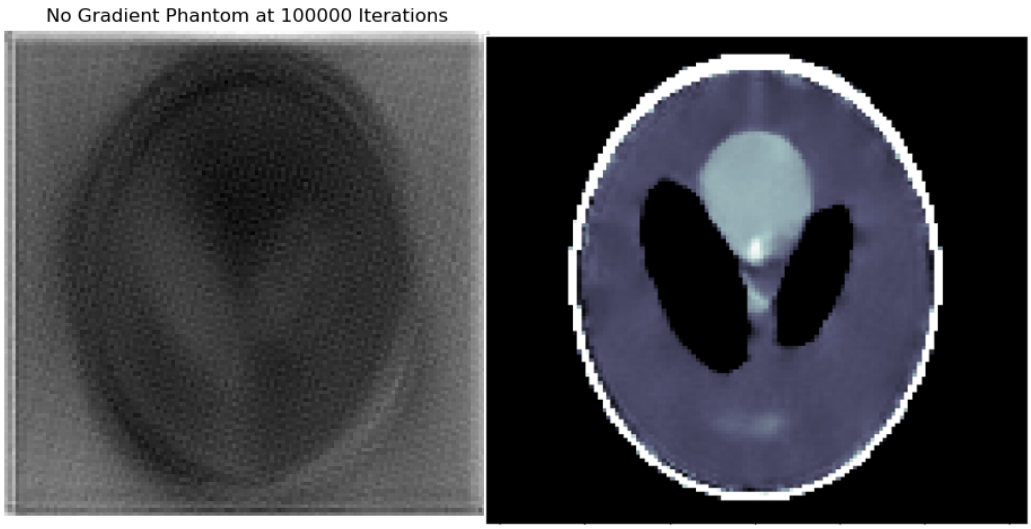

Figure 14: (Right) Learned Reconstruction with no gradient steps with a modified loss function.
(Left) Regular Learned Reconstruction with no gradient steps.

iteration that the validation loss values were in the 2000s and PSNR values were in the (-33)dB range. It was quite a bizarre phenomenon and was a great observation of the lack of failsafe mechanisms within the code itself. By analyzing this failure, we can better improve upon the algorithms not just in a mathematical manner, but also in a computational manner

PSNR and Loss Values		
Reconstruction type	PSNR (dB)	Validation Losses
Learned Reconstruction (LR)	31.873	7.221E-4
Learned Primal-Dual (LPD)	38.834	3.496E-4
LPD w/more layers	40.351	5.242E-4
Stochastic PDHG	29.170	1.210E-3
LR No-Gradients	15.026	1.941E-3

Table 1: Table containing data on PSNR and Validation Loss values

3.5 LPD Variant: Increased Layers

Upon validating the learned primal-dual reconstruction with additional layers, we observed a higher validation loss than the original learned primal-dual reconstruction. The validation loss in Table 1 is nothing more than the per-pixel difference of the original ground truth Shepp-Logan phantom and the reconstructed phantom. A higher validation loss in this scenario indicates model overfitting, meaning this approach with additional layers struggles to generalize well when given only training ellipses and tested with the Shepp-Logan phantom.

3.6 Validation Losses and PSNR

Our analysis of the results focuses on two key questions: which method worked the best? And what factors contributed to these outcomes? We solidify this discussion by combining our previous discussion on the visual inspections of the reconstructions with the values from Table 1. We observed that we found the best results with the learned primal-dual reconstruction with 2 more added convolutional layers to each of the primal and the dual iterates. We got results for PSNR values in the 40dB range. In contrast, both the SPDHG and the base learned reconstruction exhibited the lowest PSNR values.

We see that the PSNR values for both SPDHG and LR are lower than the base LPD reconstruction with SPDHG having a 9.7 dB difference and LR having a 7 dB difference. The validation losses for both of these approaches are also larger than the base LPD reconstruction. Indicating the reconstruction itself is worse than the LPD reconstruction.

Based on this evidence, we confidently say that the LPD reconstruction is the most suitable approach for the given task amongst the methods we evaluated.

4 Conclusion and Future Work

4.1 Conclusion

In this thesis, we explored the results that we achieved by running 5 different types of reconstructions for a medical tomographic imaging purpose. Our goal was to study, compare, and deduce the optimal recon-

struction approach out of the five. We discussed some of the background information related to the general problem and each of the reconstructions. And looked over the results intimately, first visually inspecting the reconstructions, then observing the quantitative results and deducing our final conclusion from them.

