Solving ill-posed inverse problems using iterative deep neural networks

Project of Course : Mathematics of Machine Learning

Instructor : Prof. Labate

Presenters : Arash Goligerdian
Yerbol Palzhanov

Outline

- Introduction
- Solving inverse problems by learned gradient descent
- Implementation and evaluation
- Results
- Discussion and Conclusion
- End

Introduction

- Inverse Problems

Formulation of an Inverse Problems in terms of signal and data as variables:

$$g = \mathrm{T}(f_{true}) + \delta g \quad \text{where signal} \quad f_{true} \in X \quad \text{and data} \quad g \in Y$$

X and Y are topological vector spaces, $T : X \to Y$ (forward operator), δg is a single sample of a Y-valued random variable that represents the noise component of data.

- Classical regularization

A common approach in solving an inverse problem of the above form is to minimize the miss-fit against data. i.e. by minimizing: $f \to L(T(f), g)$ where $L : Y \times Y \to \square$ is a suitable affine transformation of the data log-likelihood (finding a maximum likelihood solution).

- How to avoid over fitting

There are 3 strategies:

1-Approximate inverse 2-Iterative regularisation 3-Regularisation functional

Introduction

- Regularised objective functional

$$\min_{f \in X} [L(T(f), g) + \lambda S(f)]$$

for a fixed $\lambda \geq 0$, where $S : X \to \square$ is a regularisation functional and $S(f) := \|\nabla f\|$

- Machine learning approaches to inverse problems (Supervised learning)
- Machine learning can be phrased as the problem of finding a (non-linear) mapping $T_\Theta^\dagger : Y \to X$ satisfying $T_\Theta^\dagger(g) \approx f_{true}$ whenever data g is related to f_{true} .
- Set of pseudo-inverse operators is parameterized by $\Theta \in Z$ where Z is a suitable parameter space
- Supervised learning : Estimating $\Theta \in Z$ from training data, can be formulated as minimising a loss functional: $L(\Theta) := \mathrm{E}_\mu \left[\left\| T_\Theta^\dagger(g) - f \right\|_X^2 \right]$, where μ is a known probability distribution.
- μ can be chosen by expressing as an analytically conditional density of g given f multiplied with the empirical density of f.

Introduction

- Challenges of variational regularization
- Computational feasibility
- Flexibility in the prior information that can be accounted for
- Choice of regularization parameters
- Several attempts in order to address above challenges
- Fully learned reconstruction : Learning $T_\Theta^\dagger : Y \to X$ from data such that it approximates an inverse of T
- Sequential data and knowledge driven reconstruction: Separating the learned components from a part that encodes some knowledge about the structure of T and the data manifold. Formalising this $T_\Theta^\dagger = B_\Theta \circ A \circ C_\Theta$ where $A : Y \to X$ is a known component that encodes knowledge about the structure of T and operators B_Θ and C_Θ are the learned components.
- Learning for variational reconstruction: The aim is to solve a variational problem by using ML techniques to select the regularisaton parameters.
- Learned iterative reconstruction: Compromising upon the ability to account for knowledge about the IP

Solving inverse problems by learned gradient descent

- Heuristic motivation
- Error functional: $E(f) := \|f - f_{true}\|_X^2$ where $E : X \to \square$
- Therefore we have: $E(f) := \|f - f_{true}\|_X^2 \approx \arg\min_{f \in X}[L(T(f), g) + \lambda S(f)]$
- By assuming the right hand side is Frechet differentiable and convex, A simple gradient descent scheme used to find a minimum:

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

where, assuming a differentiable likelihood and forward operator, we note that:

$$\nabla[L(T(.), g)](f) = [\partial T](f^*)\left(\nabla[L(T(.), g)](T(f))\right) \quad \text{for any } f \in X$$

likewise, Considering left hand side in the same way, we get:

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

- Learned updating operator:

$$\Lambda_\Theta\left(f, \nabla[L(T(.), g)], \lambda\nabla S(f)\right) \approx \nabla E(f) \quad \text{where} \quad \Lambda_\Theta : X \times X \times X \to X \quad \text{for given parameter} \quad \Theta \in Z$$

Solving inverse problems by learned gradient descent

- Algorithm 1 (Partially learned gradient descent scheme)

