Lipschitz neural networks for image restoration

M. Terris†
joint works with T. Moreau, J. Tachella, et al.

†INRIA

SCAI
February 2024
Paris, France

Introduction

Inverse imaging problems

Imaging problems: recover x given observation z as

$$z = Hx + e$$

with $H\colon \mathbb{R}^n \to \mathbb{K}^m$ linear, $e \in \mathbb{K}^m$ realisation of random noise.

Aim: recover an estimate of x from z.

Inverse imaging problems

Imaging problems: recover x given observation z as

$$z = Hx + e$$

with $H\colon \mathbb{R}^n \to \mathbb{K}^m$ linear, $e \in \mathbb{K}^m$ realisation of random noise.

Aim: recover an estimate of x from z.

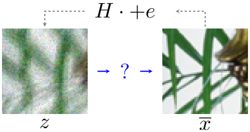
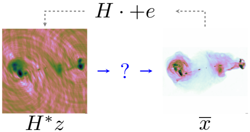
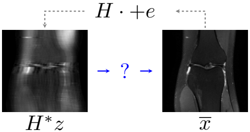


Image restoration



Astronomical imaging



Magnetic resonance imaging (MRI)

Inverse imaging problems

Imaging problems: recover x given observation z as

$$z = Hx + e$$

with $H\colon \mathbb{R}^n \to \mathbb{K}^m$ linear, $e \in \mathbb{K}^m$ realisation of random noise.

Aim: recover an estimate of x from z.

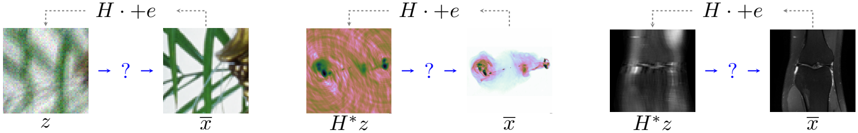


Image restoration | Astronomical imaging | Magnetic resonance imaging (MRI)

How do we solve such problem?

The optimisation approach: towards plug-and-play (PnP)

Aim: recover an estimate $\widehat{x}$ of x from z as

$$z = Hx + e$$

The optimisation approach: towards plug-and-play (PnP)

Aim: recover an estimate $\widehat{x}$ of x from z as

$$z = Hx + e$$

An estimate can be found through $p(x|z)$. For example, a maximum-a-posteriori approach gives:

$$\arg\max_x \log p(x|z) = \arg\min_x - \log p(z|x) - \log p(x)$$

The optimisation approach: towards plug-and-play (PnP)

Aim: recover an estimate $\widehat{x}$ of x from z as

$$z = Hx + e$$

An estimate can be found through $p(x|z)$. For example, a maximum-a-posteriori approach gives:

$$\arg\max_x \log p(x|z) = \arg\min_x -\log p(z|x) - \log p(x)$$

Reformulation as a mimization problem:

$$\widehat{x} = \underset{x}{\text{argmin}} \; \underbrace{f(x)}_{\text{data-fidelity}} + \underbrace{r(x)}_{\text{regularizer (prior)}}$$

The optimisation approach: towards plug-and-play (PnP)

Aim: recover an estimate $\widehat{x}$ of x from z as

$$z = Hx + e$$

An estimate can be found through $p(x|z)$. For example, a maximum-a-posteriori approach gives:

$$\arg\max_x \log p(x|z) = \arg\min_x - \log p(z|x) - \log p(x)$$

Reformulation as a mimization problem:

$$\widehat{x} = \underset{x}{\text{argmin}} \; \underbrace{f(x)}_{\text{data-fidelity}} + \underbrace{r(x)}_{\text{regularizer (prior)}}$$

Classical choice:

- $f(x) = \frac{1}{2}\|Hx - z\|^2$
- $r(x) = \lambda \, \mathrm{TV}(x), \;\; r(x) = \lambda \|\Psi x\|_1 \ldots$

The optimisation approach: towards plug-and-play (PnP)

Aim: recover an estimate $\widehat{x}$ of x from z as

$$z = Hx + e$$

An estimate can be found through $p(x|z)$. For example, a maximum-a-posteriori approach gives:

$$\arg\max_x \log p(x|z) = \arg\min_x -\log p(z|x) - \log p(x)$$

Reformulation as a mimization problem:

$$\widehat{x} = \operatorname*{argmin}_x \underbrace{f(x)}_{\text{data-fidelity}} + \underbrace{r(x)}_{\text{regularizer (prior)}}$$

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = \operatorname{prox}_{\gamma r}\left(x_k - \gamma \nabla f(x_k)\right) \qquad \text{(PGD)}$$

Illustration

Let's solve one image deconvolution problem with

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = \mathrm{prox}_{\gamma r}\left(x_k - \gamma \nabla f(x_k)\right) \qquad \text{(PGD)}$$

and we choose $r(x) = \mathrm{TV}(x)$.

Illustration

Let's solve one image deconvolution problem with

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = \operatorname{prox}_{\gamma r}\left(x_k - \gamma \nabla f(x_k)\right) \qquad \text{(PGD)}$$

and we choose $r(x) = \mathrm{TV}(x)$.

