Tuning-free Plug-and-Play Proximal Algorithm for Inverse Imaging Problems *(Supplementary Material)*

Kaixuan Wei [1] **Angelica Aviles-Rivero** [2] **Jingwei Liang** [3] **Ying Fu** [1] **Carola-Bibiane Schnlieb** [3] **Hua Huang** [1]

1. Outline

This supplementary material provides further details and results to support the content from the main paper. It is organized as follows.

- **Section 2:** In the interest of clarity, we provide further details on the implementation and our tuning-free plug-and-play (TFPnP) algorithm.
- **Section 3:** With the purpose of completeness, we present further insights of our TFPnP algorithm.
- **Section 4:** We give further details on the experimental setup, for the competing methods, for both applications compressive sensing MRI (CS-MRI) and phase-retrieval (PR).

2. More Implementation Details

In this section, we present additional implementation details of our TFPnP algorithm. It is divided in two key parts: i) the network architectures of the policy and value networks, and ii) the detailed training algorithm for policy learning.

2.1. Policy and Value Networks

The design principal of a policy/value network is to make it simple yet effective. For convenience, we follow (Huang et al., 2019) that uses residual structures similar to ResNet-18 (He et al., 2016) as the feature extractor in the policy and value networks, followed by fully-connected layers and activation functions to produce desired outputs. The network configuration of the feature extractor of the policy and value networks is shown in Table I. To make the policy aware of the problem settings, we followed the protocol from (Silver et al., 2016). That is, we concatenate extra feature planes as additional channels in the input, which include the (spatially constant) observation noise level, the sampling mask (for CS-MRI), and the number of the steps that have been taken so far in the optimization process. *It is worth noting that the extra computation cost of the policy network is marginal, compared with the iteration cost in the PnP-ADMM.*

2.2. Detailed Training Algorithm

The detailed training algorithm for policy learning is summarized in Algorithm 1. It requires an image dataset D, a degradation operator $g(\cdot)$, a state buffer B, initialized network parameters θ, ϕ, $\hat{\phi}$, learning rates lr_θ, lr_ϕ, and a weight parameter β for updating $\hat{\phi}$. We use the 17,125 resized images[1] with size 128×128 from the PASCAL VOC dataset (Everingham et al., 2014) as the image dataset D. To sample initial states s_0, we define the degradation operator $g(\cdot)$ as a composition of a observation/forward model and an initialization function. The observation/forward model maps the underlying image x to its observation y, while the initialization function generates the initial estimate x_0 from the observation y. For linear inverse problems, $g(\cdot)$ is typically defined by the composition of the forward operator and the adjoint operator of the problem. For example, in CS-MRI, $g(\cdot)$ is a composition of the partially-sampled Fourier transform and the inverse Fourier transform. For nonlinear inverse problem - phase retrieval, we employ the HIO algorithm (Fienup, 1982) as the initialization function.

We optimize networks using Adam optimizer with batch size 48 and 1500 training iterations. We start by setting the learning rates lr_θ, lr_ϕ for updating the policy π_θ and the value network V_ϕ^π to 3×10^{-4} and 10^{-3}. Then we reduced these values to 10^{-4} and 3×10^{-4} respectively at training iteration 1000. The value network learning makes use of a target value network, which is a soft copy of the value network itself. The parameter β for softly updating the target value network is set to 10^{-3}. For each training iteration, we alternate between collecting states (in a state buffer) from the environment with the current policy and updating the network parameters using policy gradients

[1]School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China [2]DPMMS, University of Cambridge, Cambridge, United Kingdom [3]DAMTP, University of Cambridge, Cambridge, United Kingdom. Correspondence to: Ying Fu <fuying@bit.edu.cn>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

[1]The policy/value network is trained on whole images rather than image patches used to train denoising networks.

Table I. Network configuration of the feature extractor of our policy and value networks. $3 \times 3,\ 64$ denote the kernel size and the number of the output feature maps. "layer$x$" is the building block of the residual network, which consists of two convolutional layers with batch normalization (BN) (Ioffe & Szegedy, 2015) and ReLU activation. The BN and ReLU are replaced by weight normalization (Salimans & Kingma, 2016) and TReLU (Xiang & Li, 2017) respectively in the value network.

