Deep Regularization via Bi-Level Optimization

Srijith Nair and Philip Schniter

THE OHIO STATE UNIVERSITY

Supported by NSF grant CCF-1955587

2022 Asilomar Conf. Signals, Systems, and Computers

Paper ID: 1259

Linear inverse problems

Goal: Recover signal $\boldsymbol{x}$ from noisy measurements $\boldsymbol{y}$:

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{w}.$$

- Applications: deblurring, superresolution, inpainting, computed tomography, magnetic resonance imaging (MRI), etc.

Challenge: Often, $\boldsymbol{A}$ is not full-column rank

- Components of $\boldsymbol{x}$ in $\mathrm{null}(\boldsymbol{A})$ are not measured!
- Accurate recovery requires leveraging prior information about $\boldsymbol{x}$

The variational method

A common approach is to recover the signal by solving an optimization problem:

$$\widehat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \left\{ \ell(\boldsymbol{x}; \boldsymbol{y}) + \lambda r(\boldsymbol{x}) \right\}$$

- The data-fidelity term $\ell(\cdot; \boldsymbol{y})$ is typically chosen as the negative log likelihood, e.g., $\ell(\boldsymbol{x}; \boldsymbol{y}) = \frac{1}{2\sigma^2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|^2$ for white Gaussian $\boldsymbol{w}$
- The regularization $r(\cdot)$ is difficult to choose. We want $r(\cdot)$ such that
  - the cost has no local minima (e.g., $r(\cdot)$ is convex)
  - the optimization is not expensive (e.g., $r(\cdot)$ is differentiable or proximable)
  - the solution is accurate (i.e., $\widehat{\boldsymbol{x}} \approx \boldsymbol{x}_0$)
- The regularization weight $\lambda > 0$ should be tuned for best performance
- An example optimization algorithm is proximal gradient descent (PGD):

$$\widehat{\boldsymbol{x}}_{\text{new}} = \mathrm{prox}_{\ell\tau/\lambda}\left(\widehat{\boldsymbol{x}} - \tau\nabla r(\widehat{\boldsymbol{x}})\right), \quad \tau \in \left(0, \tfrac{1}{\mathrm{Lip}(\nabla r)}\right]$$

Deep regularization

We propose to implement $r_{\boldsymbol{\theta}}(\cdot)$ using a deep neural network and compute the gradient $\nabla r_{\boldsymbol{\theta}}(\widehat{\boldsymbol{x}})$ in PGD using automatic differentiation

Key questions:

- How should $r_{\boldsymbol{\theta}}(\cdot)$ be constructed?
- How should $\boldsymbol{\theta}$ be trained?

Prior work

- RED algorithm: [1]
  - Train a deep denoiser $\boldsymbol{d}_{\boldsymbol{\theta}}(\boldsymbol{x})$ via $\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \mathbb{E}\{\|\boldsymbol{x}_0 - \boldsymbol{d}_{\boldsymbol{\theta}}(\boldsymbol{x}_0 + \boldsymbol{n})\|^2\}$
  - Set $r_{\boldsymbol{\theta}}(\boldsymbol{x}) \triangleq \frac{1}{2}\boldsymbol{x}^\top(\boldsymbol{x} - \boldsymbol{d}_{\boldsymbol{\theta}}(\boldsymbol{x}))$
  - Use the approximation $\nabla r_{\boldsymbol{\theta}}(\boldsymbol{x}) \approx \boldsymbol{x} - \boldsymbol{d}_{\boldsymbol{\theta}}(\boldsymbol{x})$ in PGD
    - For the above to be exact, we need that $\boldsymbol{d}_{\boldsymbol{\theta}}(\cdot)$ is Jacobian-symmetric and locally homogeneous, but these properties are not satisfied by most practical denoisers [2]
- Input-convex regularization with adversarial training: [3]
  - Construct $r_{\boldsymbol{\theta}}(\cdot)$ using an input-convex deep net [4]
  - Train $r_{\boldsymbol{\theta}}(\cdot)$ as a Wasserstein-based discriminator of real $\boldsymbol{x}$ versus fake $\boldsymbol{x}$
  - Use $\nabla r_{\boldsymbol{\theta}}(\cdot)$ in PGD
- Gradient-step approach: [5, 6]
  - Construct a deep regularizer $r_{\boldsymbol{\theta}}(\cdot)$
    - e.g., for some deep net $\boldsymbol{h}_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}^d$, construct $r_{\boldsymbol{\theta}}(\boldsymbol{x}) = \|\boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{x})\|^2$ or $r_{\boldsymbol{\theta}}(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{x})\|^2$ or $r_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{x}^\top(\boldsymbol{x} - \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{x}))$ or etc.
  - Train $\boldsymbol{d}_{\boldsymbol{\theta}}(\boldsymbol{x}) \triangleq \boldsymbol{x} - \nabla r_{\boldsymbol{\theta}}(\boldsymbol{x})$ as a denoiser
  - Use $\nabla r_{\boldsymbol{\theta}}(\cdot)$ in PGD