$z = H\overline{x} + e$

Result of (PGD)

$\overline{x}$

Illustration

Let's solve one image deconvolution problem with

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = \operatorname{prox}_{\gamma r}\left(x_k - \gamma \nabla f(x_k)\right) \qquad \text{(PGD)}$$

and we choose $r(x) = \mathrm{TV}(x)$.

$z = H\overline{x} + e$

Result of (PGD)

$\overline{x}$

Can we do better?

PnP algorithms

Replace the proximity operator by a powerful **denoiser**:

PnP algorithms

Replace the proximity operator by a powerful **denoiser**:

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = \mathrm{prox}_{\gamma r}\left(x_k - \gamma \nabla f(x_k)\right) \qquad \text{(PGD)}$$

PnP algorithms

Replace the proximity operator by a powerful **denoiser**:

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = \mathrm{prox}_{\gamma r}\left(x_k - \gamma \nabla f(x_k)\right) \qquad \text{(PGD)}$$

PnP algorithms

Replace the proximity operator by a powerful **denoiser**:

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right) \qquad \text{(PnP-PGD)}$$

PnP algorithms

Replace the proximity operator by a powerful **denoiser**:

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right) \qquad \text{(PnP-PGD)}$$

Usually, J is a deep neural network (DNN). But why a denoiser?

PnP algorithms

Replace the proximity operator by a powerful **denoiser**:

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right) \qquad \text{(PnP-PGD)}$$

Usually, J is a deep neural network (DNN). But why a denoiser?

$y = x + e$

$\mathrm{prox}_{\mathsf{TV}}(y)$

$\mathrm{DRUNet}(y)$

x

PnP algorithms

Replace the proximity operator by a powerful **denoiser**:

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right) \qquad \text{(PnP-PGD)}$$

Usually, J is a deep neural network (DNN). But why a denoiser?

Because it is very easy to train!

PnP algorithms

Replace the proximity operator by a powerful **denoiser**:

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right) \qquad \text{(PnP-PGD)}$$

Usually, J is a deep neural network (DNN). But why a denoiser?

$y = x + e$

$\mathrm{prox}_{\mathsf{TV}}(y)$

$\mathrm{DRUNet}(y)$

x

Take home message 1: denoisers act as implicit priors!

PnP algorithms: illustration

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right)$$

where $J = \mathrm{DRUNet}$.

$z = H\overline{x} + e$

$x_0 - \gamma \nabla f(x_0)$

$J(x_0 - \gamma \nabla f(x_0))$

PnP algorithms: illustration

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right)$$

where $J = \mathrm{DRUNet}$.

$z = H\overline{x} + e$

$x_0 - \gamma \nabla f(x_0)$

$J(x_0 - \gamma \nabla f(x_0))$
$:= x_1$

PnP algorithms: illustration

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right)$$

where $J = \mathrm{DRUNet}$.

$z = H\overline{x} + e$

$x_1 - \gamma \nabla f(x_1)$

$J(x_1 - \gamma \nabla f(x_1))$
$:= x_2$

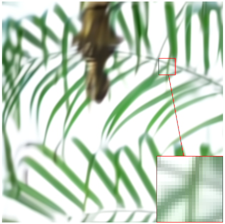
PnP algorithms: illustration

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right)$$

where $J = \mathrm{DRUNet}$.

$z = H\overline{x} + e$

$x_{10} - \gamma \nabla f(x_{10})$



$J(x_{10} - \gamma \nabla f(x_{10}))$
$:= x_{11}$

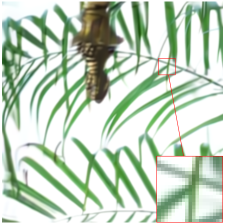
PnP algorithms: illustration

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right)$$

where $J = \mathrm{DRUNet}$.

$z = H\overline{x} + e$

$x_{30} - \gamma \nabla f(x_{30})$



$J(x_{30} - \gamma \nabla f(x_{30}))$
$:= x_{31}$

PnP algorithms: illustration

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right)$$

where $J = \mathrm{DRUNet}$.

$z = H\overline{x} + e$

$x_{50} - \gamma \nabla f(x_{50})$

$J(x_{50} - \gamma \nabla f(x_{50}))$

PnP algorithms: illustration

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right)$$

where $J = \mathrm{DRUNet}$.

$z = H\overline{x} + e$

$x_{100} - \gamma \nabla f(x_{100})$

$J(x_{100} - \gamma \nabla f(x_{100}))$

Nice results after ~ 50 iterations, but **does not converge**...

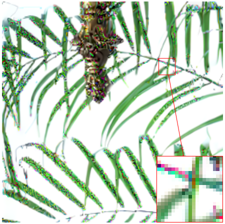
PnP algorithms: illustration

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right)$$

where $J = \mathrm{DRUNet}$.

