Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter now, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Junyuan Wu April 8, 2018

A Trainable Conjugate Gradient Method for Image Reconstruction

by

Junyuan Wu

Lars Ruthotto
Adviser

Department of Mathematics

Lars Ruthotto

Adviser

Yuanzhe Xi

Committee Member

Shun Yan Cheung

Committee Member

2019

A Trainable Conjugate Gradient Method for Image Reconstruction

By

Junyuan Wu

Lars Ruthotto

Adviser

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Sciences with Honors

Department of Mathematics

2019

Abstract

A Trainable Conjugate Gradient Method for Image Reconstruction
By Junyuan Wu

Deep learning has become an important tool in imaging classification, recognition, and recently in reconstruction. Image reconstruction is an ill-posed inverse problem, which is commonly solved by minimizing an objective function consisting of a data misfit term and a regularization term. Two key challenges in solving inverse problems are to design an effective regularization term and iterative solver.

This thesis presents a trainable Conjugate Gradient method that we call VNCG. Our method is obtained by following the framework of variational networks (VN), with the key idea of unrolling and training a CG method with fixed number of iterations. In our numerical experiments, we consider linear inverse problems and train a convolution stencil that represents the regularization operator in a Tikhonov form. We compare two strategies: using a constant stencil for all iterations or a more flexible approach that assigns different stencils to each iteration.

A Trainable Conjugate Gradient Method for Image Reconstruction

By

Junyuan Wu

Lars Ruthotto

Adviser

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Sciences with Honors

Department of Mathematics

2019

Contents

List of Figures

Chapter 1

Introduction

During the past decades, machine learning has become an important tool in image classification, recognition, and recently in reconstruction. Among various machine learning models, a neural network is a model designed to simulate how human brains work to solve problems. In this work, we use the idea of machine learning for solving image reconstruction problems.

An image reconstruction problem is an ill-posed inverse problem of finding a solution image $\mathbf{x}$ such that $\mathbf{b} = \mathbf{A}\mathbf{x} + \mathbf{n}$, where matrix $\mathbf{A}$ is an operator and $\mathbf{n}$ is the noise. Due to ill-posedness, the solution $\mathbf{x}$ is highly sensitive to the noise $\mathbf{n}$. Thus, an ill-posed inverse problem is commonly solved by minimizing an objective function consisting of a data misfit term and a regularization term. Two key challenges in solving inverse problems are to design an effective regularization term and iterative solver.

There are many iterative solvers for a minimizing problem. One of these solvers is the Gradient Descent method (GD) used in [2]. Innovated by the fact that the

Conjugate Gradient method (CG) converges faster than GD, we present in this thesis a trainable CG method that we call VNCG following the framework of variational networks (VN) [2], which is a neural network with the key idea of unrolling and training an image reconstruction algorithm.

In our numerical experiments, we train a convolution stencil that represents the regularization operator in a Tikhonov form. We compare two strategies: using a constant stencil for all iterations or a more flexible approach that assigns different stencils to each iteration.

In this thesis, we design three numerical experiments to train VNCG for reconstruction of Shepp-Logan images [8], general blurred images and undersampled tomography. The reconstruction of undersampled tomography is of great importance in medical areas. Since the acquisition of the reconstructing data (such as MRI data) involves inspections that might have adverse effects to patients, it would be preferable to do inspections on smaller areas but obtain the information with similar accuracy. Also, considering the cost of medical machines to do inspections and acquire data, smaller inspection areas also decrease the economic pressure of patients, hospitals, and clinics. Thus, we want to train VNCG to improve the accuracy of reconstruction from undersampled data.

Contributions and Outline

In this thesis, we present a trainable CG which we call VNCG. In practice, we train VNCG to learn the convolution stencil(s) of an image reconstruction problem. Through numerical experiments, we show that by training VNCG, we could make improvement in reconstructions as in Figure 1.1.

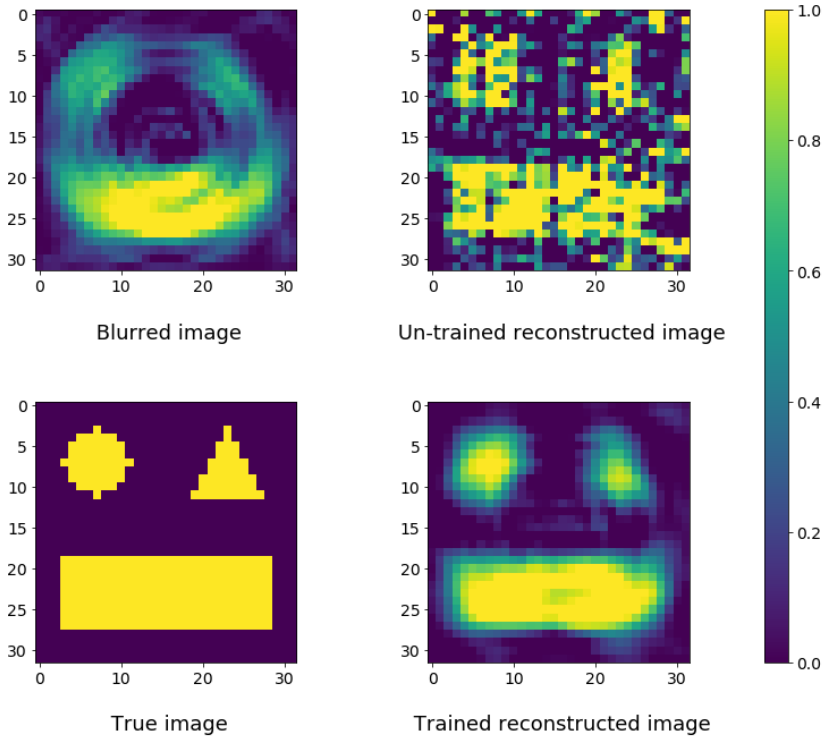


Figure 1.1: Perfomance of VNCG for image deblurring. The true image (left-bottom) is blurred by a rotational blurring operator and becomes the blurred image (left-top). Images share a common color bar. With an un-trained, random convolution operator, CG results in the reconstructed image as (right-top). Take the untrained operator as an initial guess and train VNCG with the Gradient Descent method (GD), the learned, constant operator reconstructs the image as (right-bottom).

This thesis is organized as follows. Chapter 2 generalizes CG to operate on matrices, introduces Tikhonov regularization to the reconstruction problem and builds up VNCG for learning the convolution stencil.

Based on the theory, we design three experiments in Chapter 3 to examine the performance of VNCG in solving the following reconstruction problems:

1. Sparse (tomography) forward problem matrix and Shepp-Logan phantoms

(Section 3.2).

2. General image debluring that treats filter as a convolution operator (Section 3.3).

3. Undersampled tomographic forward problem and Sheep-Logan phantoms (Section 3.4).

Based on results of the above experiments, Chapter 4 summarizes the findings and provides directions for future works.

Chapter 2

Background

To better illustrate how we could train the Conjugate Gradient method (CG) as a variational network (VN), in this chapter we first briefly show the implementation of CG as an iterative method. Then we bring in Tikhonov regularization to the inverse problem and introduce the structure of our VNCG. Finally, we show the loss function of the learning problem and train the operator of the Tikhonov regularization.

2.1 Conjugate Gradient Method

In this work, for convenience of training, a bijective mapping from an image $\hat{\mathbf{x}} \in \mathbb{R}^{n \times n}$ to a vector $\mathbf{x} \in \mathbb{R}^{n^2}$ is defined. With this mapping, a set of images $\hat{\mathbf{X}} \in \mathbb{R}^{n \times n \times m}$, where m is the number of images in this set, can be converted to a 2D matrix $\mathbf{X} \in \mathbb{R}^{n^2 \times m}$.

Now we consider an ill-posed inverse problem of finding reconstructed set of

images $\mathbf{X} \in \mathbb{R}^{n^2 \times m}$ from a given image set $\mathbf{B} \in \mathbb{R}^{k \times m}$ satisfying the following system of equations

$$\mathbf{B} = \mathbf{A}\mathbf{X} + \mathbf{N},$$

where $\mathbf{A} \in \mathbb{R}^{k \times n^2}$ is an operator and $\mathbf{N} \in \mathbb{R}^{k \times m}$ is the noise. The inverse problem is sensitive to noise and modeling errors, which is said to be ill-posed. Because of ill-posedness, we solve this problem as an optimization problem where we minimize the least square error

$$\min_{\mathbf{X}} \frac{1}{2} \Big|\Big| \mathbf{A}\mathbf{X} - \mathbf{B} \Big|\Big|_F^2. \tag{2.1}$$

There are plenty of approaches to solve this minimizing problem. One of these approaches is to use the Gradient Descent method (GD) as in [2]. Here in this thesis, since CG converges in less iterations than GD (see [3, 7]), we use CG as an iterative algorithm to solve the normal equation corresponding to the minimizing problem [4],

$$\mathbf{A}^T\mathbf{A}\mathbf{X} = \mathbf{A}^T\mathbf{B}. \tag{2.2}$$

To solve $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$ for a vector $\mathbf{x}$, where $\mathbf{A}$ is some matrix and $\mathbf{b}$ is a vector, CG starts with an initial guess $\mathbf{x}_0$ and computes the residual $\mathbf{r}_k$ and the moving direction $\mathbf{p}_k$ in its k^{th} iteration following procedures in Algorithm 1 to get $\mathbf{x}_k$ after k iterations [3].

Then we use Algorithm 1 for solving equation 2.2, where we deal with matrices $\mathbf{X}$ and $\mathbf{B}$. Here we consider every image $\mathbf{x}$ in the set separately, which means that we do operations in Algorithm 1 on every column of $\mathbf{X}$. For every step in the algorithm, we compute the result for each column, and put the results next to each other to be the result of the step. In this way, we extend Algorithm 1 to

Result: approximate result $\mathbf{x}_N$ of $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$
$\mathbf{A}_{\text{new}} = \mathbf{A}^T\mathbf{A}$;
$\mathbf{b}_{\text{new}} = \mathbf{A}^T\mathbf{b}$;
$\mathbf{r}_0 = \mathbf{b}_{\text{new}} - \mathbf{A}_{\text{new}}\mathbf{x}$;
$\mathbf{p}_0 = \mathbf{r}_0$;
$k = 0$;
while $k \leq N$ **do**
  $\alpha_k = \dfrac{\mathbf{r}_k^T\mathbf{r}_k}{\mathbf{p}_k^T\mathbf{A}_{\text{new}}\mathbf{p}_k}$;
  $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$;
  $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\mathbf{A}_{\text{new}}\mathbf{p}_k$;
  $\beta_k = \dfrac{\mathbf{r}_{k+1}^T\mathbf{r}_{k+1}}{\mathbf{r}_k^T\mathbf{r}_k}$;
  $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$;
  $k = k + 1$;
end

Algorithm 1: Conjugate Gradient Method for vectors. N is the total number of iterations. $\mathbf{r}_k$, $\mathbf{p}_k$ are vectors and α_k, β_k are scalars.

Algorithm 2 for solving $\mathbf{AX} = \mathbf{B}$, where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{X}$ are all matrices.

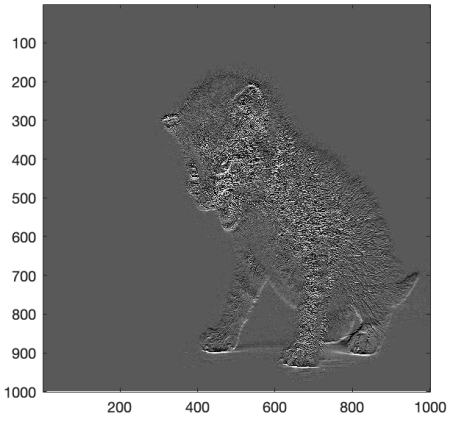
With Algorithm 2, we can find an approximate solution $\mathbf{X}_N$ to Equation 2.2. However, because of the noise $\mathbf{N}$, there might exist over-fitting in our method, which we discuss in detail in the next section [9].