More specifically, we explored the following reconstructions:

- learned reconstruction (section 1.5)
- learned primal-dual reconstruction (section 1.6)
- learned reconstruction with no gradients
- learned primal-dual reconstruction with additional convolutional layers
- SPDHG (section 1.7).

We ran the reconstructions for 10e4 or 100,000 iterations before we presented the results on the PSNR and validation losses in Table (1). Summarizing our discussion, we found that of the 5 reconstructions, the learned primal-dual reconstruction was the best reconstruction that reduced overfitting from additional layers and also was able to produce closer to original reconstructions.

4.2 Future Work

4.2.1 SPDHG Implementation

Due to the technical obstacles, we were unable to properly run our Stochastic Primal-Dual Hybrid Gradient approach to reconstruction. This version of the Primal-Dual Hybrid Algorithm uses random subsets of data as opposed to the whole data. We hypothesize that with smaller datasets as well as larger datasets we will notice a speed-up in latency for the training and testing phases. We suggest the method proposed in [9] for implementation. That is to say that we extend the learned primal-dual reconstruction with the stochastic random subset selection to train our parameters Θ.

4.2.2 Error Functional Enhancements

We also suggest an improvement to the error function used. Currently we utilized the simply squared norm function $E(f) = ||f - f_{true}||_X^2$. A more sophisticated formula may yield better PSNR values and more generalizable models for our medical tomographic imaging use case. We point out with special emphasis that the most direct way to see improvements or degradations with these neural network-based computerized reconstructions is to modify their loss functions. This would be a great place to start making observations on how to improve the algorithms.

4.2.3 Technological Enhancements

One of the main hindrances to our work was old and unmaintained code. Consequently, for future work, we propose an update to the code base so that there is in-built support for TensorFlow v2.x. This will allow for simpler and more readable code plus an additional bonus to performance through the eager execution default mode. We also heavily endorse the use of the TensorBoard functionality to save data summaries about the model and its stages during the iterations. Additionally, we endorse the use of batched intermittent data logging. These two methods provide a backup in case of power failure and also intimate details about the model's training process itself. Especially as the model's training phase is not particularly short (1.73 days to 4+ days depending on GPU).

5 Acknowledgements

This work was done as a part of a bachelor's thesis for the capstone course at the University of Minnesota's Mathematics department. As such, we would like to thank Professor Ru-Yu Lai for her countless hours of supervision and for providing valuable advice from her years of experience as a researcher.

6 Appendix

In this appendix, we added some information/charts about what results we fetched from our experimentations.

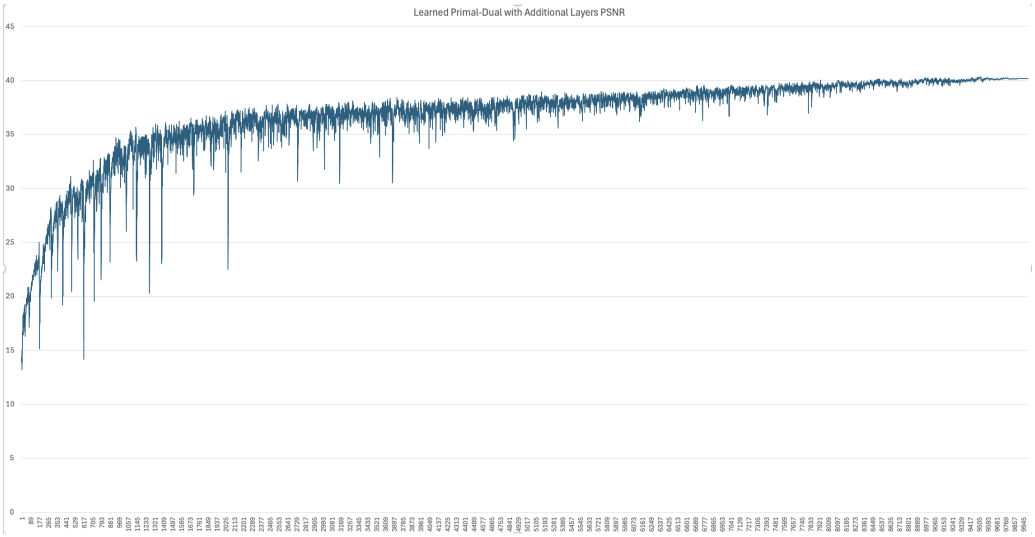

Figure 15: The graph for the PSNR values for the Learned Primal-Dual Reconstruction with additional layers.