$z = H\overline{x} + e$

$x_{200} - \gamma \nabla f(x_{200})$



$J(x_{200} - \gamma \nabla f(x_{200}))$

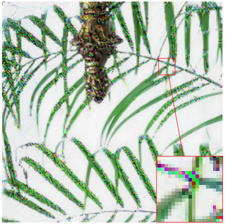
Nice results after ~ 50 iterations, but **does not converge**...

PnP algorithms: illustration

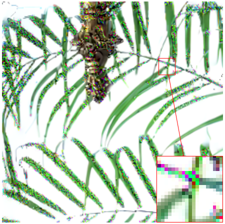
$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right)$$

where $J = \mathrm{DRUNet}$.

$z = H\overline{x} + e$



$x_{300} - \gamma \nabla f(x_{300})$



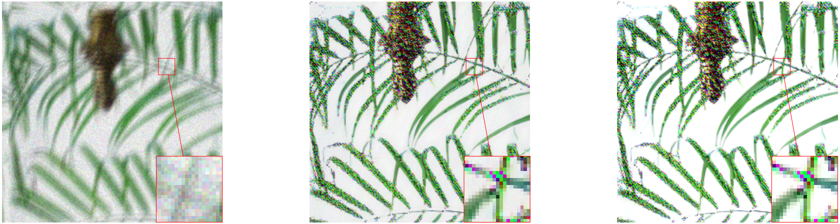
$J(x_{300} - \gamma \nabla f(x_{300}))$

Nice results after ~ 50 iterations, but **does not converge**...

PnP algorithms: illustration

$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right)$$

where $J = \mathrm{DRUNet}$.



$z = H\overline{x} + e$ | $x_{900} - \gamma \nabla f(x_{900})$ | $J(x_{900} - \gamma \nabla f(x_{900}))$

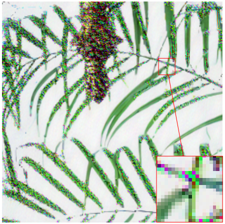
Nice results after ~ 50 iterations, but **does not converge**...

PnP algorithms: illustration

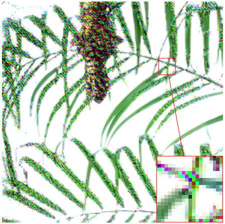
$$(\forall k \in \mathbb{N}) \quad x_{k+1} = J\left(x_k - \gamma \nabla f(x_k)\right)$$

where $J = \mathrm{DRUNet}$.

$z = H\overline{x} + e$



$x_{900} - \gamma \nabla f(x_{900})$



$J(x_{900} - \gamma \nabla f(x_{900}))$

Nice results after ~ 50 iterations, but **does not converge**...

This raises many questions!

In this presentation

Questions:

- How to solve the non-convergence problem?
- Can one restore the link between a prior and the DNN in the PnP algorithm?
- Do we really need constraints?

Outline:

1. Resolvent architectures through 1-Lip regularisation (arxiv 2012.13247)
2. Beyond Lipschitz constraints (arxiv 2312.01831)

Learning resolvent networks

Convergence and characterisation

$$x_{k+1} = J_\theta(x_k - \gamma \nabla f(x_k)) \qquad \text{(PnP-PGD)}$$

Convergence and characterisation

$$x_{k+1} = J_\theta(x_k - \gamma \nabla f(x_k)) \qquad \text{(PnP-PGD)}$$

Definition

We say that $J_\theta \colon \mathcal{H} \to \mathcal{H}$ is firmly nonexpansive it there exists a 1-Lipschitz operator $Q_\theta \colon \mathcal{H} \to \mathcal{H}$ such that

$$J_\theta = \frac{\mathrm{Id} + Q_\theta}{2}.$$

Convergence and characterisation

$$x_{k+1} = J_\theta(x_k - \gamma \nabla f(x_k)) \tag{PnP-PGD}$$

Definition

We say that $J_\theta \colon \mathcal{H} \to \mathcal{H}$ is firmly nonexpansive it there exists a 1-Lipschitz operator $Q_\theta \colon \mathcal{H} \to \mathcal{H}$ such that

$$J_\theta = \frac{\mathrm{Id} + Q_\theta}{2}.$$

Theorem (informal)

If J_θ is firmly nonexpansive and γ is small enough, there exists a convex function g_θ such that $(x_k)_{k\in\mathbb{N}}$ in (PnP-PGD) converges to $x \in \mathbb{R}^N$ satisfying

$$0 \in \gamma \nabla f(x) + \partial g_\theta(x).$$

How to?

Goal:

Build a DNN denoiser J, i.e. $2J - \mathrm{Id}$ is 1-Lipschitz.

Two possible approaches

How to?

Goal:

Build a DNN denoiser J, i.e. $2J - \mathrm{Id}$ is 1-Lipschitz.

Two possible approaches

A tight approach

Define an architecture of J s.t.

$$J = \frac{\mathrm{Id} + Q}{2}$$

with Q 1-Lipschitz.

How to?

Goal:

Build a DNN denoiser J, i.e. $2J - \mathrm{Id}$ is 1-Lipschitz.

Two possible approaches

A tight approach

Define an architecture of J s.t.

$$J = \frac{\mathrm{Id} + Q}{2}$$

with Q 1-Lipschitz.