Result: approximate result $\mathbf{X}_N$ of $\mathbf{AX} = \mathbf{B}$
$\mathbf{A}_{\text{new}} = \mathbf{A}^T\mathbf{A}$;
$\mathbf{B}_{\text{new}} = \mathbf{A}^T\mathbf{B}$;
$\mathbf{R}_0 = \mathbf{B}_{\text{new}} - \mathbf{A}_{\text{new}}\mathbf{X}$;
$\mathbf{P}_0 = \mathbf{R}_0$;
$k = 0$;
while $k \leq N$ **do**
  $\mathbf{a}_k = \dfrac{\mathbf{R}_k^T\mathbf{R}_k}{\mathbf{P}_k^T\mathbf{A}_{\text{new}}\mathbf{P}_k}$;
  $\mathbf{X}_{k+1} = \mathbf{X}_k + \mathbf{a}_k\mathbf{P}_k$;
  $\mathbf{R}_{k+1} = \mathbf{R}_k - \mathbf{a}_k\mathbf{A}_{\text{new}}\mathbf{P}_k$;
  $\mathbf{b}_k = \dfrac{\mathbf{R}_{k+1}^T\mathbf{R}_{k+1}}{\mathbf{R}_k^T\mathbf{R}_k}$;
  $\mathbf{P}_{k+1} = \mathbf{R}_{k+1} + \mathbf{b}_k\mathbf{P}_k$;
  $k = k + 1$;
end

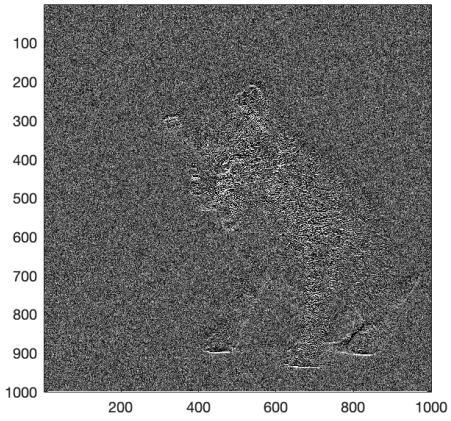
Algorithm 2: Generalized Conjugate Gradient Method for matrices. N is the total number of iterations. $\mathbf{R}_k, \mathbf{P}_k$ are matrices and $\mathbf{a}_k$, $\mathbf{b}_k$ are vectors. In the iterations, we define $\mathbf{R}_k^T\mathbf{R}_k$ as column-wise inner products, matrix division $\mathbf{A}_k/\mathbf{B}_k$ as column-wise division, and $\mathbf{a}_k\mathbf{P}_k = \text{diag}(\mathbf{a}_k) \times \mathbf{P}_k$.

2.2 Trainable Regularization Operators

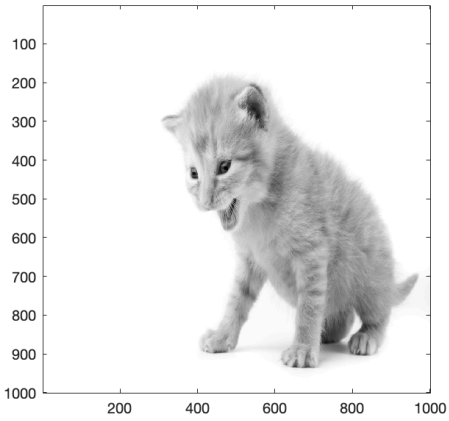
In statistics and machine learning, over-fitting happens when a model fits the data too closely and may, therefore, fail to fit additional data [9]. In solving reconstruction problems, the semi-convergence property of an iterative method states that when there are too many iterations, the reconstructed image converges to the true image in its first several iterations and then goes into a wrong direction. As shown in Figure 2.1, the reconstructed image 2.1d might look totally different from the true image 2.1c.



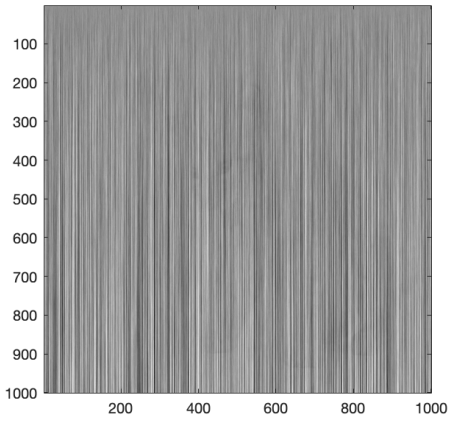
(a) **B** without noise



(b) **B** with noise



(c) Reconstructed from 2.1a: a cat



(d) Reconstructed from 2.1b

Figure 2.1: True image vs. over-fitted reconstruction image. The color bars of data images are adjusted for visibility.

There are many approaches to prevent over-fitting, one of which is to stop the iterations early by having a small number of iterations N. Another approach we could use is to introduce a regularization term $R(\mathbf{X})$ to the original least square

problem

$$\min_{\mathbf{X}} \left[\lambda R(\mathbf{X}) + \frac{1}{2} \left|\left| \mathbf{AX} - \mathbf{B} \right|\right|_F^2 \right],$$

where $\lambda > 0$ controls the influence of the regularization term.

In this thesis, we pick the Tikhonov regularization

$$R(\mathbf{X}) = \frac{1}{2} \left|\left| \mathbf{LX} \right|\right|_2^2,$$

where $\mathbf{L}$ is a convolution operator. Then the problem becomes

$$\min_{\mathbf{X}} \frac{1}{2} \left(\left|\left| \mathbf{LX} \right|\right|_2^2 + \left|\left| \mathbf{AX} - \mathbf{B} \right|\right|_F^2 \right),$$

where λ is implicitly included in $\mathbf{L}$. Then we could generate the normal equation corresponding to the least square problem

$$\mathbf{L}^T\mathbf{LX} + \mathbf{A}^T\mathbf{AX} = \mathbf{A}^T\mathbf{B}.$$

Since $\mathbf{L}$ is a convolution operator, we define a function conv such that

$$\mathbf{LX} = \text{conv}(\mathbf{K}, \mathbf{X}),$$

where the convolution kernel $\mathbf{K}$ is a small (3×3), sparse matrix. And we define the transpose of the convolution

$$\mathbf{L}^T\mathbf{X} = \text{conv}^T(\mathbf{K}, \mathbf{X}).$$

Then the normal equation to solve becomes

$$\mathrm{conv}^T\Big(\mathbf{K},\, \mathrm{conv}\big(\mathbf{K},\, \mathbf{X}\big)\Big) + \mathbf{A}^T\mathbf{A}\mathbf{X} = \mathbf{A}^T\mathbf{B}. \tag{2.3}$$

In this work, our reconstruction algorithm is to solve Equation 2.3 with extended CG (Algorithm 2), which we define as a function

$$\mathbf{X}_N = \mathrm{CG}(\mathbf{A},\, \mathbf{B},\, \mathbf{K}_{\mathrm{const}},\, N), \tag{2.4}$$

where besides $\mathbf{A}$ and $\mathbf{B}$, we take two more parameters: the number of iterations N and a convolution kernel $\mathbf{K}_{\mathrm{const}}$. In our work, we also want our reconstruction algorithm to allow different convolution kernels in each iteration of CG, so we also define our algorithm as

$$\mathbf{X}_N = \mathrm{CG}(\mathbf{A},\, \mathbf{B},\, \mathbf{K}_{\mathrm{free}}), \tag{2.5}$$

where $\mathbf{K}_{\mathrm{free}}$ is a list of convolution kernels for each iteration, and its first dimension indicates the number of iterations $\mathbf{K}_{\mathrm{free}}[0] = N$.

As defined in Equation 2.4, 2.5, the choice of convolution kernel(s) $\mathbf{K}$ is crucial to our reconstruction algorithm. Thus, to find an optimal value of $\mathbf{K}$, we want to use machine learning techniques to learn $\mathbf{K}$ from a neural network, which will be introduced in detail in the next section.

2.3 Variational Network

With regularization introduced to CG (Algorithm 2) as our reconstruction algorithm, we aim to build up a neural network that can be trained to learn the

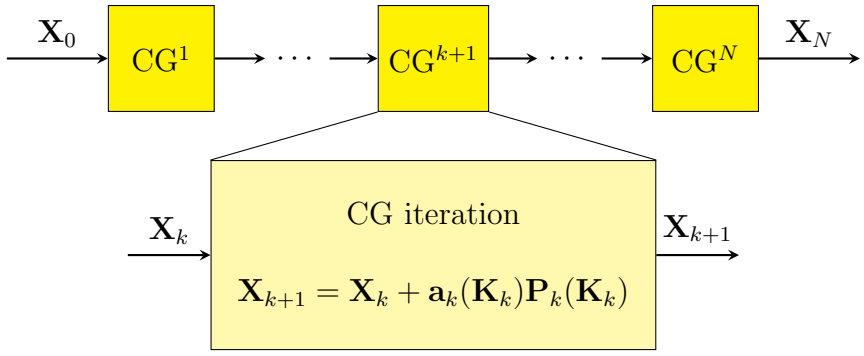


Figure 2.2: Structure of VNCG. $\mathbf{a}_k$ and $\mathbf{P}_k$ are defined in Algorithm 2.

convolution kernel of our reconstruction algorithm. We take our reconstruction algorithm as the forward propagation

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \mathbf{a}_k(\mathbf{K}_k)\mathbf{P}_k(\mathbf{K}_k), \quad k = 0,\, 1,\, 2,\, \cdots,\, N-1,$$

where N is the total number of iterations in our algorithm, and the moving step sizes $\mathbf{a}_k$ and moving directions $\mathbf{P}_k$ defined as in Algorithm 2 both take the convolution kernel $\mathbf{K}_k$ as a parameter. As in [2], a neural network that takes a reconstruction algorithm as forward propagation is termed as a VN. To avoid confusion, we term our network to be VNCG.

As shown in Figure 2.2, VNCG has $N+1$ layers in total, where N is the total number of iterations in CG. We set the initial guess of our algorithm $\mathbf{X}_0$ to be the input layer, the reconstructed image $\mathbf{X}_N$ to be the output layer, and all $\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_{N-1}$ be hidden layers.

2.4 Training Convolution Operator

With the VN depicted in Figure 2.2, we now define a learning problem that aims at estimating the convolution kernels in the forward propagation so that the forward propagation finds a reconstructed image $\mathbf{X}$ closest to the true image $\mathbf{X}_{\text{true}}$. Therefore, we set up the loss function of the learning problem to be

$$\mathcal{L}(\mathbf{K}) = \frac{1}{2}\left|\left|\mathbf{X} - \mathbf{X}_{\text{true}}\right|\right|_F^2,$$

where $\mathbf{X}$ denotes the approximate result given by the forward propagation and $\mathbf{X}_{\text{true}}$ is the true image. We then rephrase the learning problem to an optimization problem

$$\min_{\mathbf{K}} \mathcal{L}(\mathbf{K}) \text{ s.t. } \mathbf{X}_{k+1} = \mathbf{X}_k + \mathbf{a}_k(\mathbf{K}_k)\mathbf{P}_k(\mathbf{K}_k),\ k = 0,\ 1,\ 2,\ \cdots,\ N-1.$$

There are many approaches to solve this learning problem. In this work, we use a standard Gradient Descent method (GD). The implementations are explained in detail in Chapter 3.

Chapter 3

Numerical Experiments

In order to perform some numerical experiments, we design an experimental method based on the theory stated in Chapter 2. In section 3.1, we explain our method in three parts: data acquisition, network training, and validation.