LAYER NAME	FEATURE EXTRACTOR	OUTPUT SIZE
conv1	$3 \times 3, 64$	64×64
layer1	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	32×32
layer2	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	16×16
layer3	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	8×8
layer4	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	4×4
avgpool	4×4 average pool	1×1

Algorithm 1 Training algorithm

Require: Image dataset D, degradation operator $g(\cdot)$, state buffer B, initialized network parameters θ, ϕ, $\hat{\phi}$, learning rates lr_θ, lr_ϕ, weight parameter β for updating $\hat{\phi}$.

for each training iteration **do**
  sample initial state s_0 from D via $g(\cdot)$
  for environment step $t \in [0, N)$ **do**
    $a_t \sim \pi_\theta(a_t|s_t)$
    $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
    $B \leftarrow B \cup \{s_{t+1}\}$
    break if the boolean outcome of a_t equals to 1
  end for
  for each gradient step **do**
    sample states from the state buffer B
    $\theta_1 \leftarrow \theta_1 + lr_\theta \nabla_{\theta_1} J(\pi_\theta)$
    $\theta_2 \leftarrow \theta_2 + lr_\theta \nabla_{\theta_2} J(\pi_\theta)$
    $\phi \leftarrow \phi - lr_\phi \nabla_\theta L_\phi$
    $\hat{\phi} \leftarrow \beta\phi + (1-\beta)\hat{\phi}$
  end for
end for

from batches sampled from the state buffer B. Ten gradient steps are performed at each training iteration.

3. Algorithm Investigation

In this section, we conduct experiments to further investigate our TFPnP algorithm. We focus on three aspects of the algorithm: i) the choice of hyperparameters m, N and η, ii) the relationship between the Gaussian denoising performance and the PnP performance, and iii) the behavior of the learned policy (*i.e.* how do the generated internal parameters look like?)

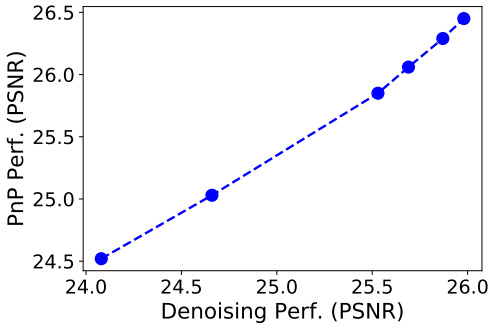

Figure I. Relationship between Gaussian denoising performance and PnP performance. A better Gaussian denoiser is also a better denoiser prior.

3.1. Hyperparameter Analysis

Here, we discuss the choice of hyperparameters m, N and η in our TFPnP algorithm, where m denotes the number of iterations of the optimization involved in the transition function p; N is the max time step to run the policy; η sets the degree of penalty defined in the reward function. Table II shows the results of learned policies trained with different hyperparameter settings (m, N and η) for CS-MRI. These results are divided into two groups (separated by the "midrule") to analyze the effects of (m, N) and η respectively. In the first group, we fix the value of η and change the value of m (N is varied with m such that the maximum number of iterations $m \times N$ is fixed to 30 to keep comparisons fair). We observe all these settings yield similar results. It is up to users' preference to choose m: larger m leads to coarser control of the terminal time, but only requires less rounds of the decision making. In the second group, we keep (m, N) constant while manipulating the value of η. We find η serves as a key parameter to encourage the early stopping behavior. With $\eta = 0$, the policy would not learn to early stop the optimization process, whilst large η would make the policy stop too early.

3.2. Is a Better Gaussian Denoiser a Better Denoiser Prior?

To reveal the relationship between the Gaussian denoising performance and the PnP performance, we train a series of denoising networks with incremental Gaussian denoising performance (by adjusting the number of filters of networks). Then we incorporate these denoisers into the PnP-ADMM framework, to evaluate their PnP performance on the CS-MRI application. The denoising strength/penalty parameters and the terminal time are exhaustively searched to maximize PSNR, such that the impact of internal parameters is excluded. We illustrate this in Fig I, in which we show the intrinsic relation between the Gaussian denoising and PnP performance.