1: Select an initial guess f_0
2: **for** $i = 1, \dots, I$ **do**
3: $\quad \Delta f_i \leftarrow -\sigma \Lambda_{\Theta}\Big(f_{i-1}, \nabla\big[\mathcal{L}\big(\mathcal{T}(\cdot), g\big)\big](f_{i-i}), \lambda \nabla \mathcal{S}(f_{i-1})\Big)$
4: $\quad f_i \leftarrow f_{i-1} + \Delta f_i$
5: $\mathcal{T}^{\dagger}_{\Theta}(g) \leftarrow f_I$

- Shortcomings of Algorithm 1:

  1-Parameter λ and step length σ are not explicitly chosen 2-Convergence rate

- Algorithm 2 (Partially learned gradient descent scheme)
- Updated learned updating operator: $\Lambda_{\Theta} : X^M \times X \times X \times X \to X^M \times X$ where persistent memory $s \in X^M$

Solving inverse problems by learned gradient descent

$f_0 \leftarrow \mathcal{T}^\dagger(g)$.
Initialize "memory" $s_0 \in X^M$.
for $i = 1, \ldots, I$ **do**
    $(s_i, \Delta f_i) \leftarrow \Lambda_\Theta\left(s_{i-1}, f_{i-1}, \nabla\left[\mathcal{L}(\mathcal{T}(\cdot), g)\right](f_{i-1}), \nabla\mathcal{S}(f_{i-1})\right)$
    $f_i \leftarrow f_{i-1} + \Delta f_i$
$\mathcal{T}^\dagger_\Theta(g) \leftarrow f_I$

- Parametrising the learned updating operators
- The goal here is to specify the class of learned updating operators parametrised $\Theta \in Z$
- Following the paradigm in (deep) neural networks, we define:

  Family of affine(linear) operators $W_{w_n,b_n} : X^{c_n-1} \to X^{c_n}$ *for* $n = 0,\ldots,N$, $w_n : X^{c_n-1} \to X^{c_n}$ $(weights)$, $b_n \in X^{c_n}$ $(biases)$

  Family of non-linear operators $A_n : X^{c_n} \to X^{c_n}$ *for* $n = 0,\ldots,N$ (response function)

  Parametrised family of learned $\Lambda_\Theta := (A_N \circ W_{w_N,b_N}) \circ \ldots \circ (A_1 \circ W_{w_1,b_1})$

  where N is depth of NN, c_n number of channels in the n:th layer, $\Theta := ((w_N, b_N), \ldots, (w_1, b_1))$

Solving inverse problems by learned gradient descent

- Choice of affine and non-linear operator families
- Narrowing down the generative models for the operator families:

$$W_{w_n,b_n} = \left(W^1_{w_n,b_n},\ldots,W^{c_n}_{w_n,b_n}\right) \ , \ W^l_{w_n,b_n} := X^{c_{n-1}} \to X \quad for\ l = 1,\ldots,c_n$$

where the components $W^l_{w_n,b_n}$ represents the affine transformation for the l:channel in n:th layer, By Linearity:

$$W^l_{w_n,b_n}(f_1,\ldots,f_{c_n-1}) = b^l_n + \sum_{j=1}^{c_{n-1}} w^{j,l}_n(f_j)$$

$w^{j,l}_n : X \to X$ is a channel-wise linear operator

- Using CNNs to learn the learned updating operator
- By using convolution operators, affine operators can be written: $W^l_{w_n,b_n}(f_1,\ldots,f_{c_n-1}) = b^l_n + \sum_{j=1}^{c_{n-1}} w^{j,l}_n * f_j$

where $b^l_n \in \square$ and w_n as a 'matrix' of convolution kernels $w^{j,l}_n \in X$

- Non-linear response function can be chosen as rectified linear unit: $relu(x) = \begin{cases} x \text{ if } x > 0 \\ 0 \text{ else} \end{cases}$

Solving inverse problems by learned gradient descent

- The partially learned gradient descent algorithm
- A number of hyper-parameters needs to be chosen prior to learning:

  1-Number of layers N 2-Number of channels $c_1,\ldots,c_{N-1}$ 3-Number of iteration I 4-Size of memory M
- In examples, Above hyper-parameters are valued as:

  1-Weights w_i be represented by 3×3 pixel convolutions and N=3 2-Number of convolution in each layer is selected $m_1 = 32$ and $m_2 = 32$ 3-I=10 4-M =5
- Algorithm 3 (Partially learned gradient descent scheme)