With this method, we conduct three experiments. We first start with a tomographic Shepp-Logan phantom reconstruction in section 3.2 [8]. Then we continue to train VNCG to deblur general images in section 3.3. In the end, we work on undersampled tomographic reconstruction in section 3.4. Through these three experiments, we show that trained convolution kernels reconstruct the images better than untrained ones. Moreover, we show that allowing different convolution kernels for each iteration provides a better reconstruction results than using a constant kernel for image deblurring. The experiment results and analysis are stated in detail in this chapter.

3.1 Methods

3.1.1 Data Acquisition

For the convenience of computation and image display, all experiments shown in this work are conducted on one-channel grey value images only. However, the experiments can be easily extended to three-channel RGB images, which only requires several changes to the implementation code.

In the experiments, we separately generate two sets of data based on their usage: the training set S for training VNCG to learn the convolution kernel(s) in 3.1.2, and the validation set S' for the test of validity in reconstructing other images in 3.1.3.

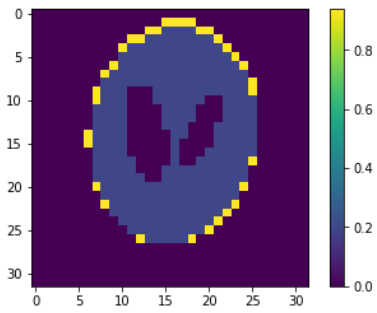
All the data we use in our experiments are generated through the following MATLAB functions:

1. randomSheppLogan(n, param) for Shepp-Logan phantoms [6].
2. PRtomo(varargin) for tomographic operators[1].
3. PRblur(varargin) for blurring stencils and images [1].

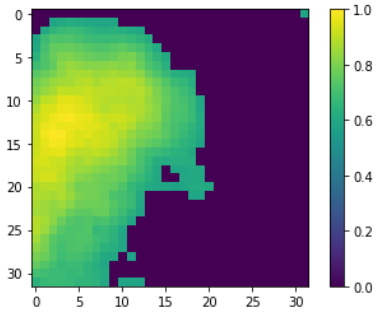
Examples of generated images are shown in Figure 3.1.

In our experiments, for every inverse problem, we first generate the true images $\mathbf{X}_{\text{true}}$ and the common operator $\mathbf{A}$, from which we add noise $\mathbf{N}$ randomly distributed for every single image and generate the data $\mathbf{B}$ to reconstruct. The noise level we use in the experiments is

$$\frac{1}{100}||\mathbf{B}||_2 \leq ||\mathbf{N}||_2 \leq \frac{1}{10}||\mathbf{B}||_2.$$



(a) Sample Shepp-Logan phantom: a model of human head.



(b) Sample general image: a random image with patterns of nonzero pixels

Figure 3.1: Sample images generated by MATLAB code.

3.1.2 Training VNCG

In this work, we implement VNCG training and validation with pytorch, which is an open-source deep learning library [5]. Related python code for our method is listed in Appendix C.

We separate the training section of our method into three steps, which are listed in detail below:

1. Before we start training, we first manually choose the kernel to be a discrete Laplacian

$$\mathbf{K}_{\text{lap}} = \begin{vmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{vmatrix}. \tag{3.1}$$

   Without training VN, we take $\mathbf{K}_{\text{lap}}$ as parameter of the Conjugate Gradient method (CG) and run Equation 2.4 to get

$$\mathbf{X}_{\text{lap}} = \text{CG}(\mathbf{A}, \mathbf{B}, \mathbf{K}_{\text{lap}}, N),$$

where $\mathbf{X}_{\text{lap}}$ is the reconstructed image and N is number of iterations.

For comparison, we also randomly build a kernel $\mathbf{K}_{\text{ran}}$. Also without training, take $\mathbf{K}_{\text{ran}}$ as a parameter and run Equation 2.4 to get

$$\mathbf{X}_{\text{ran}} = \text{CG}(\mathbf{A}, \mathbf{B}, \mathbf{K}_{\text{ran}}, N),$$

where $\mathbf{X}_{\text{ran}}$ is the reconstructed image and N is number of iterations.

2. Then we want to start training VNCG to learn the convolution kernel. We first pick $\mathbf{K}_{\text{lap}}$ or $\mathbf{K}_{\text{ran}}$ as an initial guess of our learning problem. With the initial guess, we go through the learning process to find a new learned convolution kernel $\mathbf{K}_{\text{const}}$ for reconstruction. Then we run Equation 2.4 to get

$$\mathbf{X}_{\text{const}} = \text{CG}(\mathbf{A}, \mathbf{B}, \mathbf{K}_{\text{const}}, N),$$

where $\mathbf{X}_{\text{const}}$ is the reconstructed image and N is number of iterations.

3. In the last step of training, we want the forward propagation to allow different convolution kernels in each layer of VNCG. That is to say, we can now have different kernels in each iteration of CG. To realize it, we first build a list of N convolution kernels as a $N \times 3 \times 3$ matrix, where N is the number of iterations of CG. Initially, every kernel of the set equals $\mathbf{K}_{\text{const}}$. Then we take this set of kernels as the initial guess of our learning problem and find a new learned list of convolution kernels $\mathbf{K}_{\text{free}}$. Then we take $\mathbf{K}_{\text{free}}$ as an input parameter of CG and run Equation 2.5 to get

$$\mathbf{X}_{\text{free}} = \text{CG}(\mathbf{A}, \mathbf{B}, \mathbf{K}_{\text{free}}),$$

where $\mathbf{X}_{\text{free}}$ is the reconstructed image.

In our implementation, we take $\mathbf{X}_0 = \mathbf{0}$ to be the initial guess of the CG, and $N = 20$ to be the number of iterations of CG by default.

When training VNCG, we use the Gradient Descent method (GD) with constant step size and batch size being the number of images. We implement GD with torch.optim.SGD() class of torch package [5]. The optimizer SGD class takes three inputs: the trained parameter(s), the learning rate and the momentum. In this work, we keep the momentum to be 0.9 for all training processes and adjust the learning rate to find the optimal kernel(s).

To improve the performance of training VNCG, we decide to try different learning rates in our training process. We introduce a new parameter lr_num to denote the number of learning rates we try in a training process. The optimizer is implemented in a loop, where the learning rate is decreased by half in each iteration. In this way, we can see the relationship between the learning rate and the loss. Moreover, we introduce another parameter epoch_num to denote the number of epochs of each optimizer. By default, we set lr_num = 4, and epoch_num = 10.

There are two ways for us to evaluate the performance of VNCG. One is by comparing visualized reconstructed images $\mathbf{X}_{\text{ran}}$, $\mathbf{X}_{\text{lap}}$, $\mathbf{X}_{\text{const}}$, and $\mathbf{X}_{\text{free}}$ with the true image $\mathbf{X}_{\text{true}}$. If we see significant improvement in the images, then we could say that VNCG provides a better reconstruction for training data. Also, this comparison is used for the validation data, which will be discussed in detail in 3.1.3. The other way to evaluate the performance of VNCG is to look at the loss in the training process. If the loss is decreased by a significant percentage, then we could say that VNCG provides an improvement of some percentage.

3.1.3 Validation

After the training, we run the following reconstructions

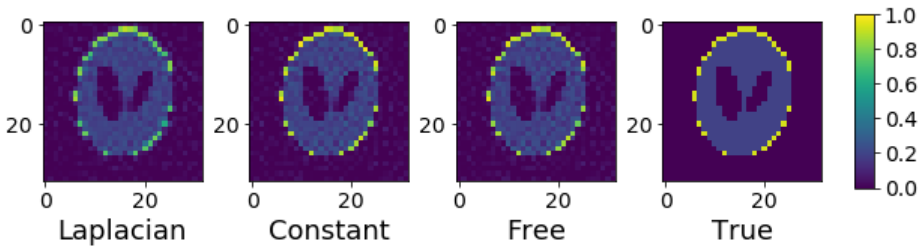
$$\begin{aligned}
\mathbf{X}'_{\text{ran}} &= \text{CG}(\mathbf{A}', \mathbf{B}', \mathbf{K}'_{\text{ran}}, N) \\
\mathbf{X}'_{\text{lap}} &= \text{CG}(\mathbf{A}', \mathbf{B}', \mathbf{K}'_{\text{lap}}, N) \\
\mathbf{X}'_{\text{const}} &= \text{CG}(\mathbf{A}', \mathbf{B}', \mathbf{K}'_{\text{const}}, N) \\
\mathbf{X}'_{\text{free}} &= \text{CG}(\mathbf{A}', \mathbf{B}', \mathbf{K}'_{\text{free}})
\end{aligned}$$

for the validation data. By comparing the resulting images $\mathbf{X}'_{\text{ran}}$, $\mathbf{X}'_{\text{lap}}$, $\mathbf{X}'_{\text{const}}$, and $\mathbf{X}'_{\text{free}}$ with the true validation image $\mathbf{X}'_{\text{true}}$, we could see if the learned kernels work for reconstructing other images of the same type.

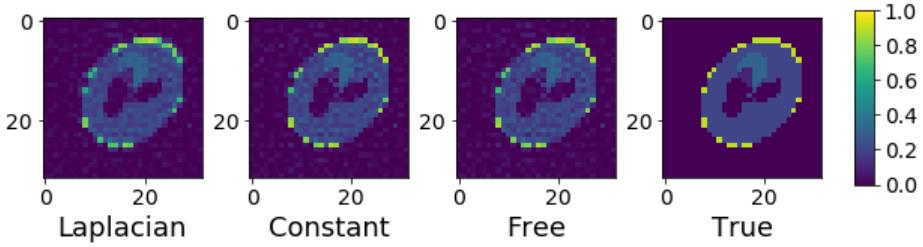
3.2 Tomographic Shepp-Logan Phantom Reconstruction

3.2.1 Single Shepp-Logan Phantoms

Our first experiments deal with the Shepp-Logan phantoms, which is the standard testing images of image reconstruction problems [8]. We first start with reconstructing a single image. For the training set, we generate a random 32×32 Shepp-Logan phantom $\mathbf{X}_{\text{true}}$ and a tomographic operator $\mathbf{A}$. From these, we generate the data $\mathbf{B} = \mathbf{AX} + \mathbf{N}$ with noise $\mathbf{N}$. Similarly, for the validation set, we generate another random 32×32 Shepp-Logan phantom $\mathbf{X}'_{\text{true}}$ and tomographic operator $\mathbf{A}'$, from which we build the data $\mathbf{B}'$. Following the method in section 3.1



(a) Training Results



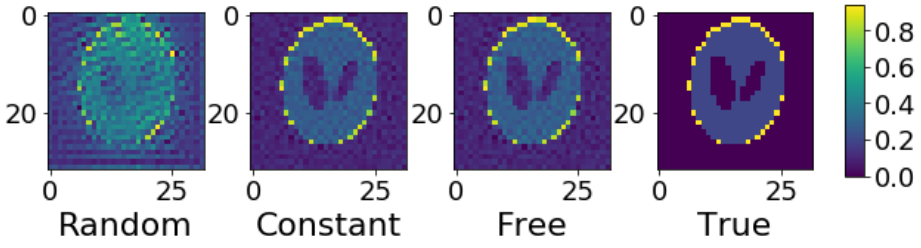
(b) Validation Results

Figure 3.2: Results for tomographic reconstruction of a single 32×32 Shepp-Logan phantom with the initial guess $\mathbf{K}_{\text{lap}}$. Images share a common color bar.

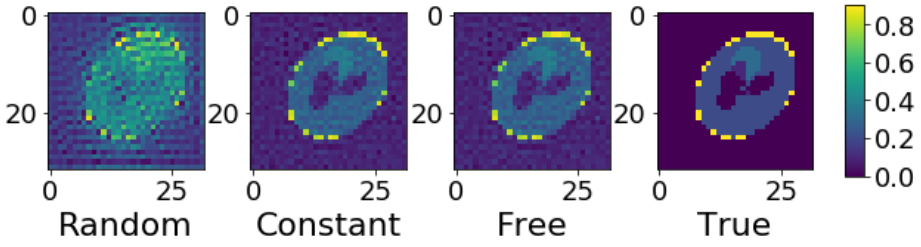
we obtain Figure 3.2 as a result.