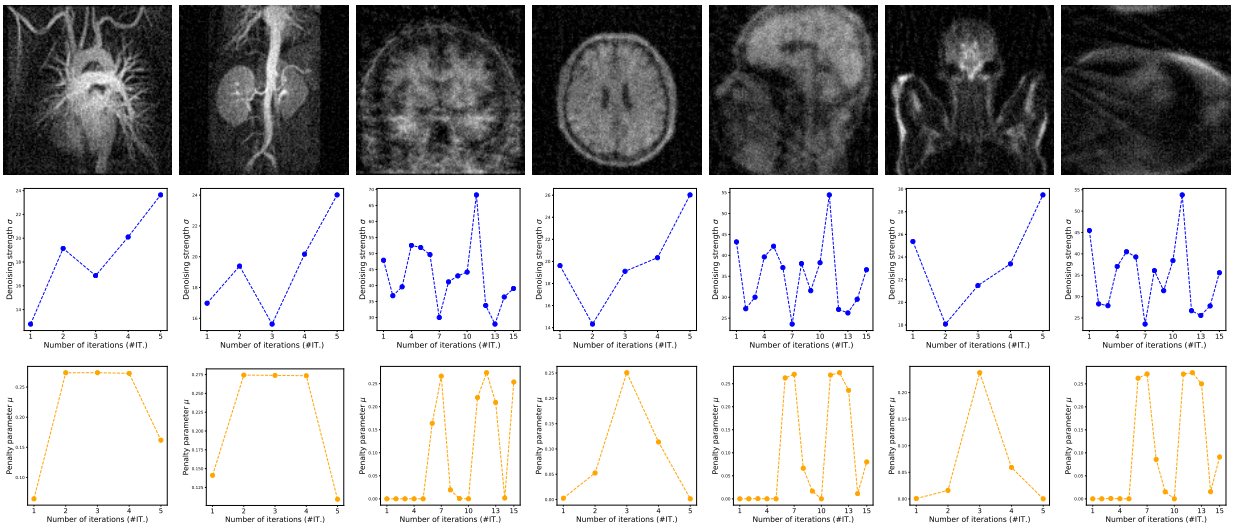

Figure II. Behaviors of our learned policy for CS-MRI on medical images. *First row*: the initial estimate x_0 of the underlying image; *Second row*: the predicted denoising strength σ by our learned policy; *Third row*: the predicted penalty parameters μ by our learned policy. *Our learned policy customizes different internal parameters with respect to different images, meanwhile the generated parameters change adaptively across iterations of the optimization process.*

Table II. Comparisons of learned policies trained with different hyperparameters for CS-MRI on seven widely used medical images under various acceleration factors (x2/x4/x8) and noise level 15. We show both PSNR and the number of iterations (#IT.) used to induce results.

	×2		×4		×8	
(m, N, η)	PSNR	#IT.	PSNR	#IT.	PSNR	#IT.
(3, 10, 0.05)	30.26	3.0	28.59	9.9	26.27	16.7
(10, 3, 0.05)	30.32	10.0	28.58	10.0	26.39	10.0
(5, 6, 0.05)	30.33	5.0	28.42	5.0	26.44	15.0
(5, 6, 0)	30.30	27.1	28.60	29.3	26.40	30.0
(5, 6, 0.1)	30.34	5.0	28.44	5.0	26.29	10.7
(5, 6, 0.25)	30.34	5.0	28.37	5.0	25.53	5.0

3.3. Behaviors of Learned Policy

In the Section 4.2 of the main paper, we find our learned policy sometimes even surpasses the oracle policy tuned via the inaccessible ground truth. We attribute this phenomena to the adaptive parameters over iterations generated automatically in our TFPnP algorithm, as the adaptive penalization has been demonstrated to result in faster convergence in the convex setting (Xu et al., 2017). We give a graphical illustration, in Fig. II, of our learned policy, *i.e.* to visualize its behavior (generated parameters) towards different images.