A relaxed approach

Regularise the training loss as

$$\mathsf{loss}_{\mathsf{usual}} + \lambda \,\mathrm{Lip}(2J - \mathrm{Id})$$

Applies to **any kind** of architecture (but not tight...).

Application to denoising

Goal: build a DNN denoiser J s.t. $2J - \mathrm{Id}$ is 1-Lipschitz, **regardless of the architecture**.

Application to denoising

Goal: build a DNN denoiser J s.t. $2J - \mathrm{Id}$ is 1-Lipschitz, **regardless of the architecture**.

Step 1: Training dataset $(\overline{x}, y)$:

- x: groundtruth (target) image;
- $y = x + \sigma n$: noisy image.

Application to denoising

Goal: build a DNN denoiser J s.t. $2J - \mathrm{Id}$ is 1-Lipschitz, **regardless of the architecture**.

Step 1: Training dataset $(\overline{x}, y)$:

- x: groundtruth (target) image;
- $y = x + \sigma n$: noisy image.

Step 2: Proposed training loss:

$$\underset{\theta \in \mathbb{R}^P}{\text{minimize}} \; \frac{1}{L} \sum_{\ell=1}^{L} \underbrace{\|J_\theta(y_\ell) - x_\ell\|_1}_{\text{denoising}} + \underbrace{\lambda \max\left\{\|\nabla Q_\theta(y_\ell)\|^2, 1 - \varepsilon\right\}}_{\text{relaxed 1-Lip constraint}}$$

where $Q_\theta = 2J_\theta - \mathrm{Id}$, and where $\nabla(\cdot)$ denotes the Jacobian operator.

Application to denoising

Goal: build a DNN denoiser J s.t. $2J - \mathrm{Id}$ is 1-Lipschitz, **regardless of the architecture**.

Step 1: Training dataset $(\overline{x}, y)$:

- x: groundtruth (target) image;
- $y = x + \sigma n$: noisy image.

Step 2: Proposed training loss:

$$\underset{\theta \in \mathbb{R}^P}{\text{minimize}} \; \frac{1}{L} \sum_{\ell=1}^{L} \underbrace{\| J_\theta(y_\ell) - x_\ell \|_1}_{\text{denoising}} + \underbrace{\lambda \max \left\{ \| \nabla Q_\theta(y_\ell) \|^2, 1 - \varepsilon \right\}}_{\text{relaxed 1-Lip constraint}}$$

where $Q_\theta = 2J_\theta - \mathrm{Id}$, and where $\nabla(\cdot)$ denotes the Jacobian operator.

$\| \nabla Q_\theta(x_\ell) \|$ is an approximation of the Lipschitz constant of $Q = 2J - \mathrm{Id}$

Application to denoising

Goal: build a DNN denoiser J s.t. $2J - \mathrm{Id}$ is 1-Lipschitz, **regardless of the architecture**.

Step 1: Training dataset $(\overline{x}, y)$:

- x: groundtruth (target) image;
- $y = x + \sigma n$: noisy image.

Step 2: Proposed training loss:

$$\underset{\theta \in \mathbb{R}^P}{\text{minimize}} \ \frac{1}{L} \sum_{\ell=1}^{L} \underbrace{\|J_\theta(y_\ell) - x_\ell\|_1}_{\text{denoising}} + \underbrace{\lambda \max\left\{\|\nabla Q_\theta(y_\ell)\|^2, 1 - \varepsilon\right\}}_{\text{relaxed 1-Lip constraint}}$$

where $Q_\theta = 2J_\theta - \mathrm{Id}$, and where $\nabla(\cdot)$ denotes the Jacobian operator.

$\|\nabla Q_\theta(x_\ell)\|$ is an approximation of the Lipschitz constant of $Q = 2J - \mathrm{Id}$

$$\|\nabla Q\| \leq 1 \Rightarrow \text{ convergence of PnP}$$

About the training loss...

$$\underset{\theta\in\mathbb{R}^P}{\text{minimize}}\ \frac{1}{L}\sum_{\ell=1}^{L} \underbrace{\|J_\theta(y_\ell)-\overline{x}_\ell\|_1}_{\textbf{denoising}} + \underbrace{\lambda \max\left\{\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2, 1-\varepsilon\right\}}_{\textbf{relaxed 1-Lip constraint}}$$

where $\nabla(\cdot)$ denotes the Jacobian operator and $\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2$ is an approximation of the Lipschitz constant of $Q = 2J - \mathrm{Id}$.

About the training loss...

$$\underset{\theta\in\mathbb{R}^P}{\text{minimize}}\ \frac{1}{L}\sum_{\ell=1}^{L} \underbrace{\|J_\theta(y_\ell)-\overline{x}_\ell\|_1}_{\text{denoising}} + \underbrace{\lambda \max\left\{\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2, 1-\varepsilon\right\}}_{\text{relaxed 1-Lip constraint}}$$

where $\nabla(\cdot)$ denotes the Jacobian operator and $\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2$ is an approximation of the Lipschitz constant of $Q = 2J - \mathrm{Id}$.