By comparing the reconstructed images with the true image in Figure 3.2a, we could see that the learned kernels indeed reconstruct the image in a slightly better way than the un-trained kernel. Also, comparing Figure 3.2b with Figure 3.2a, we could see that the kernels reconstruct the validation image in a manner similar to how they reconstruct the training image. Thus, we could confirm the validity of VNCG training for tomographic reconstruction of single Shepp-Logan phantoms.

However, by comparing the reconstructed images with the true image in Figure 3.2a, we could see that although the learned kernels work well to reconstruct the pattern of the image, the images only show slight differences between trained results and un-trained results. This is caused by the fact that the Laplacian $\mathbf{K}_{\text{lap}}$



(a) Training Results



(b) Validation Results

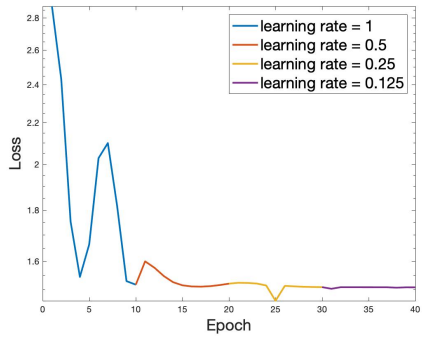
Figure 3.3: Results for tomographic reconstruction of a single 32×32 Shepp-Logan phantom with random initial guess $\mathbf{K}_{\text{ran}}$. The images share a common color bar.

is a good enough parameter for reconstruction so that there is not so much improvement space for training VNCG.

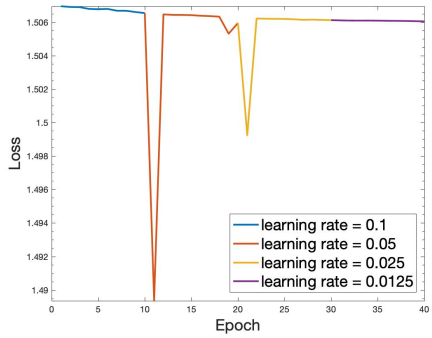
To further understand the performance of VNCG, we take $\mathbf{K}_{\text{ran}}$ as the initial guess to train VNCG for reconstructing with the same true images and generate the results in Figure 3.3.

Now we can see from Figure 3.3 that $\mathbf{K}_{\text{ran}}$ is a "bad" convolution kernel for our reconstruction algorithm, which provides enough improvement space for VNCG. And the learned kernel $\mathbf{K}_{\text{const}}$ now gives a more visibly better reconstruction than the initial guess $\mathbf{K}_{\text{ran}}$. However, doubts still exist for the necessity of allowing different kernels in each iteration of CG in this problem.

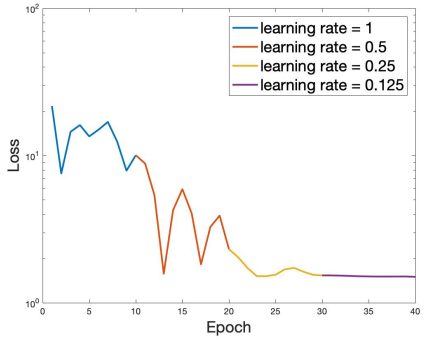
To quantify the performance of VNCG, we want to show the loss with respect



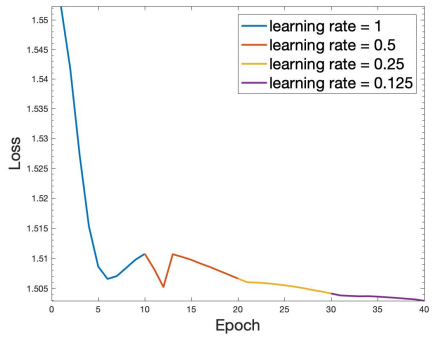
(a) From $\mathbf{K}_{\text{lap}}$ to $\mathbf{K}_{\text{const}}$



(b) From $\mathbf{K}_{\text{const}}$ to $\mathbf{K}_{\text{free}}$



(c) From $\mathbf{K}_{\text{ran}}$ to $\mathbf{K}_{\text{const}}$



(d) From $\mathbf{K}_{\text{const}}$ to $\mathbf{K}_{\text{free}}$

Figure 3.4: Loss v.s. Epoch in training a single 32×32 Shepp-Logan phantom. Row 1 shows training with manual initial guess $\mathbf{K}_{\text{lap}}$; Row 2 shows training with random initial guess $\mathbf{K}_{\text{ran}}$. Column 1 goes from un-trained kernel to trained kernel; Column 2 goes from constant kernel to free kernels.

to the epoch number of our learning process. Also, since we learn the convolution kernels by adjusting the learning rate of SGD function in our experiment, we want to show the relationship between the loss and the learning rate. Thus, we generate Figure 3.4.

Comparison between Figure 3.4a and Figure 3.4c justifies our assumption that $\mathbf{K}_{\text{lap}}$ is a much better convolution kernel than $\mathbf{K}_{\text{ran}}$ for our reconstruction algorithm. In Figure 3.4a, the loss decreases by around 50% in training, while it drops

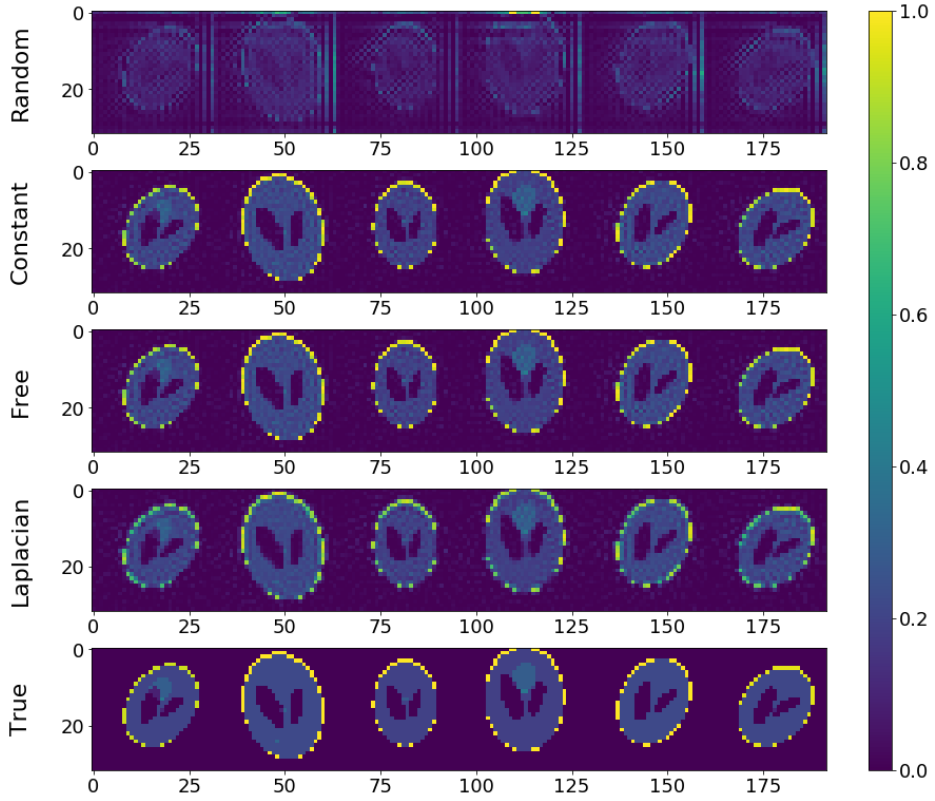
by around 90% in Figure 3.4c. Although the improvement percentage is significantly different, the two learning processes both end up in convergence to a loss of around 1. From this, we could say that the learning processes from un-trained kernel to constant kernel almost reach the minimum loss of our learning problem. Therefore, there is not much improvement space for allowing free kernels. If we look at Figure 3.4b and Figure 3.4d, we could easily find out that the two learning processes do not result in making a big difference in value of loss. Another possible explanation for not having big improvement by allowing free kernels is the optimizer we are using might not be a good choice to learn free kernels.

Then we want to look at the relationship between loss and learning rates. Since the learning process from constant kernel to free kernels does not generate big differences in loss, we only look at Figure 3.4a and Figure 3.4c. As shown in these two images, when we decrease the learning rate of our learning algorithm by half, the loss eventually decreases by about half after the learning process until it reaches an approximate minimum of the optimizing learning problem.

3.2.2 Shepp-Logan Phantom Sets

Now we want to generalize this experiment to reconstruction of multiple phantoms at a same time. We generate our data in the same way as we do for reconstructing single images in 3.2.1. The only difference is that the true images $\mathbf{X}_{\text{true}}$ and $\mathbf{X}'_{\text{true}}$, instead of single random 32×32 Shepp-Logan phantoms, are now random sets of 32×32 Shepp-Logan phantoms.

Since we already show the Laplacian $\mathbf{K}_{\text{lap}}$ is an effective regularization operator, we want to see if VNCG could learn a similarly effective kernel as $\mathbf{K}_{\text{lap}}$ from a



(a) Training Result

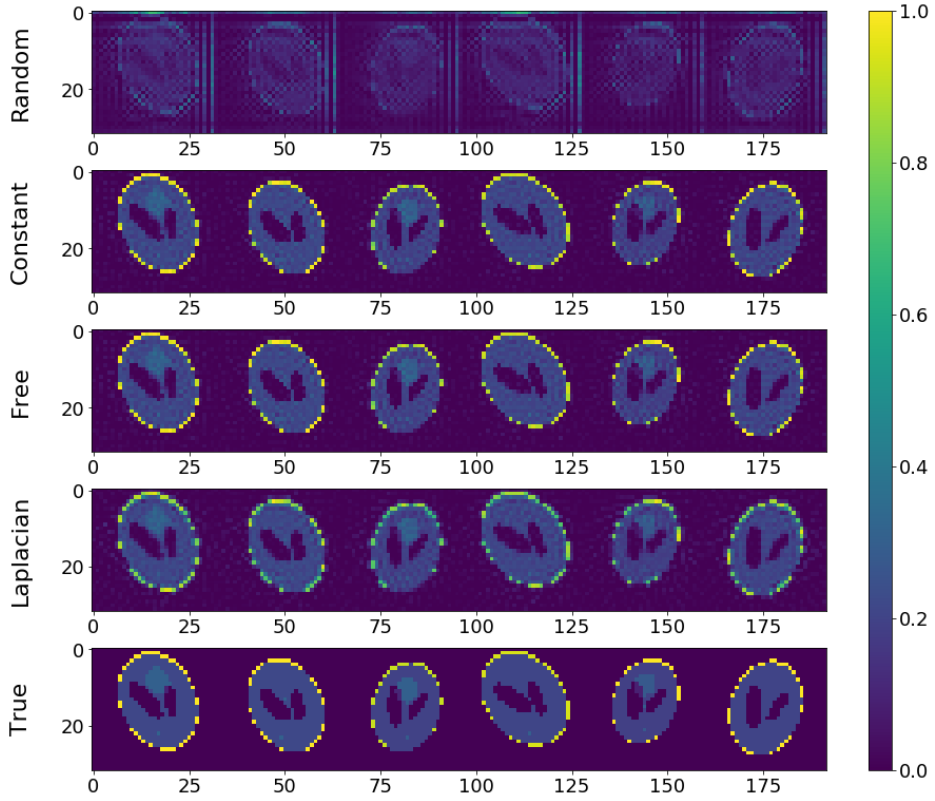
Figure 3.5: Performance of variational network for tomographic reconstruction of 32×32 Shepp-Logan phantom set. Images share a common color bar.

completely ineffective kernel $\mathbf{K}_{\text{ran}}$. We obtain Figure 3.5 for reconstructed images and Figure 3.6 for loss analysis.