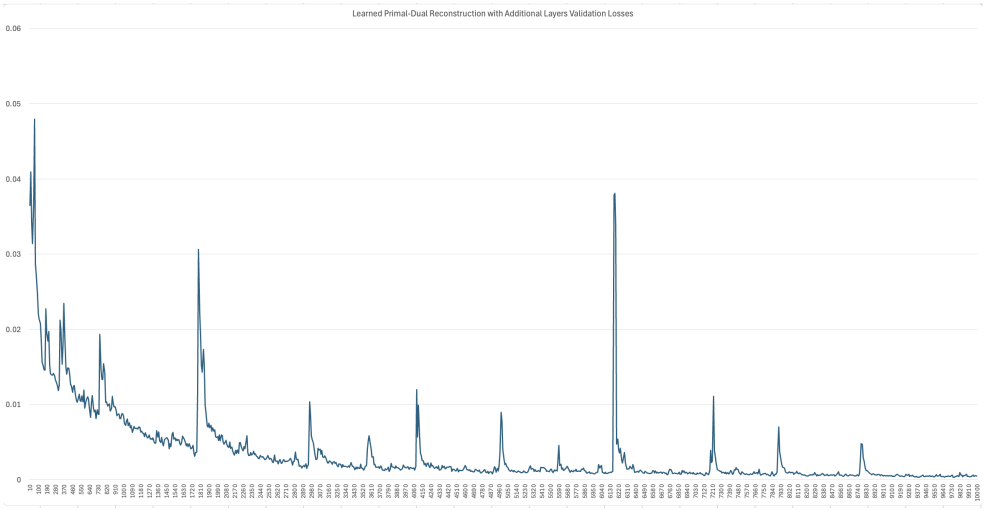

Figure 16: The graph for the validation loss values for the Learned Primal-Dual Reconstruction with additional layers.

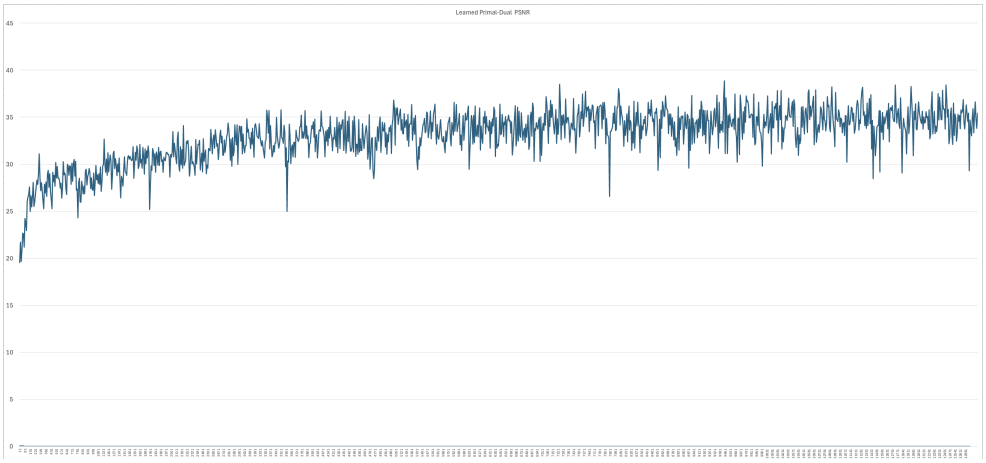

Figure 17: The graph for the PSNR values for the Learned Primal-Dual Reconstruction.

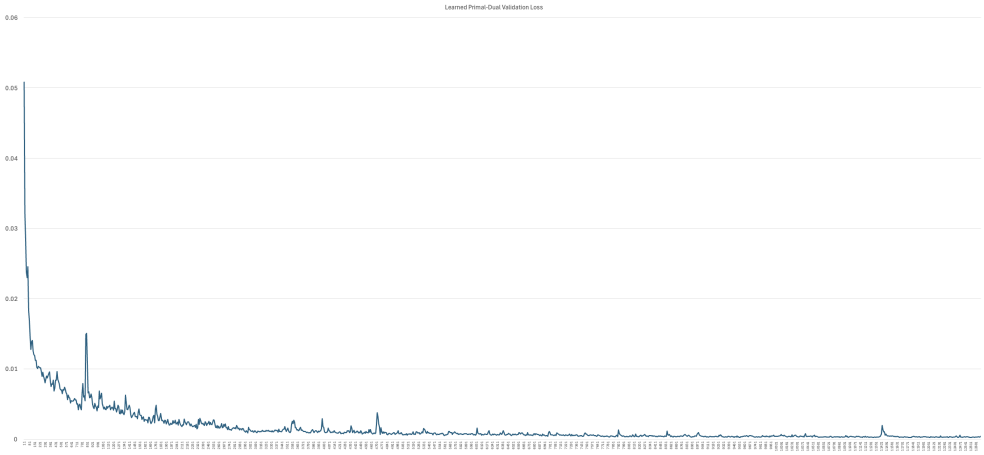

Figure 18: The graph for the validation loss values for the Learned Primal-Dual Reconstruction.