No access to ∇Q, but we can compute $\|\nabla Q\|$ with autograd!

About the training loss...

$$\underset{\theta\in\mathbb{R}^P}{\text{minimize}}\ \frac{1}{L}\sum_{\ell=1}^{L} \underbrace{\|J_\theta(y_\ell) - \overline{x}_\ell\|_1}_{\textbf{denoising}} + \underbrace{\lambda \max\left\{\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2, 1-\varepsilon\right\}}_{\textbf{relaxed 1-Lip constraint}}$$

where $\nabla(\cdot)$ denotes the Jacobian operator and $\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2$ is an approximation of the Lipschitz constant of $Q = 2J - \mathrm{Id}$.

No access to ∇Q, but we can compute $\|\nabla Q\|$ with autograd!

How?

Given a function (DNN) Q:

- the `grad` operation in PyTorch gives the product $u \times \mathrm{Jac}(Q)^\top$;
- the "double backward trick" gives $\mathrm{Jac}(Q) \times v$.

About the training loss...

$$\underset{\theta\in\mathbb{R}^P}{\text{minimize}}\ \frac{1}{L}\sum_{\ell=1}^{L} \underbrace{\|J_\theta(y_\ell)-\overline{x}_\ell\|_1}_{\text{denoising}} + \underbrace{\lambda \max\left\{\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2, 1-\varepsilon\right\}}_{\text{relaxed 1-Lip constraint}}$$

where $\nabla(\cdot)$ denotes the Jacobian operator and $\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2$ is an approximation of the Lipschitz constant of $Q = 2J - \mathrm{Id}$.

No access to ∇Q, but we can compute $\|\nabla Q\|$ with autograd!

How?

Given a function (DNN) Q:

- the `grad` operation in PyTorch gives the product $u \times \mathrm{Jac}(Q)^\top$;
- the "double backward trick" gives $\mathrm{Jac}(Q) \times v$.

These are all the ingredients for using the power method!

About the training loss...

$$\underset{\theta\in\mathbb{R}^P}{\text{minimize}}\ \frac{1}{L}\sum_{\ell=1}^{L} \underbrace{\|J_\theta(y_\ell)-\overline{x}_\ell\|_1}_{\textbf{denoising}} + \underbrace{\lambda \max\left\{\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2, 1-\varepsilon\right\}}_{\textbf{relaxed 1-Lip constraint}}$$

where $\nabla(\cdot)$ denotes the Jacobian operator and $\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2$ is an approximation of the Lipschitz constant of $Q = 2J - \mathrm{Id}$.

No access to ∇Q, but we can compute $\|\nabla Q\|$ with autograd!

```
import torch.autograd.grad as grad

...
for n_it in range(num_iter):
    w = torch.ones_like(y, requires_grad=True)
    v = grad(grad(y, x, w, create_graph=True), w, u, create_graph=True)[0]
    v = grad(y, x, v, retain_graph=True, create_graph=True)[0]
```

About the training loss...

$$\underset{\theta\in\mathbb{R}^P}{\text{minimize}}\ \frac{1}{L}\sum_{\ell=1}^{L}\underbrace{\|J_\theta(y_\ell)-\overline{x}_\ell\|_1}_{\text{denoising}} + \underbrace{\lambda \max\left\{\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2, 1-\varepsilon\right\}}_{\text{relaxed 1-Lip constraint}}$$

where $\nabla(\cdot)$ denotes the Jacobian operator and $\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2$ is an approximation of the Lipschitz constant of $Q = 2J - \mathrm{Id}$.

No access to ∇Q, but we can compute $\|\nabla Q\|$ with autograd!

```
import torch.autograd.grad as grad

...
for n_it in range(num_iter):
   w = torch.ones_like(y, requires_grad=True)
   v = grad(grad(y, x, w, create_graph=True), w, u, create_graph=True)[0]
   v = grad(y, x, v, retain_graph=True, create_graph=True)[0]
```

Take home message 2:
backprop allows to compute the lipschitz constant of a DNN!

Influence of the Jacobian penalization

$$\underset{\theta\in\mathbb{R}^P}{\text{minimize}}\ \frac{1}{L}\sum_{\ell=1}^{L} \underbrace{\|J_\theta(y_\ell)-\overline{x}_\ell\|_1}_{\textbf{denoising}} + \underbrace{\lambda \max\left\{\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2, 1-\varepsilon\right\}}_{\textbf{relaxed FNE constraint}}$$

Influence of the Jacobian penalization

$$\underset{\theta \in \mathbb{R}^P}{\text{minimize}} \ \frac{1}{L} \sum_{\ell=1}^{L} \underbrace{\| J_\theta(y_\ell) - \overline{x}_\ell \|_1}_{\text{denoising}} + \underbrace{\lambda \max \left\{ \| \nabla Q_\theta(\widetilde{x}_\ell) \|^2, 1 - \varepsilon \right\}}_{\text{relaxed FNE constraint}}$$

Convergence of PnP depending on the value of λ.