Experiments on Shepp-Logan phantom sets give similar results as the one on single phantoms in Figure 3.2. Thus, we could have the following conclusions for this experiment of tomographic Shepp-Logan phantom reconstruction:

1. The Laplacian $\mathbf{K}_{\text{lap}}$ is an effective regularization operator for this problem.

2. Taking a completely ineffective kernel $\mathbf{K}_{\text{ran}}$ as initial guess, VNCG is able



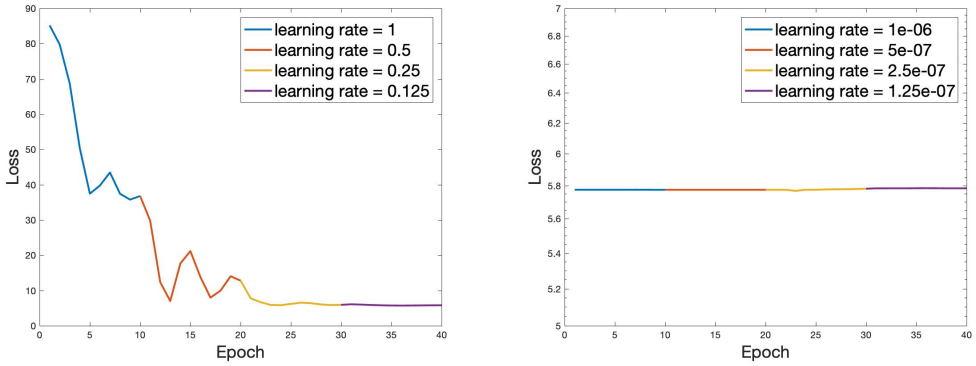
(b) Validation Result

Figure 3.5: Performance of VNCG for tomographic reconstruction of 32×32 Shepp-Logan phantom set (continued). Images share a common color bar.

to learn a constant kernel that is at least as effective as the Laplacian $\mathbf{K}_{\text{lap}}$.

3. We do not see big improvement by allowing different kernels in different iterations. There are some possible explanations for this phenomenon: there might be no necessity of allowing different kernels in solving this problem, or GD is not a good optimizer for learning free convolution kernels, or we do not have a good enough learning problem.

4. Keeping the momentum of the learning algorithm constant, if we decrease



(a) From $\mathbf{K}_{\text{ran}}$ to $\mathbf{K}_{\text{const}}$ **(b)** From $\mathbf{K}_{\text{const}}$ to $\mathbf{K}_{\text{free}}$

Figure 3.6: Convergence plots for training VNCG for tomographic reconstruction of 32×32 Shepp-Logan phantom set.

the learning rate by half, the loss will eventually also decrease by approximately half if it does not reach the approximated optimal value.

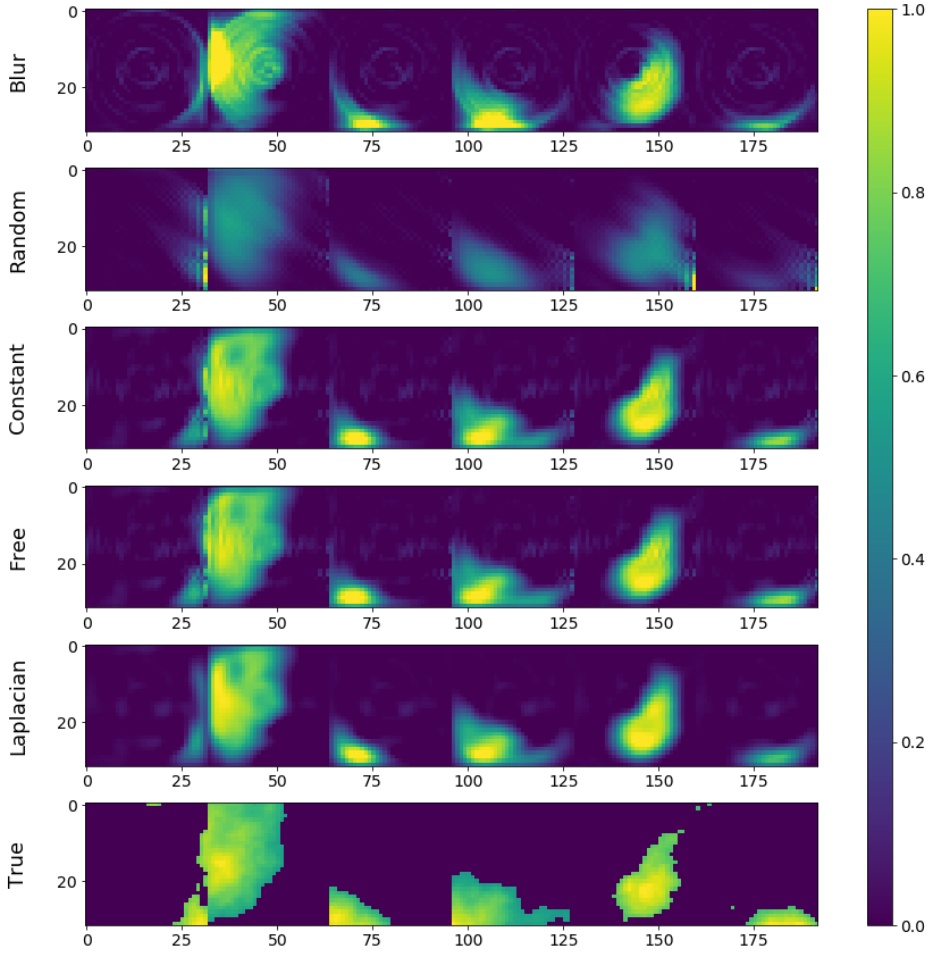
Based on the conclusions above, we will continue to look at other situations in sections 3.3 and 3.4.

3.3 General Image Deblurring

3.3.1 Moderate Blurring

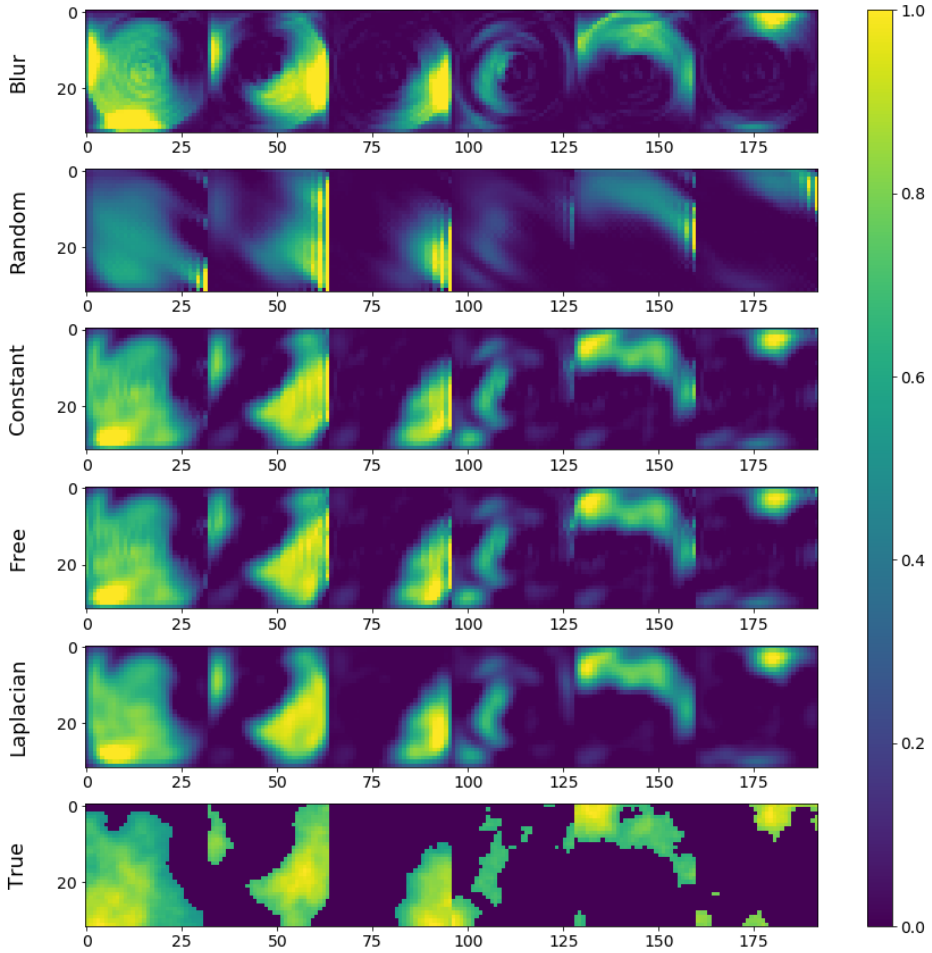
In the second experiment, we want to see how VNCG works with reconstruction from a blurred general image. We first start with images blurred moderately. For the training set, we generate a random 32×32 image $\mathbf{X}_{\text{true}}$ and a rotational blurring operator $\mathbf{A}_{\text{moderate}}$, which moderately blurs the true image. Then we generate the data $\mathbf{B}_{\text{moderate}}$ for some noise $\mathbf{N}$ as in section 3.2. Similarly, we generate $\mathbf{A}'_{\text{moderate}}$, $\mathbf{X}'_{\text{true}}$ and $\mathbf{B}'_{\text{moderate}}$ for the validation set. In training VNCG,

we use $\mathbf{K}_{\text{ran}}$ as our initial guess and see if the learned results is better than the Laplacian $\mathbf{K}_{\text{lap}}$. Besides the reconstructed images and the true images, we also include the blurred images in Figure 3.7 for comparison.



(a) Training Result

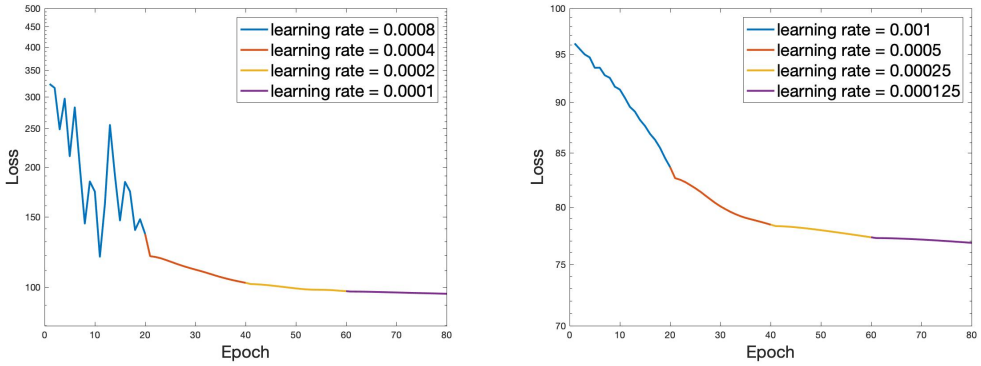
Figure 3.7: Performance of VNCG for deblurring moderately burred 32×32 general images. Images share a common color bar.



(b) Validation Result

Figure 3.7: Performance of VNCG for deblurring moderately burred 32×32 general images (continued). Images share a common color bar.

The performance of VNCG shown in Figure 3.7 reveals the same information we got for tomographic reconstruction: $\mathbf{K}_{\text{ran}}$ serves as a bad convolution kernel for this problem, while $\mathbf{K}_{\text{lap}}$ is an effective operator. Also, the reconstructed images $\mathbf{K}_{\text{const}}$ and $\mathbf{K}_{\text{free}}$ appear to be as effective as the Laplacian $\mathbf{K}_{\text{lap}}$. One thing noticeable from the reconstructed images is that when the image has non-



(a) From $\mathbf{K}_{\text{ran}}$ to $\mathbf{K}_{\text{const}}$ **(b)** From $\mathbf{K}_{\text{const}}$ to $\mathbf{K}_{\text{free}}$

Figure 3.8: Convergence plots of learning processes for deblurring moderately blurred 32×32 general images.