Proposed regularizer construction

- We propose to construct the deep regularizer $r_{\boldsymbol{\theta}}(\cdot)$ as

$$\boxed{r_{\boldsymbol{\theta}}(\boldsymbol{x}) = \tfrac{1+\alpha}{2}\|\boldsymbol{x}\|^2 - s_{\boldsymbol{\theta}}(\boldsymbol{x})}$$

with $\alpha \geq 0$ and β-smooth neural network $s_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}$, i.e.,

$$\|\nabla s_{\boldsymbol{\theta}}(\boldsymbol{x}_1) - \nabla s_{\boldsymbol{\theta}}(\boldsymbol{x}_2)\| \leq \beta\|\boldsymbol{x}_1 - \boldsymbol{x}_2\| \quad \forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^d$$

  - If $\beta = 1$, the regularizer $r_{\boldsymbol{\theta}}(\cdot)$ is convex (or strongly convex when $\alpha > 0$)
  - The structure of $s_{\boldsymbol{\theta}}(\cdot)$ is arbitrary
- We propose to enforce β-smoothness adversarially, by adding the following Lipschitz penalty while training the regularizer $r_{\boldsymbol{\theta}}(\cdot)$ [7]:

$$\sum_i \max\left\{0, \max_{\boldsymbol{\delta}} \widehat{\beta}_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{\delta}) - \beta\right\}$$

where $\{\boldsymbol{x}_i\}$ are the training samples and

$$\widehat{\beta}_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{\delta}) \triangleq \frac{\|\nabla s_{\boldsymbol{\theta}}(\boldsymbol{x}) - \nabla s_{\boldsymbol{\theta}}(\boldsymbol{x} + \boldsymbol{\delta})\|}{\|\boldsymbol{\delta}\|}$$

Proposed training scheme

We propose a bi-level training scheme [8]:

- $\boldsymbol{\theta}$ and λ are chosen to minimize the loss

$$\mathcal{L}(\boldsymbol{\theta}, \lambda) \triangleq \sum_{i=1}^n \left[\|\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta}, \lambda) - \boldsymbol{x}_i\|^2 + \xi \max\left\{0, \max_{\boldsymbol{\delta}_i} \widehat{\beta}_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{\delta}_i) - \beta\right\}\right]$$

where

$$\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta}, \lambda) = \arg\min_{\boldsymbol{x}} \{ \underbrace{\ell(\boldsymbol{x}; \boldsymbol{y}_i) + \lambda r_{\boldsymbol{\theta}}(\boldsymbol{x})}_{\triangleq J_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}_i)} \}$$

Note:

- The supervised L2 term $\|\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta}, \lambda) - \boldsymbol{x}_i\|^2$ is one of several options
- In practice, we alternate between updating $\boldsymbol{\theta}$, λ, and $\{\boldsymbol{\delta}_i\}$

Bi-level optimization details

Details of the $\boldsymbol{\theta}$ optimization:

- For fixed $\{\boldsymbol{\delta}_i\}$, define $\rho(\boldsymbol{\theta}) \triangleq \sum_{i=1}^n \max\left\{0, \max_{\boldsymbol{\delta}_i} \widehat{\beta}_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{\delta}_i) - \beta\right\}$
- Can show that

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \sum_{i=1}^n \frac{\partial^2 J_{\boldsymbol{\theta}}(\widehat{\boldsymbol{x}}_i; \boldsymbol{y}_i)}{\partial \boldsymbol{\theta} \partial \boldsymbol{x}^\top} \underbrace{\left[\frac{\partial^2 J_{\boldsymbol{\theta}}(\widehat{\boldsymbol{x}}_i; \boldsymbol{y}_i)}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\top}\right]^{-1} (\boldsymbol{x}_i - \widehat{\boldsymbol{x}}_i)}_{\triangleq \boldsymbol{\gamma}_i} + \xi \frac{\partial \rho(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

- To compute $\boldsymbol{\gamma}_i$, use CG (or similar) to solve $\left[\frac{\partial^2 J_{\boldsymbol{\theta}}(\widehat{\boldsymbol{x}}_i; \boldsymbol{y}_i)}{\partial \boldsymbol{x} \partial \boldsymbol{x}^\top}\right]\boldsymbol{\gamma}_i = \boldsymbol{x}_i - \widehat{\boldsymbol{x}}_i$
- Use auto-differentiation for gradients & Hessians (via JAX or FuncTorch)

Summary of each $\boldsymbol{\theta}$ update:

1. For each i, use PGD to compute $\widehat{\boldsymbol{x}}_i = \arg\min_{\boldsymbol{x}} \{\ell(\boldsymbol{x}; \boldsymbol{y}_i) + \lambda r_{\boldsymbol{\theta}}(\boldsymbol{x})\}$
2. For each i, use CG to compute $\boldsymbol{\gamma}_i$ above
3. Take gradient step $\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta} - \mu\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ using Adam

Numerical experiments

Inverse problem:

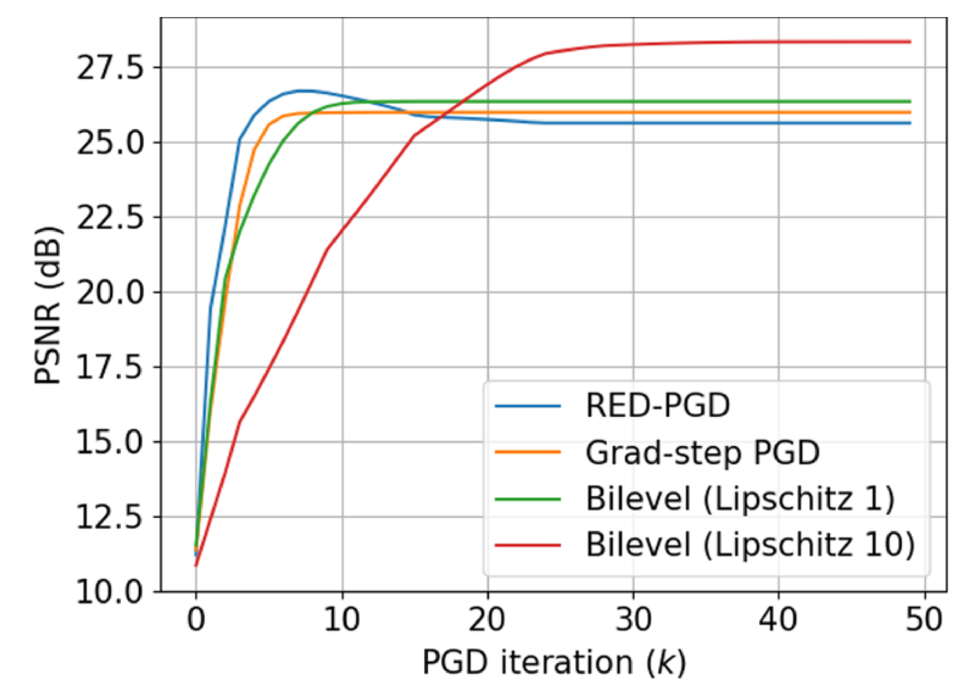
- Compressive sensing: Recover $\boldsymbol{x}$ from $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$
- $\boldsymbol{x} \in \mathbb{R}^d$ is a $d = 180^2$-pixel grayscale image from BSD400
  - 370 training images (augmented to 1480 via rotation & flipping)
  - 30 testing images
- $\boldsymbol{A} \in \mathbb{R}^{\frac{d}{2} \times d}$ is a fast, structurally random matrix [9]