4. Experimental Setup

In this section, we detail the experimental setup for the competing methods on both CS-MRI and PR applications. All the methods were tested on a machine with NVIDIA GTX 1080Ti GPU, Intel(R) Core(TM) i7-7700K CPU of 4.2GHz and 16 GB RAM. We note all parameters involved in the competing algorithms were manually tuned optimally or automatically chosen as described in the reference paper. The specific parameter setting of each methods is presented next

4.1. Compressive Sensing MRI

On CS-MRI application, the competing method includes RecPF (Yang et al., 2010), ADMMNet (Yang et al., 2016), ISTANet (Zhang & Ghanem, 2018), BM3D-MRI (Eksioglu, 2016) and IRCNN (Zhang et al., 2017). For RecPF, the two parameters "alpha" and "beta" are set to 2.5 and 3.5×10^{-2} respectively. For ADMMNet and ISTANet, we use the default configurations and train these networks from scratch using the same dataset as ours. For BM3D-MRI, we set the "final noise level (the denoising strength in the last iteration)" as two times of the measurement noise level. For IRCNN, the "final noise level" is specified as the measurement noise level.

4.2. Phase Retrieval

On PR application, the competing method includes HIO (Fienup, 1982), WF (Candes et al., 2014), DOLPHIn (Mairal et al., 2016), SPAR (Katkovnik, 2017), BM3D-prGAMP (Metzler et al., 2016) and prDeep (Metzler et al., 2018). All algorithms are initialized by the vector of ones. The HIO algorithm is run for 200 iterations and its parameter "beta" is assigned to 0.96. The WF algorithm is also run for 200 iterations and its parameters "tau0" and "gamma" are chosen as 10 and 0.05. The BM3D-prGAMP is run for 100

iterations, whose parameter "wvar" is set to the standard deviation of the measurement noise. Other algorithms, *i.e.* DOLPHIn, SPAR and prDeep, use their default parameters.

References

Candes, E., Li, X., and Soltanolkotabi, M. Phase retrieval via wirtinger flow: Theory and algorithms. *IEEE Transactions on Information Theory*, 61, 07 2014.

Eksioglu, E. M. Decoupled algorithm for mri reconstruction using nonlocal block matching model: Bm3d-mri. *Journal of Mathematical Imaging and Vision*, 56(3):430–440, 2016.

Everingham, M., Eslami, S., Van Gool, L., Williams, C., Winn, J., and Zisserman, A. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111, 01 2014.

Fienup, J. R. Phase retrieval algorithms: a comparison. *Applied Optics*, 21(15):2758–2769, 1982.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Huang, Z., Heng, W., and Zhou, S. Learning to paint with model-based deep reinforcement learning. In *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Katkovnik, V. Phase retrieval from noisy data based on sparse approximation of object phase and amplitude. *arXiv preprint arXiv:1709.01071*, 2017.

Mairal, Julien, Tillmann, Andreas, M., Eldar, Yonina, and C. Dolphin-dictionary learning for phase retrieval. *IEEE Transactions on Signal Processing*, 2016.

Metzler, C., Schniter, P., Veeraraghavan, A., et al. prdeep: Robust phase retrieval with a flexible deep network. In *International Conference on Machine Learning (ICML)*, pp. 3498–3507, 2018.

Metzler, C. A., Maleki, A., and Baraniuk, R. G. Bm3d-prgamp: Compressive phase retrieval based on bm3d denoising. In *IEEE International Conference on Image Processing*, 2016.

Salimans, T. and Kingma, D. P. Weight normalization: a simple reparameterization to accelerate training of deep neural networks. *Advances in Neural Information Processing Systems (NIPS)*, pp. 901–909, 2016.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

Xiang, S. and Li, H. On the effects of batch and weight normalization in generative adversarial networks. *arXiv preprint arXiv:1704.03971*, 2017.

Xu, Y., Liu, M., Lin, Q., and Yang, T. Admm without a fixed penalty parameter: Faster convergence with new adaptive penalization. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1267–1277, 2017.

Yang, J., Zhang, Y., and Yin, W. A fast alternating direction method for tvl1-l2 signal reconstruction from partial fourier data. *IEEE Journal of Selected Topics in Signal Processing*, 4(2):288–297, 2010.

Yang, Y., Sun, J., Li, H., and Xu, Z. Deep admm-net for compressive sensing mri. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 10–18. 2016.

Zhang, J. and Ghanem, B. Ista-net: Interpretable optimization-inspired deep network for image compressive sensing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Zhang, K., Zuo, W., Gu, S., and Zhang, L. Learning deep cnn denoiser prior for image restoration. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.