$s_0 \leftarrow 0$
$f_0 \leftarrow \mathcal{T}^\dagger(g)$
for $i = 1, \ldots, I$ **do**
  $u_i^1 \leftarrow \left(f_{i-1}, s_{i-1}, \nabla\left[\mathcal{L}(\mathcal{T}(\cdot), g)\right](f_{i-1}), \nabla\mathcal{S}(f_{i-1})\right)$
  $u_i^2 \leftarrow \mathrm{relu}(\mathcal{W}_{w_1,b_1}(u_i^1))$
  $u_i^3 \leftarrow \mathrm{relu}(\mathcal{W}_{w_2,b_2}(u_i^2))$
  $(u_i^4, \Delta f_i) \leftarrow \mathcal{W}_{w_3,b_3}(u_i^3)$
  $s_i \leftarrow \mathrm{relu}(u_i^4)$
  $f_i \leftarrow f_{i-1} + \Delta f_i$
$\mathcal{T}_{\Theta}^\dagger(g) \leftarrow f_I$

Implementation and evaluation

- Forward operator is expressible in terms ray transform $\mathrm{P}: X \to Y$ integrating the signal over a set of lines M, Hence elements in Y are functions on lines:

$$\mathrm{P}(f)(\ell) = \int_{\ell} f(x)dx \quad \text{for } \ell \in \mathrm{M}$$

- CT simulations from two particular types of phantoms are considered as training data with different forward operators and noise models:

  1-Ellipses : The log-likelihood was selected as $L(\cdot, g) := \frac{1}{2}\|\cdot - g\|_Y^2$ which implies $\nabla[L(\mathrm{P}(.), g](f) = \mathrm{P}^*(\mathrm{P}(f) - g)$

  2-Heads : Non-linear forward operator given by $T(f)(\ell) = \lambda \exp(-\mu \mathrm{P}(\mathrm{f})(\ell))$

  The log-likelihood is given by Kullback-Leibler divergence and data discrepancy becomes:

$$L(T(f), g) := \int_M \left(T(f)(\ell) + g(\ell) \log\left(\frac{g(\ell)}{T(f)(\ell)}\right) \right) d\ell$$

  which implies that $\nabla[L(T(.), g)](f) = [\partial T(f)]^* \left(1.0 - \frac{g}{T(f)}\right)$

  After some simplifications expression for the gradient is: $\nabla[L(T(.), g)](f) = -\mu P^*(T(f) - g) \quad \text{for } f \in X$

  For both above cases regulariser was selected as the Dirichlet energy: $S(f) := \frac{1}{2}\|\nabla f\|_2^2 \Rightarrow \nabla S(f) = \nabla^*(\nabla f)$

Implementation and evaluation

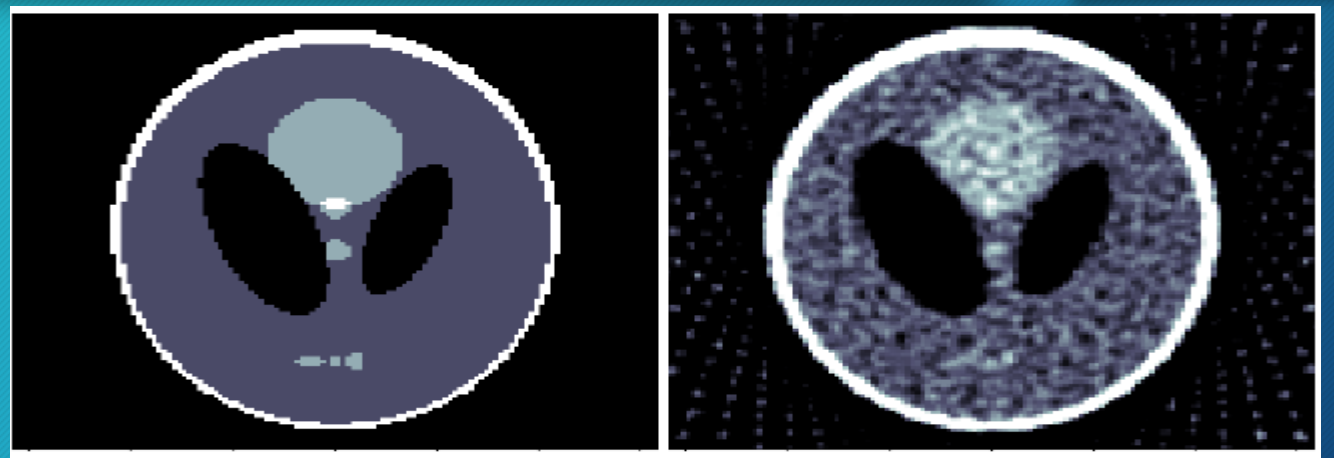
- Implementation
- The methods are implemented in Python using ODL. The NN layers and training were implemented using Tensorflow.
- 1 GB of GPU is required for storing ray transform used for the heads dataset as sparse matrix of floating numbers.
- Parameters Θ is trained using RMSPropOptimizer optimizer in Tensorflow.
- Performance of partially learned(PL) iterative algorithm against the FBP algorithm and TV regularization is compared.
- All 3 algorithms (regularisation parameter) maximise peak signal to noise ratio (PSNR).
- For the ellipse dataset, the evaluation is performed on the modified Shepp-Logan phantom.
- For the heads dataset a slice through the nasal region is used.
- Results
- Reconstructions of the PL algorithm with the FBP and TV reconstructions for both the ellipse and head datasets were compared, the PSNR and runtime were computed. Results are given in the following figures and table.
- Impact of including gradient mappings in the PL gradient scheme was investigated and shown in last figure.