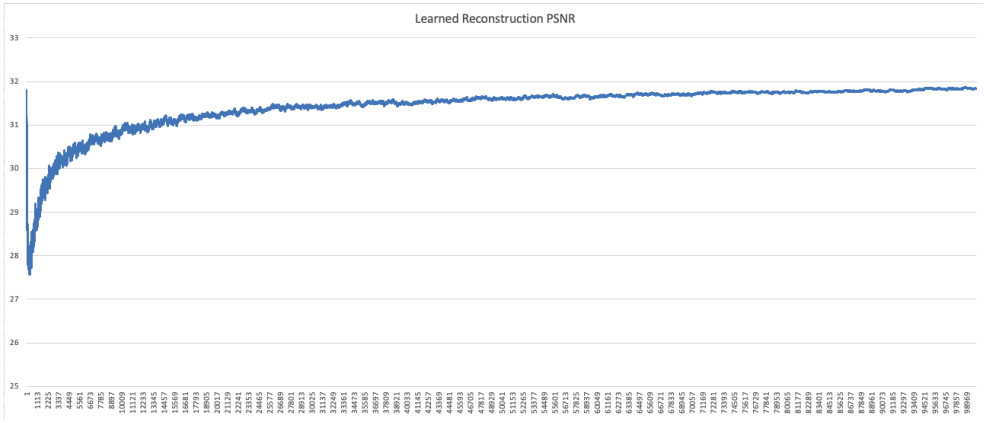

Figure 19: The graph for the PSNR values for the Learned Reconstruction.

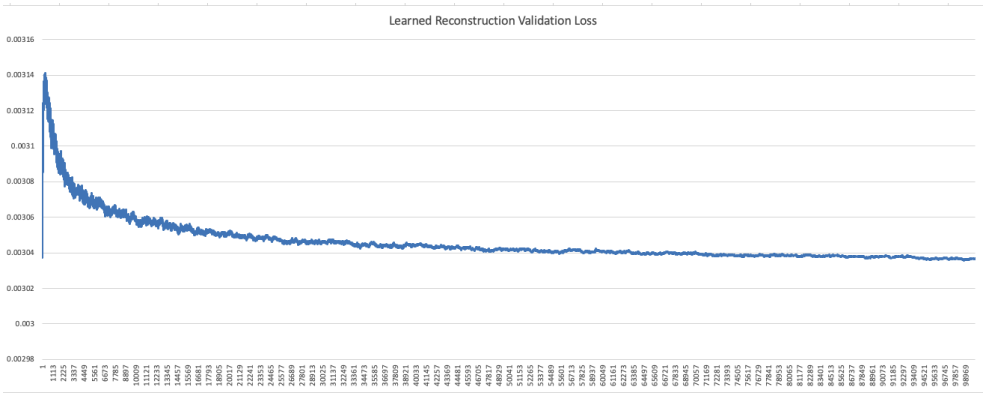

Figure 20: The graph for the validation loss values for the Learned Reconstruction.