☛ Deblurring problem: $\overline{x}$ from BSD10 test set

Influence of the Jacobian penalization

$$\underset{\theta\in\mathbb{R}^P}{\text{minimize}} \; \frac{1}{L}\sum_{\ell=1}^{L} \underbrace{\|J_\theta(y_\ell) - \overline{x}_\ell\|_1}_{\text{denoising}} + \underbrace{\lambda \max\left\{\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2, 1-\varepsilon\right\}}_{\text{relaxed FNE constraint}}$$

Convergence of PnP depending on the value of λ.

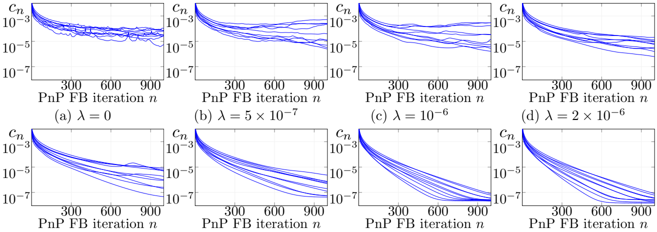
☛ $c_k = \|x_k - x_{k-1}\|/\|x_0\|$, for $(x_k)_{k\in\mathbb{N}}$ should be monotone

Influence of the Jacobian penalization

$$\underset{\theta\in\mathbb{R}^P}{\text{minimize}}\ \frac{1}{L}\sum_{\ell=1}^{L} \underbrace{\|J_\theta(y_\ell)-\overline{x}_\ell\|_1}_{\text{denoising}} + \underbrace{\lambda \max\left\{\|\nabla Q_\theta(\widetilde{x}_\ell)\|^2, 1-\varepsilon\right\}}_{\text{relaxed FNE constraint}}$$

Convergence of PnP depending on the value of λ.

- $c_k = \|x_k - x_{k-1}\|/\|x_0\|$, for $(x_k)_{k\in\mathbb{N}}$ should be monotone

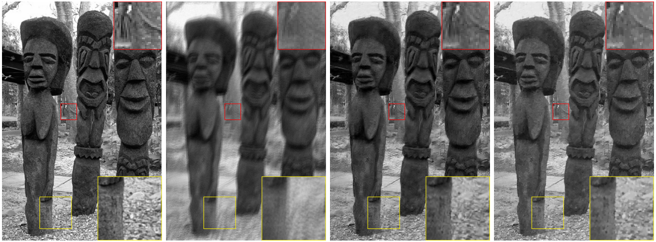


(a) $\lambda = 0$ (b) $\lambda = 5\times10^{-7}$ (c) $\lambda = 10^{-6}$ (d) $\lambda = 2\times10^{-6}$

(e) $\lambda = 5\times10^{-6}$ (f) $\lambda = 10^{-5}$ (g) $\lambda = 4\times10^{-5}$ (h) $\lambda = 1.6\times10^{-4}$

https://arxiv.org/abs/2012.13247

Visual results

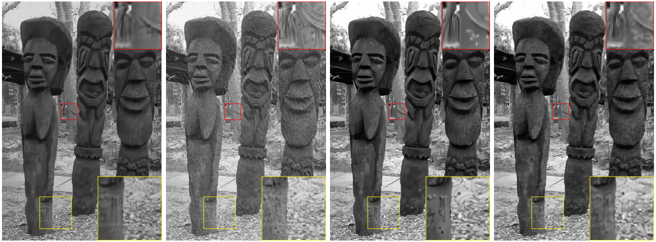


(a) Groundtruth

(b) Observation $(20.48, 0.387)$

(c) $\mathrm{prox}_{\mu\|\Psi^\dagger\cdot\|_1}$ $(26.13, 0.775)$

(d) $\mathrm{prox}_{\mu\|\cdot\|_{\mathrm{TV}}}$ $(26.57, 0.787)$



(e) BM3D $(26.09, 0.732)$

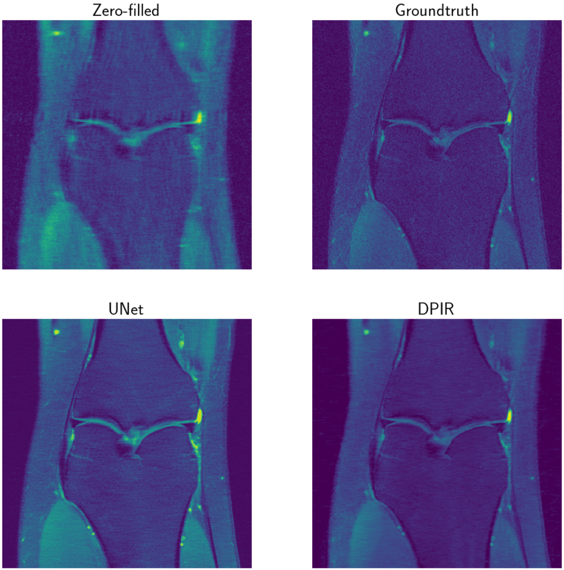
(f) RealSN $(24.68, 0.726)$

(g) DnCNN $(26.12, 0.643)$

(h) Proposed $(\mathbf{27.09}, \mathbf{0.789})$

Beyond 1-Lipschitz networks

Do we really need 1 Lipschitz?