Results

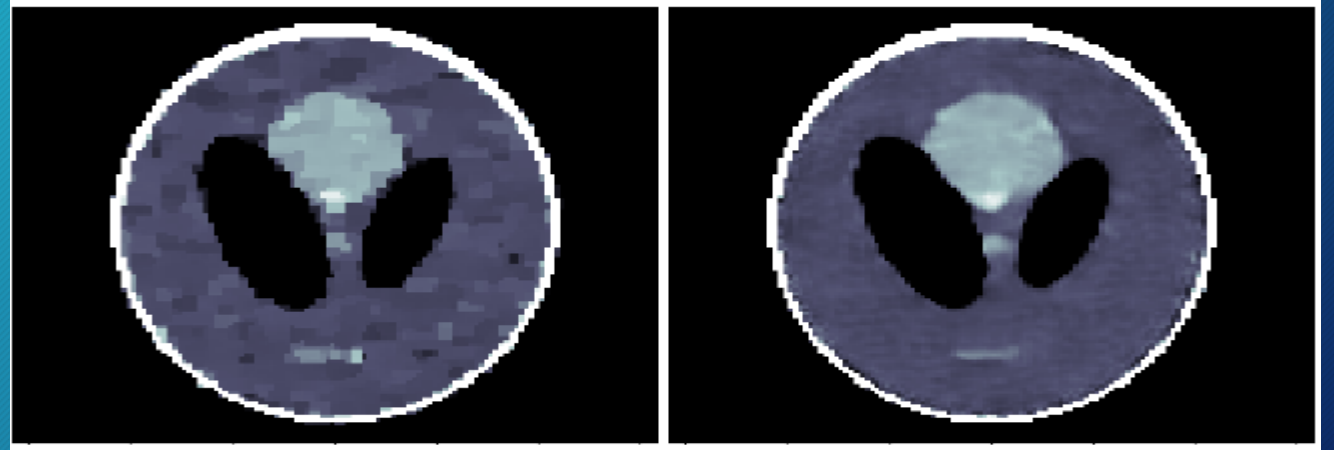
- Reconstructing Shepp-Logan phantom using FBP, TV and PL gradient scheme.

	PSNR (dB)		Runtime (ms)	
Method	Ellipses	Heads	Ellipses	Heads
FBP	19.75	36.12	4	**130**
Learned	**32.02**	**43.82**	58	430
TV	29.83	38.40	11 963	173 845

Comparison of the learned method with standard methods.