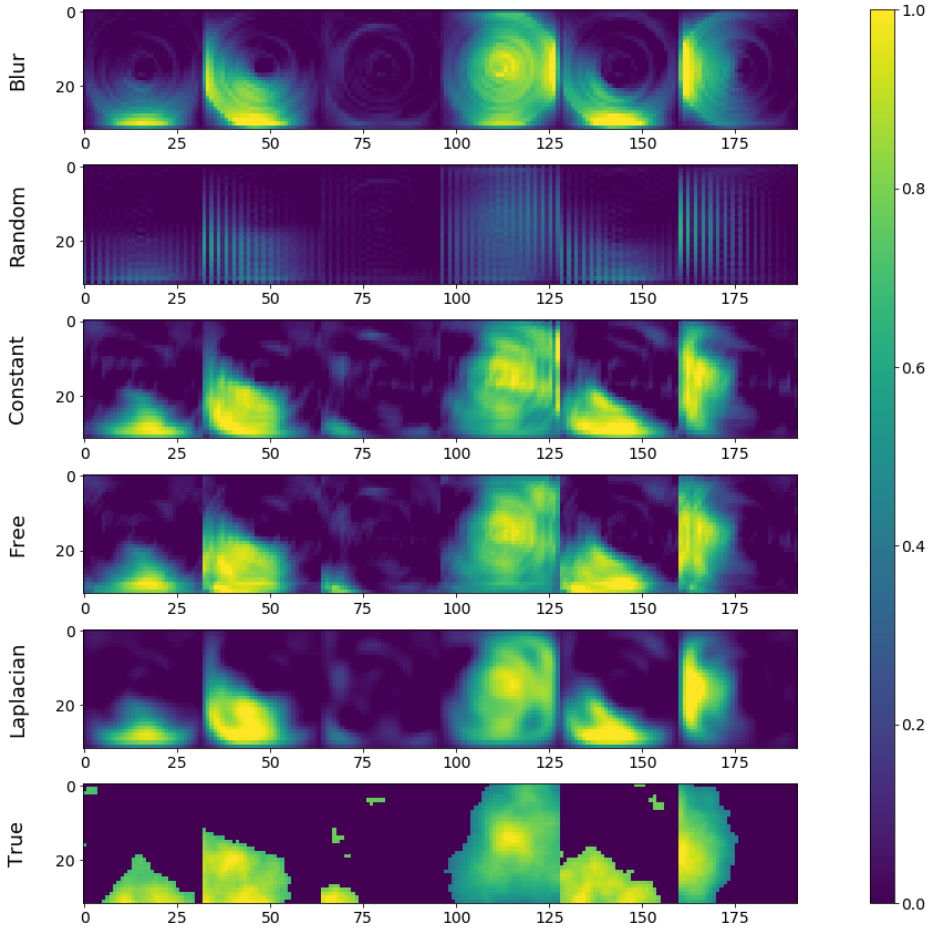
zero boundaries as in Figure 3.7a, the trained kernels does better in reconstruct the boundaries than the Laplacian when we use the zero boundary conditions.

As shown in Figure 3.8a, the loss drops from about 350 to about 100, which is a 70% improvement in terms of loss. Also, in Figure 3.8b, the loss drops by around 10%. In this case, we can say that although the improvement is still hard to be recognized from the reconstructed images, the convergence plots show that allowing different kernels in each iteration of CG indeed provide better regularization operators in terms of loss, which is a difference between this image deblurring problem and the tomographic Shepp-Logan phantom problem.

3.3.2 Severe Blurring

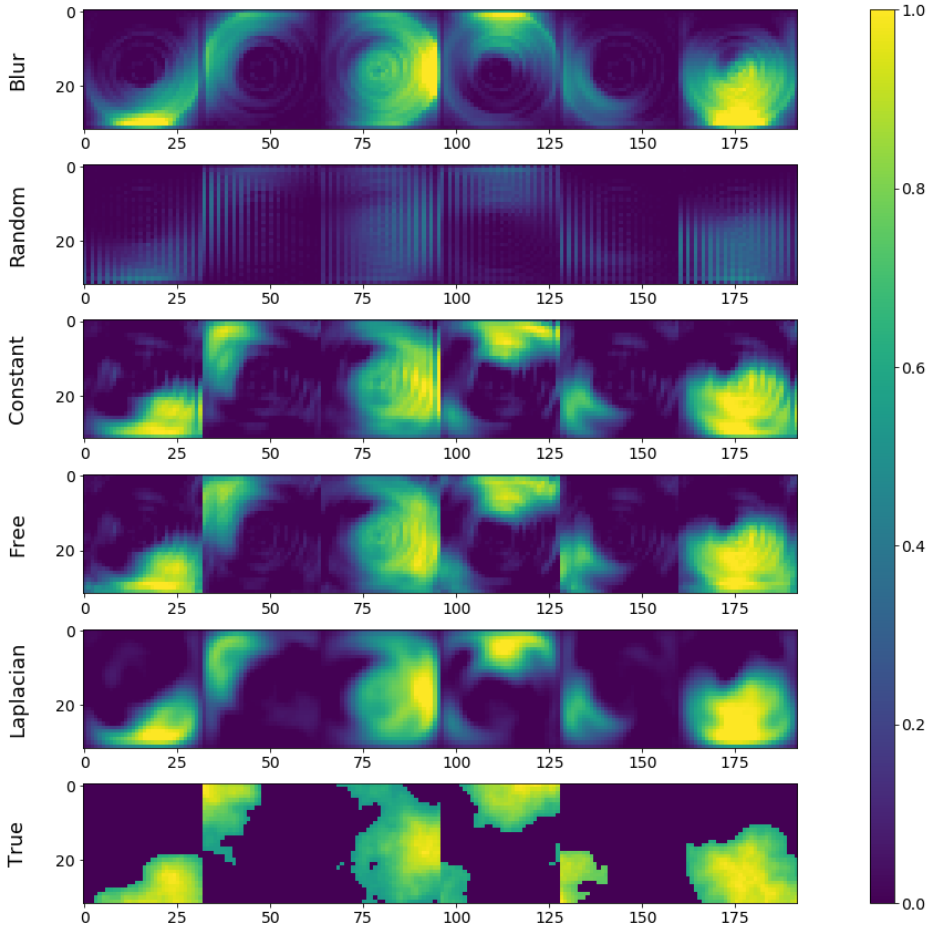
Now we want to increase the level of blurring. We generate our data in the same way as we do for moderate blurring in 3.3.1. The only difference is that we replace the blurring operator $\mathbf{A}_{\text{moderate}}$ with $\mathbf{A}_{\text{severe}}$, which blur the images severely.

Therefore, the blurred image is now denoted $\mathbf{B}_{\text{severe}}$. We train VNCG with initial guess $\mathbf{K}_{\text{ran}}$ and display reconstructed images together with the blurred images and the images reconstructed by $\mathbf{K}_{\text{lap}}$ in Figure 3.9.



(a) Training Result

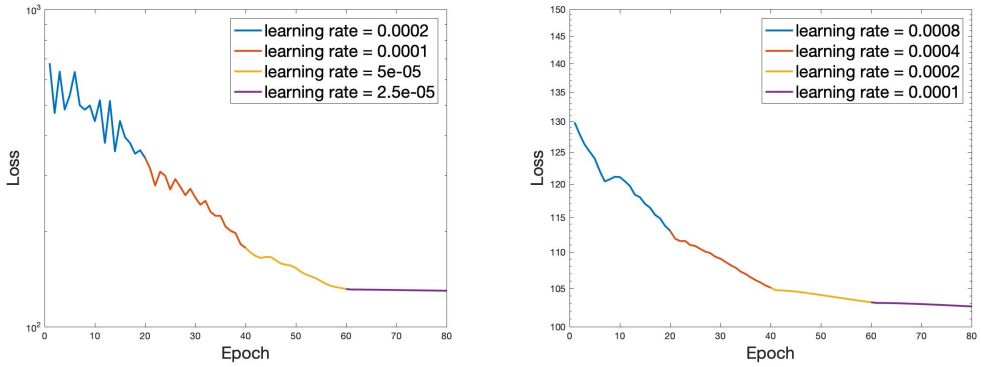
Figure 3.9: Performance of VNCG for deblurring severely burred 32×32 general images. Images share a common color bar.



(b) Validation Result

Figure 3.9: Performance of VNCG for deblurring severely burred 32×32 general images (continued). Images share a common color bar.

Figure 3.9 again shows the improvement brought by training VNCG. Also, from Figure 3.10 we can see that, in the training process to learn $\mathbf{K}_{\text{const}}$ from $\mathbf{K}_{\text{ran}}$, the loss drops by over 75% from about 800 to about 150. Moreover, in the training process to learn $\mathbf{K}_{\text{free}}$ from $\mathbf{K}_{\text{const}}$, the loss drops by around 20%. This gives us a similar result as what we obtained for moderately blurred images in



(a) From $\mathbf{K}_{\text{ran}}$ to $\mathbf{K}_{\text{const}}$ **(b)** From $\mathbf{K}_{\text{const}}$ to $\mathbf{K}_{\text{free}}$

Figure 3.10: Convergence plots for deblurring severely blurred 32×32 general images.

Figure 3.8. However, one thing different between Figure 3.8 and 3.10 is the value of loss, which indicates that the the reconstruction is better for moderately blurred images than for severely blurred images.

Based on the results we get, we could draw the following conclusions for this experiment of deblurring general images:

1. Just like in tomographic reconstruction, $\mathbf{K}_{\text{lap}}$ is an effective convolution kernel for reconstruction.

2. From the convergence plots, we could find that by training VNCG, we can learn a constant convolution kernel nearly as effective as the Laplacian $\mathbf{K}_{\text{lap}}$. Moreover, when the image has non-zero pixels on its boundaries, the learned kernels does better than the Laplacian in reconstructing the boundary when we use trivial zero boundary conditions. Also, unlike in tomographic reconstruction of Shepp-Logan phantoms, allowing free convolution kernels provides an improvement of about 10% in loss. However, compared to the

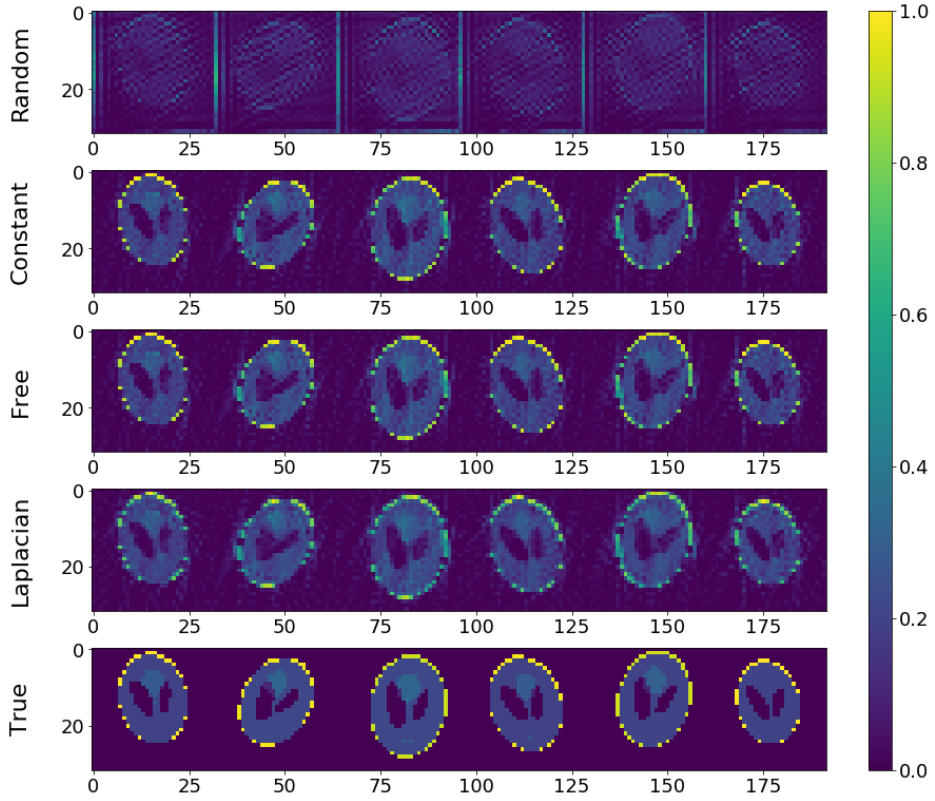
norm of the true image, the improvement is still relatively small. Thus, we still need further experiments to prove necessity of allowing different kernels in each iteration.