Do we really need 1 Lipschitz?

2 antagonist observations:

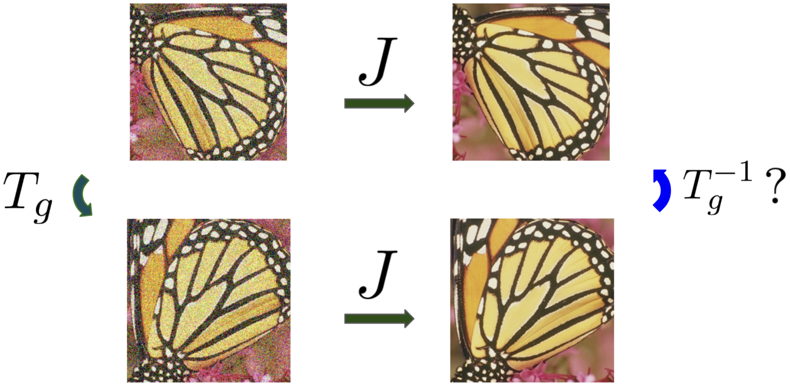
- DNNs seem to behave like proximity operators from far, but they are not 1-Lipschitz.
- Imposing 1-Lipschitz constraints solves the unstability issue, but lowers performance.

Do we really need 1 Lipschitz?

Idea: Imaging priors should have some invariance properties with respect to certain groups of transformations, such as rotations, translations, and reflections. We denote these transformations associated with a group $\mathcal{G}$, $\{T_g\}_{g\in\mathcal{G}}$ where $T_g \in \mathbb{R}^{n\times n}$ is a unitary matrix.

Do we really need 1 Lipschitz?

Idea: Imaging priors should have some invariance properties with respect to certain groups of transformations, such as rotations, translations, and reflections. We denote these transformations associated with a group $\mathcal{G}$, $\{T_g\}_{g\in\mathcal{G}}$ where $T_g \in \mathbb{R}^{n\times n}$ is a unitary matrix.



Do we really need 1 Lipschitz?

Idea: Imaging priors should have some invariance properties with respect to certain groups of transformations, such as rotations, translations, and reflections. We denote these transformations associated with a group $\mathcal{G}$, $\{T_g\}_{g\in\mathcal{G}}$ where $T_g \in \mathbb{R}^{n\times n}$ is a unitary matrix.

Definition

We say that J is equivariant to the group action $\{T_g\}_{g\in\mathcal{G}}$ if $J(T_g x) = T_g J(x)$ for all x and $g \in \mathcal{G}$.

Do we really need 1 Lipschitz?

Any operator J can be made $\mathcal{G}$-equivariant through this averaging procedure:

$$J_{\mathcal{G}}(x) \overset{\text{def}}{=} \frac{1}{|\mathcal{G}|} \sum_{g \in \mathcal{G}} T_g^{-1} J(T_g x). \tag{1}$$

Do we really need 1 Lipschitz?

Any operator J can be made $\mathcal{G}$-equivariant through this averaging procedure:

$$J_{\mathcal{G}}(x) \overset{\text{def}}{=} \frac{1}{|\mathcal{G}|} \sum_{g \in \mathcal{G}} T_g^{-1} J(T_g x). \tag{1}$$

Why can it help?

Do we really need 1 Lipschitz?

Any operator J can be made $\mathcal{G}$-equivariant through this averaging procedure:

$$J_{\mathcal{G}}(x) \stackrel{\text{def}}{=} \frac{1}{|\mathcal{G}|} \sum_{g \in \mathcal{G}} T_g^{-1} J(T_g x). \tag{1}$$

Why can it help?

Proposition

Assume that J is a linear denoiser with singular value decomposition $J = \sum_{i=1}^{n} \lambda_i u_i v_i^\top$ and $\lambda_1 > \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. If the principal component $u_1 v_1^\top$ is not $\mathcal{G}$-equivariant, then the averaged denoiser $J_{\mathcal{G}}$ has a strictly smaller Lipschitz constant than J.

Do we really need 1 Lipschitz?

Any operator J can be made $\mathcal{G}$-equivariant through this averaging procedure:

$$J_{\mathcal{G}}(x) \stackrel{\text{def}}{=} \frac{1}{|\mathcal{G}|} \sum_{g \in \mathcal{G}} T_g^{-1} J(T_g x). \tag{1}$$

Why can it help?

Proposition

Assume that J is a linear denoiser with singular value decomposition $J = \sum_{i=1}^{n} \lambda_i u_i v_i^\top$ and $\lambda_1 > \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. If the principal component $u_1 v_1^\top$ is not $\mathcal{G}$-equivariant, then the averaged denoiser $J_{\mathcal{G}}$ has a strictly smaller Lipschitz constant than J.

Take home meassage 3:
Equivariance can reduce the Lipschitz constant!