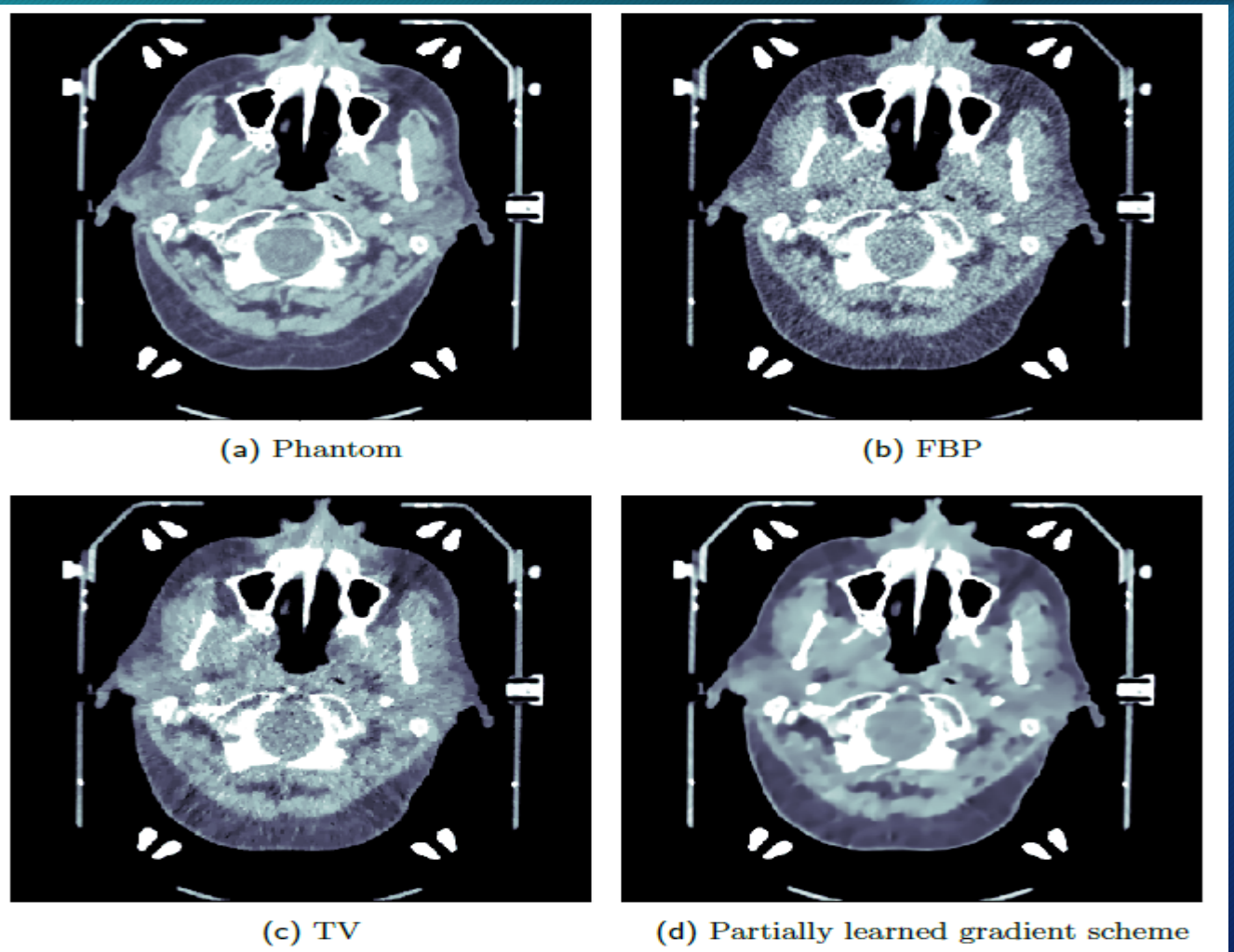
(a) Phantom (b) FBP



(c) TV (d) Partially learned gradient scheme

Results

- Reconstructing a head phantom using FBP, TV and PL gradient scheme.
- Note that for ellipse data, The FBP algorithm performs very poorly under noise while the TV and learned methods give comparable results.
- Note that for head dataset, where the noise is is lower, FBP reconstruction performs much better and is significantly better than the TV w.r.t the PSNR.
- The runtime of the PL algorithm is slower than FBP but faster than TV methods as shown in table.



