Attacking inverse problems with deep learning

Patrick Pérez
valeo.ai

Journées Calcul et Apprentissage, Lyon
25 April 2019
Inverse problems

High level formulation

• Forward process  \( y \rightarrow x = g(y) \)

  to “invert”  \( \hat{y}(x) \in g^{-1}(x) ? \)

• Optimization-based inversion, with prior to make it well-posed

  \( \forall x, \hat{y}(x) \in \arg \min_y \left( \text{Loss}(x, g(y)) + \text{Prior}(y) \right) \)

  \( E(x,y;\theta) \)

• Model bricks: given (e.g., physics), engineered, and/or learned
Classic examples

Signal enhancement or completion

• Forward process: degradation and/or masking
• Prior: spatial/temporal regularity

\[
\min_y \left( \| x - g(y) \|_2^2 + \mu \| \nabla y \|_p^q \right) \quad \text{s.t.} \quad y|_{\partial D} = y^* 
\]

Signal recovery

• Linear forward process: atom composition, random measurements
• Prior: sparsity

\[
\min_y \left( \| x - Ay \|_2^2 + \mu \| y \|_1 \right) 
\]
Inverse graphics

• CGI: from scene model to photoreal images
• Inverse rendering: from real images to scene model (“graphic code”)
• Application: understanding, editing, AR/VR, personalized models

[Tewari 2017]
Less classic examples

“Neural” inverse problems

• Deep features: from signal to representations through feedforward neural net

• Inverting: from neural activations to NN input

\[
\min_y \left( \|x - \phi_{\ell_0}(y)\|_F^2 + \mu \|\nabla y\|_1 \right)
\]

• Application: visualization, inspection, editing in feature domain
Less classic examples

“Neural” inverse problems

[Upchich 2017]
Learning and inverse problems?

Learn the model, solve by optimization

• Forward process, prior and loss can be (partly) learned
• Examples: blind deconvolution, trained MRFs
• Inference with a classic solver, *iterative* and *generic*

\[ \forall x, \text{Solver}(x) = \text{Iter}^\infty(x; y^0) \approx \hat{y}(x) \]

Train a direct solver

• From \[ \forall x, \hat{y}(x) \in \arg \min_y E(x, y) \]

• To \[ \forall x \sim P_X, f(x; W) \approx \hat{y}(x) \text{ or } \approx g^{-1}(x) \]
Neural inversion

Train a DNN to regress solution/inverse for plausible inputs

\[ \forall x \sim P_X, \text{DNN}(x; W) \approx \hat{y}(x) \text{ or } \approx g^{-1}(x) \]

• Fixed complexity
• Possibly way faster
• Flexible in various ways
• Differentiable and reusable

Architecture?
Training?
Architectures

Your favorite DNN

- Exploit popular architectures, possibly pre-trained
- Popular convolutional and recurrent neural nets

Unrolling

- Mimic (possibly loosely) structure of iterative solver
- Each $\text{[non-linear \circ linear]}$ iteration becomes a trainable neural layer
- Fixed, smaller number of “iterations”
- But, way more freedom exploited through training
Training

**Fully-supervised** – Using Solver, reconstruction loss

\[
x^{(n)} \sim \mathbb{P}_X, \quad \min_W \sum_{n=1}^N \| \text{Solver}(x^{(n)}) - \text{DNN}(x^{(n)}; W) \|^2_2
\]

**Self-supervised** – Possibly only synthetic data, reconstruction loss

\[
y^{(n)} \sim \mathbb{P}_Y, \quad \min_W \sum_{n=1}^N \| y^{(n)} - \text{DNN}(g(y^{(n)}); W) \|^2_2
\]

**Unsupervised** – Original objective function as training loss

\[
x^{(n)} \sim \mathbb{P}_X, \quad \min_W \sum_{n=1}^N E(x^{(n)}, \text{DNN}(x^{(n)}; W); \theta)
\]
Unsupervised training

Unsupervised – Original objective function as training loss

\[ x^{(n)} \sim P_X, \quad \min_W \sum_{n=1}^{N} \text{Loss}(x^{(n)}, g \circ \text{DNN}(x^{(n)}, W)) + \text{Prior}(\text{DNN}(x^{(n)}, W)) \]
Sparse coding: LISTA [Gregor 2010]  
\[ \min_{y} \left( 0.5\|x - Ay\|^2_2 + \mu\|y\|_1 \right) \]

Iterative solver: Iterative Soft Thersholding Alg. (ISTA)

Iterative solver equation:
\[ \text{Iter}(y) = \sigma_{2\mu\alpha}( (I - \alpha A^\top A)y + \alpha A^\top x ) \]

Learned neural layer (residual block and skip connection)

Learned neural layer equation:
\[ f_k(y) = \sigma \left( (I - V_k)y + W_k^\top x \right), \quad \mathcal{W} = \{ (W_k, V_k) \}_k \]

- Fully supervised training
- The deeper, the better the approximation
- Modest speed-up
Unsupervised neural inversion

Personalized 3D face model: Tewari et al. 2017
Encoder-decoder with differentiable rendering layer

Encoder-decoder with “perceptual loss”

Flexible style transfer: Puy & Pérez 2019
Unrolling descent, run time flexibility
Photo-real face rendering

\[ y = (\text{pose, lighting, albedo, shape, expression, correctives, details}) \]

\[ y \in \mathbb{R}^{257} \rightarrow g(y) = \]