3. Performance of VNCG depends on the level of blurring: when the images are severely blurred, the resulting kernels can not deblur as well as they do when the images are blurred in a medium level in terms of the loss.

3.4 Undersampled Tomography

An extension we can make for experiments in section 3.2.1 is to conduct experiments on undersampled tomographic reconstruction. We generate our data sets similarly to what we do in section 3.2.2. However, after we generate our tomographic operator $\mathbf{A}$, we take only the first 80% of columns of it, and therefore our data $\mathbf{B}$ also decrease 20% of its columns. The validation data set is generated in the same way. By training VNCG with initial guess $\mathbf{K}_{\text{ran}}$, we compare the results with images reconstructed by the Laplacian $\mathbf{K}_{\text{lap}}$ in Figure 3.11. Also we generate the loss in Figure 3.12.

The reconstruction performance of the kernels is similar to what we find in section 3.2. However, one thing noticeable for undersampled tomography is that although the reconstructions behave well when we look at the loss for convergence, the pattern of the true image is not preserved as well as how it is preserved for regular tomographic reconstruction in Figure 3.5, and the minimum value of loss is much greater than what we have in Figure 3.6. This indicates that the undersampling level of tomography relates to the performance of VNCG in reconstruction.

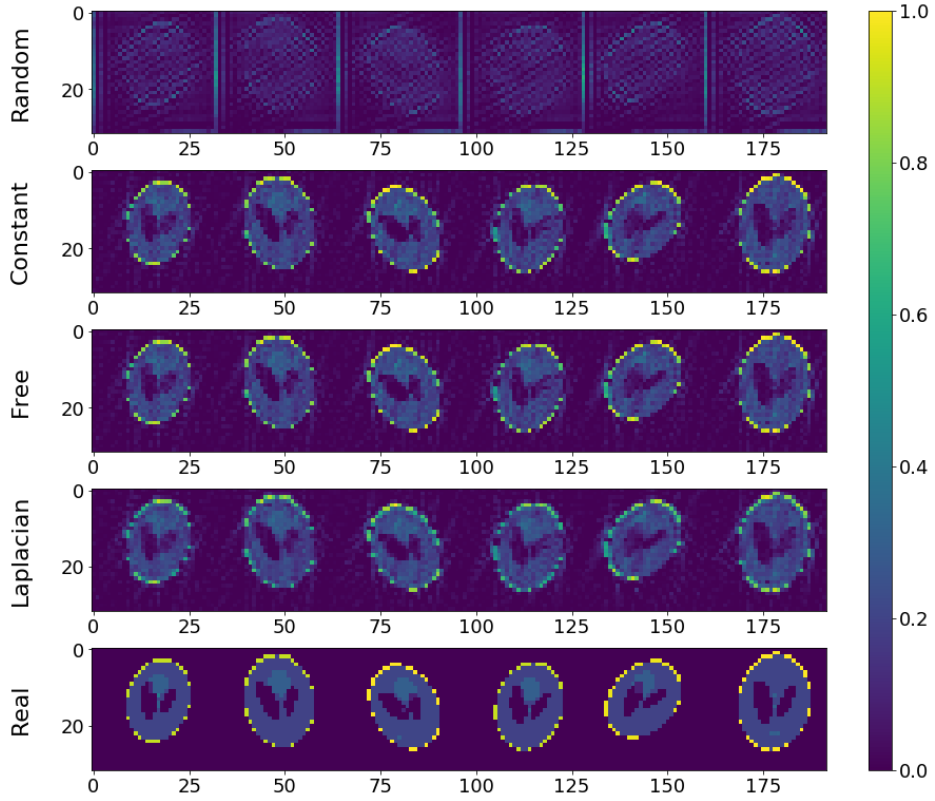


(a) Training Result

Figure 3.11: Performance of VNCG for reconstruction of 20% undersampled tomography. Images share a common color bar.

To find out the maximum undersampling level that VNCG could handle, we train VNCG for different undersampling levels. Since we already show in section 3.2.1 that allowing free kernels does not improve the reconstruction performance, we only look at the performance of the learned constant kernel $\mathbf{K}_{\text{const}}$. We decrease size of $\mathbf{A}$ by 10% each time, and compare the reconstructed images in Figure 3.13.

As shown in Figure 3.13, although some of the phantoms are reconstructed in



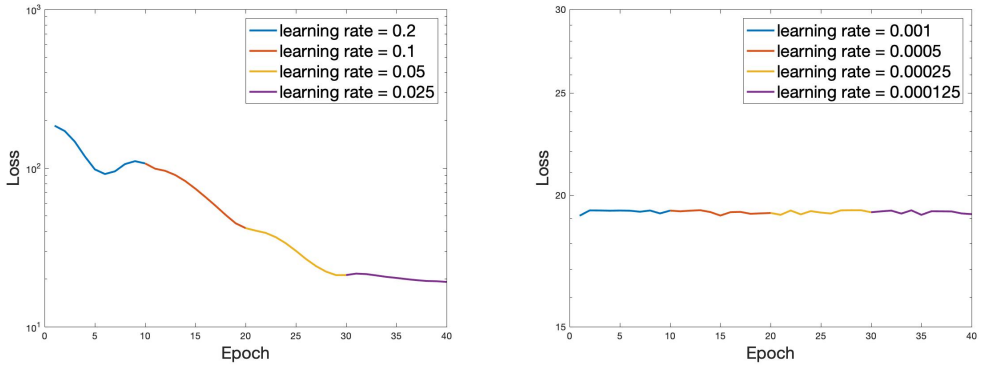
(b) Validation Result

Figure 3.11: Performance of VNCG for reconstruction of 20% undersampled tomography (continued). Images share a common color bar.

an acceptable manner for all undersampling levels, the original pattern is hardly recognizable for some images when the undersampling level is above 40%. Thus, we would make the following conclusions for this experiment:

1. Although training VNCG indeed provides better reconstruction for undersampled tomographic reconstruction, it does not do as well as when it is not undersampled.

2. The performance of VNCG is dependent on the undersampling level. Ac-



(a) From $\mathbf{K}_{\text{ran}}$ to $\mathbf{K}_{\text{const}}$ **(b)** From $\mathbf{K}_{\text{const}}$ to $\mathbf{K}_{\text{free}}$

Figure 3.12: Convergence plots of learning processes for reconstruction of 20% undersampled tomography.

cording to our experiments, VNCG is able to handle undersampling level of at most 40%.

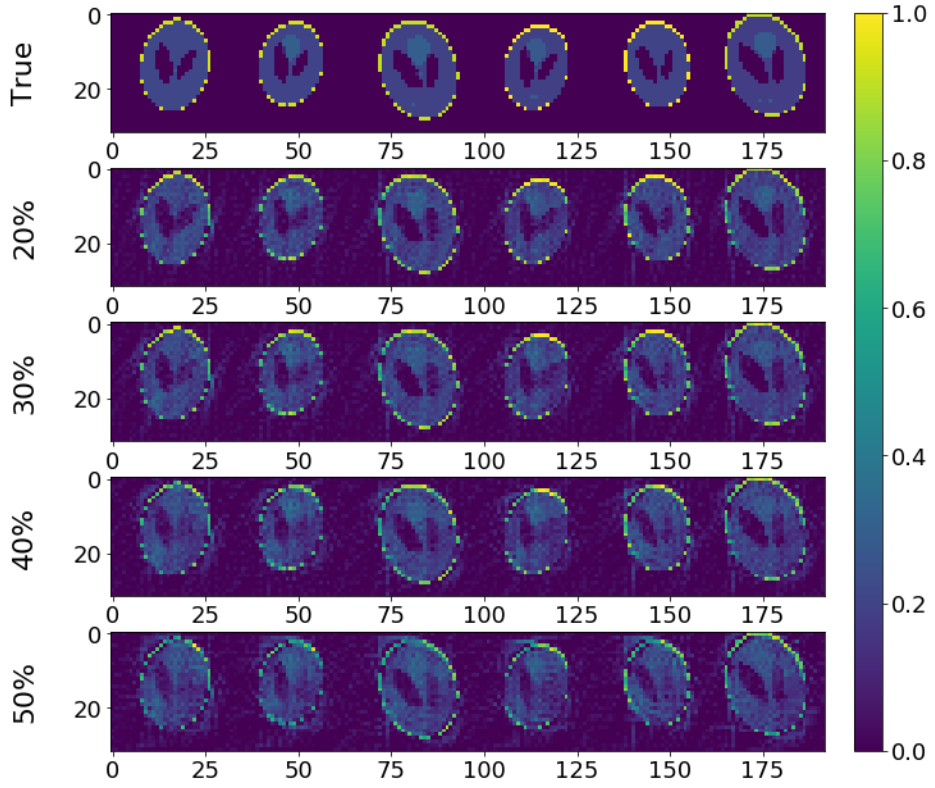


Figure 3.13: Reconstruction results for different undersampling levels.

Chapter 4

Summary and Conclusion

Overall, we presented in this thesis a trainable Conjugate Gradient method (CG) that we call VNCG. In VNCG, we followed the framework of variational networks (VN) to unroll and train a CG method in a fixed number of iterations. Then we conducted numerical experiments in which we considered linear inverse problems and trained a convolution stencil that represents the regularization operator in a Tikhonov form. We attempted two strategies for regularization: using a constant stencil for all iterations or a more flexible approch that assigns different stencils to each iteration.

Based on the results of numerical experiments in Chapter 3, we could make the following conclusions for VNCG:

1. In general, by training VNCG, we could learn convolution kernels that provide better reconstructions than un-trained kernels.

2. For tomographic Shepp-Logan phantom reconstruction, the Laplacian $\mathbf{K}_{\text{lap}}$ serves as an effective regularization operator. Taking a randomly chosen,

ineffective operator as the initial guess, training VNCG could give us an operator as effective as $\mathbf{K}_{\text{lap}}$.

When the network is trained with the Gradient Descent method (GD), if we keep the momentum of the learning process to be constant and decrease the learning rate by half, the resulting loss eventually decreases by about half before the optimizer approximately reach the optimal value.

By looking at both the loss and the reconstructed images, we do not see necessity of allowing free kernels to solve these problems, which might results from the optimizer we used for training and the learning problem we have.

3. For deblurring general images, the Laplacian $\mathbf{K}_{\text{lap}}$ still serves as an effective convolution kernel. Training VNCG to learn a constant kernel for deblurring results in kernel(s) almost as effective as $\mathbf{K}_{\text{lap}}$ in terms of the reconstructed images. Also, when the image has non-zero boundary pixels, the learned kernel does better than the Laplacian $\mathbf{K}_{\text{lap}}$ in reconstructing the boundaries under zero boundary conditions.