Results

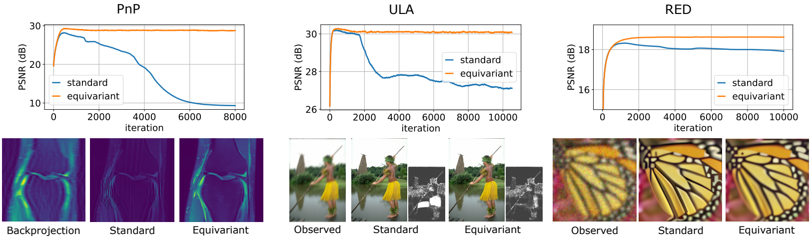
Equivariant PnP:

$$
\begin{aligned}
&\text{Sample } \; g_k \sim \mathcal{G} \\
&\text{Set } \; \widetilde{J}_{\mathcal{G},k}(x) = T_{g_k}^{-1} J(T_{g_k} x) \qquad \text{(eq. PnP-PGD)} \\
&x_{k+1} = \widetilde{J}_{\mathcal{G},k}\left(x_k - \gamma A^\top (A x_k - y)\right).
\end{aligned}
$$

Results

Equivariant PnP:

$$\begin{aligned} &\text{Sample } g_k \sim \mathcal{G} \\ &\text{Set } \widetilde{J}_{\mathcal{G},k}(x) = T_{g_k}^{-1} J(T_{g_k} x) \\ &x_{k+1} = \widetilde{J}_{\mathcal{G},k}\left(x_k - \gamma A^\top (A x_k - y)\right). \end{aligned} \qquad \text{(eq. PnP-PGD)}$$



https://arxiv.org/abs/2312.01831

Conclusion

Conclusion (i)

We have shown:

- 1-Lipschitz denoisers yield convergent PnP algorithms;
- Equivariance can lower the Lipschitz constant of denoisers.

Conclusion (i)

We have shown:

- 1-Lipschitz denoisers yield convergent PnP algorithms;
- Equivariance can lower the Lipschitz constant of denoisers.

But more than this:

- We aim at solving problems of the form $y = Ax + e$;
- Minimization problems of the form $\arg\min_x f(x) + g(x)$ are replaced with PnP
- Applied for 2D, 3D, 3D+time imaging problems...

Conclusion (i)

We have shown:

- 1-Lipschitz denoisers yield convergent PnP algorithms;
- Equivariance can lower the Lipschitz constant of denoisers.

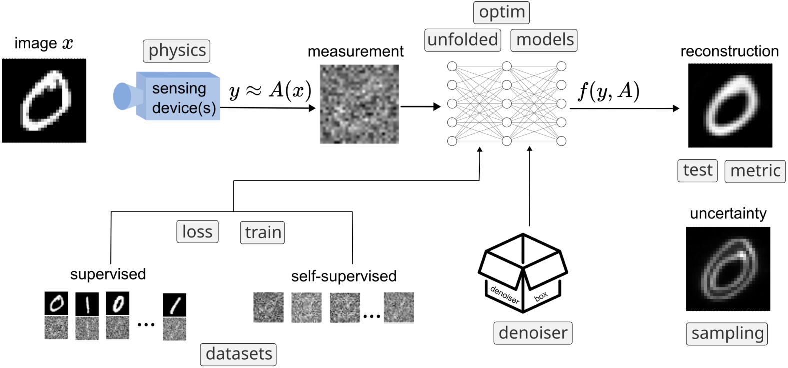
But more than this:

- We aim at solving problems of the form $y = Ax + e$;
- Minimization problems of the form $\arg\min_x f(x) + g(x)$ are replaced with PnP
- Applied for 2D, 3D, 3D+time imaging problems...

but how about other problems?

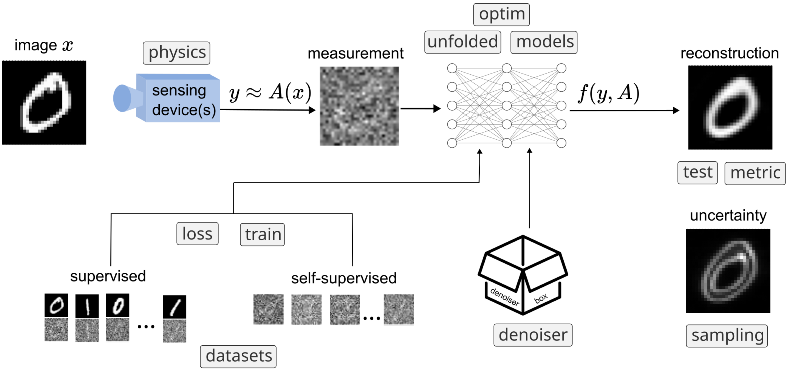
Conclusion (ii)

All this is implemented in our **brand new** library
`https://deepinv.github.io/deepinv/`!



Conclusion (ii)

All this is implemented in our **brand new** library
`https://deepinv.github.io/deepinv/`!



Thank you!

References

- 1-Lipschitz denoisers and PnP: https://arxiv.org/abs/2012.13247
- Equivariant PnP: https://arxiv.org/abs/2312.01831