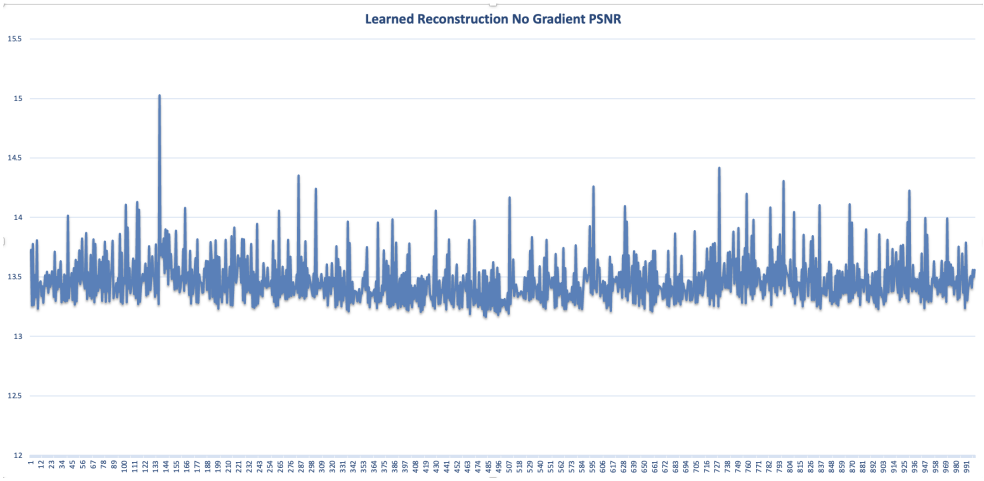

Figure 21: The graph for the PSNR values for the Learned Reconstruction without gradients.

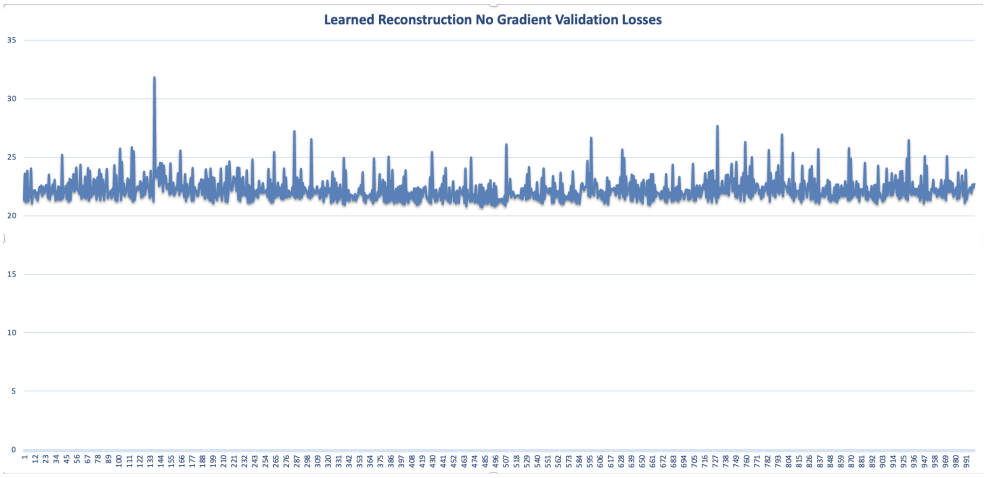

Figure 22: The graph for the validation loss values for the Learned Reconstruction without gradients.

References

[1] S. Siltanen and J. L. Mueller. *Linear and Nonlinear Inverse Problems with Practical Applications*. Society for Industrial and Applied Mathematics (SIAM), 2012.

[2] J. Adler and O. Öktem. "Solving ill-posed inverse problems using iterative deep neural networks". In: *Inverse Problems* 33 (2017). DOI: https://doi.org/10.1109/tmi.2018.2799231.

[3] J. Adler and O. Öktem. "Learned Primal-dual Reconstruction". In: *IEEE Transactions on Medical Imaging* 37 (6 2018).

[4] A. Chambolle and T. Pock. "A First-Order Primal-Dual Algorithm for Convex Problems with Applications to Imaging". In: *Journal of Mathematical Imaging and Vision* 40 (2010). DOI: https://doi.org/10.1007/s10851-010-0251-1.

[5] A. Chambolle et al. "Stochastic Primal Dual Hybrid Gradient Algorithm with Adaptive Step-Sizes". In: *Journal of Mathematical Imaging and Vision* 66 (2024). DOI: https://doi.org/10.1007/s10851-024-01174-1.

[6] A. Chambolle et al. "Stochastic Primal-Dual Hybrid Gradient Algorithm with Arbitrary Sampling and Imaging Applications". In: *SIAM Journal on Optimization* 28 (4 2018).

[7] M. Sjöberg and T. Charitidis. *Reconstruction in Computerized Tomography using Entropy Regularization, Bachelor Thesis*. 2017.

[8] T. Valkonen. "A primal-dual hybrid gradient method for nonlinear operators with applications to MRI". In: *Inverse Problems* 30 (5 2014). DOI: https://doi.org/10.1088/0266-5611/30/5/055012.

[9] J. Tang et al. "Stochastic Primal-Dual Deep Unrolling". In: *arXiv.2110.10093* (2021). URL: https://doi.org/10.48550/arXiv.2110.10093.

[10] J. Adler, H. Kohr, and O. Öktem. *ODL - a python framework for rapid prototyping in inverse problems*. 2017. URL: https://github.com/odlgroup/odl.