   Unlike for tomographic Shepp-Logan phantoms, allowing free convolution kernels for image deblurring provides an improvement of around 10% in terms of loss. However, compared with the norm of the true image, the percentage improvement is not sufficient to prove the necessity of allowing different kernels.

   Moreover, the performance of VNCG depends on the blurring level. When the images are severely blurred, the reconstruction performance is not so good as how it is when the images are moderately blurred.

4. For undersampled tomographic reconstruction, the improvement by training is also observed. The reconstruction performance of VNCG relates to the undersampling level. Through experiments, we show that the VNCG is able to handle an undersampling level of at most 40% missing data.

Besides the improvement in reconstructions listed above, experiments also reveal some shortcomings:

1. For tomographic reconstructions, assigning different convolution kernels in each iteration does not make any improvement. While in image deblurring, the improvement is relatively small so that we could not prove the necessity of assigning different kernels in solving these problems. A possible explanation is that we did not use a suitable optimizer to train VNCG for free convolution kernels.

2. For undersampled tomographic reconstruction, VNCG can only handle up to 40% undersampled operators. When we miss more than 40% of the operator, VNCG is not able to preserve the pattern of a Shepp-Logan phantom.

Based on the findings and shortcomings, there are several directions that might lead to future works. One of the directions is to introduce other parameters to our reconstruction algorithm and train them together with the convolution kernel. By introducing new training parameters, the VN could possibly be able to handle more than 40% undersampled tomography. Another possible direction is to design an appropriate training method to train VNCG for different convolution kernels. Besides, VNCG can also be applied to real medical data, such as the MRI data. In this case, the practicability of this method is examined.

Appendix A

Main Notation

$\mathbf{A}$	some matrix, operator
$\mathbf{X}$	some matrix, image
$\mathbf{B}$	some matrix
$\mathbf{K}$	some matrix, convolution kernel(s)
$\mathbf{R}$	some matrix, residual in CG
$\mathbf{P}$	some matrix, moving direction in CG
$\mathbf{L}$	some matrix, convolution operator
$\mathbf{a}$	some vector
$\mathbf{b}$	some vector
α	some scalar
β	some scalar
N	some scalar
λ	some scalar
k	some scalar

Appendix B

Abbreviations

CG	Conjugate Gradient
VN	Variational Network
GD	Gradient Descent
SGD	Stochastic Gradient Descent
VNCG	VN with CG

Appendix C

Python Code

```
import copy
import math
import matplotlib.pyplot as plt
import numpy as np
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.sparse
import scipy.io as sio
import scipy.sparse as sp
from scipy.io import loadmat
```

1 Helper Functions

```
def img2dto4d(X, n, num_image):
    # X: (n*n) * num_image: 2d image
    graph4d = torch.t(X).reshape([num_image,1,n,n])
    graph4d = torch.transpose(graph4d,2,3)
    return graph4d


def conv3x3(X,K):
    """3x3 convolution with padding"""
```

```
    return F.conv2d(X, K, stride=1, padding=1)

def conv3x3T(X,K):
    """3x3 convolution transpose with padding"""
    return F.conv_transpose2d(X, K, stride=1, padding=1)

def conv_mat(X, K, n, num_image):
    # X: (n*n) * num_image
    graph4d = torch.t(X).reshape([num_image,1,n,n])
    graph4d = torch.transpose(graph4d,2,3)
    conv_graph4d = conv3x3T(conv3x3(graph4d,K),K)
    graph2d = torch.transpose(conv_graph4d,2,3).reshape(
                                        [num_image, n*n])
    return torch.t(graph2d)

# load data from mat files, convert to tensors
def load_data(matdata):

    # load matrix generated in MATLAB
    data = loadmat(matdata)
    B = torch.FloatTensor(data['B'])
    X_true = torch.FloatTensor(data['X'])
    A_sparse = data['A'] # sparse

    # construct sparse matrix in torch
    (I, J, V) = sp.find(A_sparse)
    it = torch.LongTensor([I, J])
    vt = torch.FloatTensor(V)
    A = torch.sparse.FloatTensor(it, vt,
            torch.Size([A_sparse.shape[0], A_sparse.shape[1]]))

    return A, B, X_true
```

2 CNN with CG

```
class CG_CNN(nn.Module):

    def forward(self,A,B,K):
```

```
# A: tensor, n^2 * n^2,             operator
# B: tensor, n^2 * num_image,       image
# K: tensor, 1*1*3*3,               for constant kernel
#            20*1*1*3*3,            for free kernel sets

n = int(math.sqrt(A.shape[1]))       # image size: n * n
num_image = B.shape[1]               # num of images
if K.dim() == 4: N = 20              # num of iterations
else: N = K.shape[0]

# CG method
X = torch.zeros(A.shape[1], B.shape[1])
                        # initial guess: X[0] = 0
if K.dim() == 4:        # AX + conv(X)
    AX = conv_mat(X, K, n, num_image)
                                    + torch.spmm(A, X)
else:
    AX = conv_mat(X, K[0], n, num_image)
                                    + torch.spmm(A, X)

P = torch.add(B, -1, AX)         # P[0] = B - AX
R = P

# CG  iteration
for i in range(2,N,1):
    Rinner = torch.sum(torch.mul(R, R), 0)
                        # column-wise inner products
    if K.dim() == 4:    # AP + conv(P)
        AP = conv_mat(P, K, n, num_image)
                                    + torch.spmm(A, P)
    else:
        AP = conv_mat(P, K[i], n, num_image)
                                    + torch.spmm(A, P)
    alpha = torch.div(Rinner, torch.sum(
                                  torch.mul(AP, P), 0))
                        # alpha[i] = Rinner/<AP[i],P[i]>
    X = X + torch.mul(P, alpha)
                        # X[i+1] = X[i] + alpha * P[i]
    R = R - torch.mul(AP, alpha)
                        # R[i+1] = R[i] - alpha * P[i]
```

```
            newRinner = torch.sum(torch.mul(R, R), 0)
                                # inner product of R[i+1]
            beta = torch.div(newRinner, Rinner)
                                # beta[i] = newRinner / Rinner
            P = R + torch.mul(P, beta)
                                # P[i+1] = R[i] + beta * P[i]

        return X
```

3 Loss Function

```
def Loss(X,X_true):
    return torch.norm(X - X_true) ** 2
```

4 Load Data

```
A, B, X_true       = load_data("train.mat") # training
A_v, B_v, X_true_v = load_data("valid.mat") # validation
```

5 Initialize $\mathbf{K_{lap}}$ and $\mathbf{K_{ran}}$

```
K_lap = nn.Parameter(torch.Tensor(1,1,3,3))
K_ran = nn.Parameter(torch.Tensor(1,1,3,3))
data = np.random.randn(1,1,3,3)
data = np.float32(data)
K_ran.data = torch.from_numpy(data)*10
data[0][0] = [[0,-1,0],[-1,4,-1],[0,-1,0]]
K_lap.data = torch.from_numpy(data)
```

6 Initialize $\mathbf{K_{const}}$ and Train

```
K_const = copy.deepcopy(K_lap)

net = CG_CNN_K()

lr_num = 4
epoch_num = 20
loss_const = np.random.randn(lr_num * epoch_num)
```

```
for i in range(lr_num):

    lr = 1/(2**i)
    optimizer = optim.SGD([{'params':K_const}], lr=lr, momentum=0.9)

    # loop over the dataset multiple times
    for epoch in range(epoch_num):

        optimizer.zero_grad()       # zero the parameter gradients
        X    = net(A,B,K_const)     # forward
        loss = Loss(X, X_true)      # backward
        loss.backward()
        optimizer.step()            # optimize
        loss_const[epoch_num * i + epoch] = loss
                                     # record loss
        print('%d \t %.4f' % (epoch + 1, loss))

print('Finished Training, get K_const')
```

7 Initilize K_{free} and Train

```
K_free = nn.Parameter(torch.Tensor(20,1,1,3,3))
data = np.random.randn(20,1,1,3,3)
data = np.float32(data)
const = K_const.detach().numpy()
for i in range(0,20): data[i] = const
K_free.data = torch.from_numpy(data)

lr_num = 4
epoch_num = 20
loss_free = np.random.randn(lr_num * epoch_num)

for i in range(lr_num):

    lr = 1/(2**i)
    optimizer = optim.SGD([{'params':K_free}], lr=lr, momentum=0.9)

    # loop over the dataset multiple times
```

```
    for epoch in range(epoch_num):

        optimizer.zero_grad()       # zero the parameter gradients
        X    = net(A,B,K_free)      # forward
        loss = Loss(X, X_true)      # backward
        loss.backward()
        optimizer.step()            # optimize
        loss_free[epoch_num * i + epoch] = loss
                                    # record loss
        print('%d \t %.4f' % (epoch + 1, loss))

print('Finished Training, get K_free')
```

8 Data Collect and Image Generation

```
n = int(math.sqrt(X_true.shape[0]))
num_image = X_true.shape[1]

# training data
blur         = B.detach().numpy()
true         = X_true.detach().numpy()
output_man   = net(A,B,K_lap).detach().numpy()
output_const = net(A,B,K_const).detach().numpy()
output_free  = net(A,B,K_free).detach().numpy()

I_blur_t  = np.transpose(np.reshape(np.transpose(blur), (-1, n)))
I_true_t  = np.transpose(np.reshape(np.transpose(true), (-1, n)))
I_man_t   = np.transpose(np.reshape(np.transpose(output_man),
                                                    (-1, n)))

I_const_t = np.transpose(np.reshape(np.transpose(output_const),
                                                    (-1, n)))
I_free_t  = np.transpose(np.reshape(np.transpose(output_free),
                                                    (-1, n)))

# validation data
blur_v         = B_v.detach().numpy()
true_v         = X_true_v.detach().numpy()
output_man_v   = net(A_v,B_v,K_lap).detach().numpy()
```

```
output_const_v = net(A_v,B_v,K_const).detach().numpy()
output_free_v  = net(A_v,B_v,K_free).detach().numpy()

I_blur_v  = np.transpose(np.reshape(np.transpose(blur_v),
                                                    (-1, n)))
I_true_v  = np.transpose(np.reshape(np.transpose(true_v),
                                                    (-1, n)))
I_man_v   = np.transpose(np.reshape(np.transpose(output_man_v),
                                                    (-1, n)))
I_const_v = np.transpose(np.reshape(np.transpose(output_const_v),
                                                    (-1, n)))
I_free_v  = np.transpose(np.reshape(np.transpose(output_free_v),
                                                    (-1, n)))
```

Bibliography

[1] S. Gazzola, P. C. Hansen, and J. G. Nagy. Ir tools: a matlab package of iterative regularization methods and large-scale test problems. *Numerical Algorithms*, pages 1–39, 2018. 15

[2] K. Hammernik, T. Klatzer, E. Kobler, M. P. Recht, D. K. Sodickson, T. Pock, and F. Knoll. Learning a variational network for reconstruction of accelerated mri data. *Magnetic resonance in medicine*, 79(6):3055–3071, 2018. 1, 2, 6, 12

[3] M. R. Hestenes and E. Stiefel. Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952. 6

[4] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer Science & Business Media, New York, Dec. 2006. 6

[5] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017. 16, 18

[6] L. Ruthotto, J. Chung, and M. Chung. Optimal experimental design for constrained inverse problems. *arXiv preprint arXiv:1708.04740*, 2017. 15

[7] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Second Edition. SIAM, Philadelphia, Apr. 2003. 6

[8] L. A. Shepp and B. F. Logan. The fourier reconstruction of a head section. *IEEE Transactions on nuclear science*, 21(3):21–43, 1974. 2, 14, 19

[9] I. V. Tetko, D. J. Livingstone, and A. I. Luik. Neural network studies. 1. comparison of overfitting and overtraining. *Journal of chemical information and computer sciences*, 35(5):826–833, 1995. 7, 8