(a) Phantom (b) FBP

(c) TV (d) Partially learned gradient scheme

Results

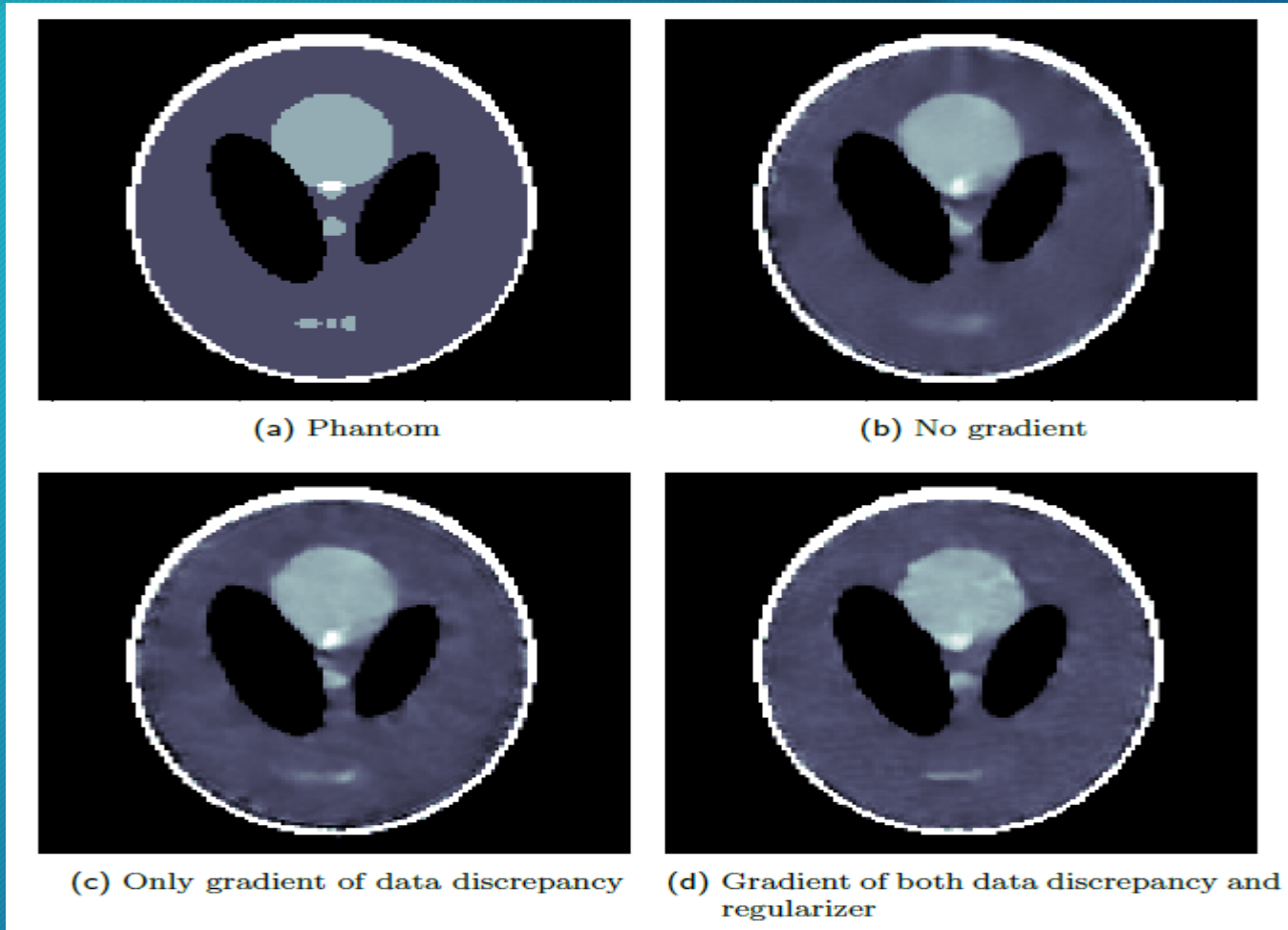
- Impact of including the gradient mappings

$$\nabla[L(T(.), g)] \, , \nabla S : X \to X$$

in the PL gradient scheme.

- Without the gradient, the PSNR was 29.65 dB while it was 30.51 dB with the gradient of the data discrepancy.

- Method took 19 ms without gradients, 64 ms with gradient of data discrepancy and 66 ms with both gradients.



(a) Phantom

(b) No gradient

(c) Only gradient of data discrepancy

(d) Gradient of both data discrepancy and regularizer

Discussion and Conclusion

- The partially learned gradient scheme is presented with a strong emphasis on the algorithmic aspects, its implementation and its performance.
- The framework for partially learned reconstruction was primarily motivated by a number of use cases involving ill-posed inverse problems.
- The PL gradient scheme does not have an explicit regularization parameter, its regularization properties are implicitly contained in the training dataset. Hence, a significant change in the training dataset would require re-training.
- Numerical experiments on tomographic data shows that the method gives notably better reconstructions than traditional FBP and TV regularization.
- Adding prior information improves the reconstruction.
- using prior knowledge about the forward operator, data acquisition, data noise model and regulariser can significantly improve the performance of deep learning based approaches for solving inverse problems.
- A possible way to improve PL algorithm and further leverage the power of the learning approach is to use a more sophisticated error functional.

End

Thank you for your attention