Deep regularizer:

- RED-style regularization: $r_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top(\boldsymbol{x} - \boldsymbol{h}_{\boldsymbol{\theta}}(\boldsymbol{x}))$
  - Our scheme constrains $\mathrm{Lip}(\nabla s_{\boldsymbol{\theta}})$ where $s_{\boldsymbol{\theta}}(\boldsymbol{x}) \triangleq \frac{1}{2}\|\boldsymbol{x}\|^2 - r_{\boldsymbol{\theta}}(\boldsymbol{x})$
- $\boldsymbol{h}_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}^d$ is a convolutional deep network
  - 3 layers, 16 channels, SiLU activation, LayerNorm
  - This "simple" network acts as a proof-of-concept
- Recovery used PGD with backtracking line-search to automatically tune τ
  - Note: we increase the stepsize τ on successful iterations

Results:

	PSNR	SSIM
RED	25.63	0.69
gradient-step	25.98	0.68
bi-level ($\beta = 1$)	26.34	0.71
bi-level ($\beta = 3$)	27.42	0.75
bi-level ($\beta = 10$)	**28.33**	**0.79**



Key points:

- As $\beta = \mathrm{Lip}(\nabla s)$ increases, the final-PSNR increases and the convergence speed decreases
- Setting $\beta = 1$ yields a convex regularizer
- The proposed bi-level schemes attain the highest PSNRs

References

[1] Y. Romano, M. Elad, and P. Milanfar, "The little engine that could: Regularization by denoising (RED)," *SIAM J. Imag. Sci.*, vol. 10, no. 4, pp. 1804–1844, 2017.

[2] E. T. Reehorst and P. Schniter, "Regularization by denoising: Clarifications and new interpretations," *IEEE Trans. Comput. Imag.*, vol. 5, pp. 52–67, Mar. 2019.

[3] S. Lunz, O. Öktem, and C.-B. Schönlieb, "Adversarial regularizers in inverse problems," *Proc. Neural Inf. Process. Syst. Conf.*, vol. 31, 2018.

[4] B. Amos, L. Xu, and J. Z. Kolter, "Input convex neural networks," in *Proc. Int. Conf. Mach. Learn.*, pp. 146–155, 2017.

[5] R. Cohen, Y. Blau, D. Freedman, and E. Rivlin, "It has potential: Gradient-driven denoisers for convergent solutions to inverse problems," in *Proc. Neural Inf. Process. Syst. Conf.*, vol. 34, pp. 18152–18164, 2021.

[6] S. Hurault, A. Leclaire, and N. Papadakis, "Gradient step denoiser for convergent plug-and-play," in *Proc. Int. Conf. on Learn. Rep.*, 2022.

[7] L. Bungert, R. Raab, T. Roith, L. Schwinn, and D. Tenbrinck, "CLIP: Cheap Lipschitz training of neural networks," in *Intl. Conf. Scale Space & Variational Methods in Comp. Vis.*, pp. 307–319, 2021.

[8] K. G. Samuel and M. F. Tappen, "Learning optimized MAP estimates in continuously-valued MRF models," in *Proc. IEEE Conf. Comp. Vision Pattern Recog.*, pp. 477–484, 2009.

[9] T. T. Do, L. Gan, N. H. Nguyen, and T. D. Tran, "Fast and efficient compressive sensing using structurally random matrices," *IEEE Trans. Signal Process.*, vol. 60, pp. 139–154, Jan. 2012.