[Garrido 2016]
Invert rendering
to obtain animatable personnalized 3D rig

\[
y = (\text{pose}, \text{lighting}, \text{albedo}, \text{shape}, \text{expression}, \text{correctives}, \text{details})
\]

\[
x \rightarrow y \in \mathbb{R}^{257}
\]

[Garrido 2016]
Invert rendering

to obtain animatable personalized 3D rig

\[ \mathbf{y} = (\text{pose}, \text{lighting}, \text{albedo}, \text{shape}, \text{expression}, \text{correctives}, \text{details}) \]

[Garrido 2016]
Fast DNN solver

\[ \text{Loss} = \| \text{observed} \| - \| \text{rendered} \|^2 + \| \text{2D landmarks} \| - \| \text{correspondences} \|^2 \]

[Tewari 2017-2018]
"Artistic" style transfer
retain structure, imitate texture

$$\min_y \left( \| x - \phi_{\ell_0}(y) \|_F^2 + \lambda \sum_{\ell \in \mathcal{L}_{\text{sty}}} \| G_\ell - \phi_\ell(y)^T \phi_\ell(y) \|_F^2 \right)$$

[Galys 2015-2016]
“Artistic” style transfer
retain structure, imitate texture

$$\min_y \left( \|x - \phi_{\ell_0}(y)\|_F^2 + \lambda \sum_{\ell \in \mathcal{L}_{\text{sty}}} \|G_\ell - \phi_{\ell}(y)^T \phi_{\ell}(y)\|_F^2 \right)$$

[Gatys 2015-2016]
“Artistic” style transfer
retain structure, imitate texture

$$\min_y \left( \| x - \phi_{l_0}(y) \|_F^2 + \lambda \sum_{\ell \in \mathcal{L}_{\text{sty}}} \| G_\ell - \phi_\ell(y)^T \phi_\ell(y) \|_F^2 \right)$$

[Image of paintings and diagram]

[Refs: Gatys 2015-2016]
Fast artistic style transfer

[Ulyanov 2016, Johnson 2016] and successors

• Convolutional encoder-decoder architectures
• Unsupervised training for specified paintings

[Ulyanov 2017]
Fast artistic style transfer

[Ulyanov 2016, Johnson 2016] and successors

• Convolutional encoder-decoder architectures
• Unsupervised training for specified paintings

[Ulyanov 2017]
Fast flexible style transfer [Puy 2019]

Unrolling (part of) gradient descent

\[
E(x, y) = \|x - \phi_0(y)\|_F^2 + \mu \|\nabla y\|_1 \\
+ \sum_{\ell \in \mathcal{L}_{sty}} \lambda_\ell \|G_\ell - \phi_\ell(y)\top \phi_\ell(y)\|_F^2
\]

- One layer mimics one step \( y \leftarrow y - \alpha \nabla E_{sty}(x, y) \)

\[
y_k = y_{k-1} - f_k(x, y_{k-1}; \mathcal{W}_k, \{\lambda_\ell, G_\ell\}_{\ell \in \mathcal{L}_{sty}})
\]

- Unsupervised training

modifiable at \textit{run time}
Runtime restructuring

Choose style, mix styles, tune stylization intensity or scale
Runtime restructuring

Add new regularizers via proximal operator, e.g. for photorealism

\[ y_k = y_{k-1} - \text{Prox}_\Omega[f_k(x, y_{k-1})] \]
Runtime restructuring

Add new regularizers via proximal operator, e.g. for photorealism

\[ y_k = y_{k-1} - \text{Prox}_\Omega[f_k(x, y_{k-1})] \]
Runtime restructuring

Add new regularizers via proximal operator, e.g. for photorealism

\[ y_k = y_{k-1} - \text{Prox}_\Omega[f_k(x, y_{k-1})] \]
Perspectives

• Encoder-decoder view of unsupervised inversion: make forward process (partly) trainable, e.g. [Tewari 2018]

• **Invertible NN** [Ardizzone 2019] trained to mimic $g$, inversion for free
• Multiple pre-images?
• Inversion of state-of-art GAN?
• Encoder-decoder view of unsupervised inversion: make forward process (partly) trainable, e.g. [Tewari 2018]

• **Invertible NN** [Ardizzone 2019] trained to mimic $g$, inversion for free

• Multiple pre-images?

• Inversion of state-of-art GAN?

• Mathematical properties of neural solver? Robustness, generability
Learn an invertible neural net (INN) to mimic forward process

\[ \forall y \sim \mathbb{P}_Y, \ h(y; \mathcal{W}) = \left[ h_x(y; \mathcal{W}), h_z(y; \mathcal{W}) \right] \approx g(y) \]

Invert it to get a posterior distribution

\[ \forall x, z \sim N(0, I) \rightarrow h^{-1}([x, z]; \mathcal{W}) \approx y \mid x \]

Bi-directional supervised/unsupervised training

- Self-supervised training of forward mimicking using simulation
- Unsupervised on distributions of \( y \) and \( z \), and independence of \( x \) vs. \( z \)
Conclusion

Neural solvers for inverse problems

• fast, specialized, differentiable, possibly unsupervised, flexible
• can go beyond original model by learning
• applies to other optimization-based/variational problems
References


