Deep learning-based posterior inference for inverse problems

Deep Ray
Department of Aerospace & Mechanical Engineering
University of Southern California

Email: deepray@usc.edu
Website: deepray.github.io

Jointly with Harisankar Ramaswamy, Dhruv Patel, Assad Oberai

Supported by ARO grant W911NF2010050, NRL grant N00173-19-P-1119, AIER-USC, CARC-USC

USC Viterbi
School of Engineering

Conference on PDE and numerical analysis
TIFR-CAM, April 28th, 2022

Outline

- Inverse problems and Bayesian inference
- Deep neural networks
- Conditional generative adversarial networks (cGANs)
- Deep posteriors

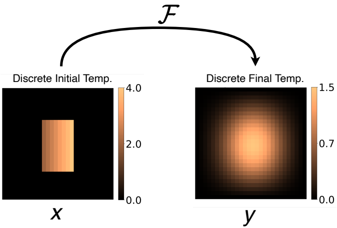
Consider a forward problem

$$\mathcal{F} : \boldsymbol{x} \in \Omega_x \mapsto \boldsymbol{y} \in \Omega_y, \quad \Omega_x \in \mathbb{R}^{N_x}, \ \Omega_y \in \mathbb{R}^{N_y}$$

For example, the heat conduction PDE model for temperature field u

$$\begin{aligned} \frac{\partial u(\boldsymbol{s}, t)}{\partial t} - \nabla \cdot (\kappa(\boldsymbol{s}) \nabla u(\boldsymbol{s}, t)) &= f(\boldsymbol{s}), \qquad && \forall\, (\boldsymbol{s}, t) \in (0,1)^2 \times (0, T] \\ u(\xi, 0) &= u_0(\boldsymbol{s}), && \forall\, \boldsymbol{s} \in (0,1)^2 \\ u(\xi, t) &= 0, && \forall\, \boldsymbol{s} \in \partial(0,1)^2 \times (0, T] \end{aligned}$$

Forward problem: Given $u_0(\boldsymbol{s})$ determine $u(\boldsymbol{s}, T)$



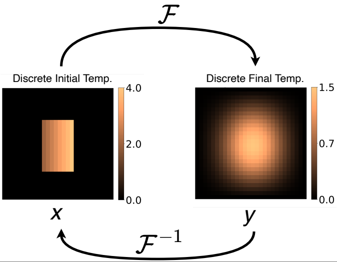
Consider a forward problem

$$\mathcal{F} : \boldsymbol{x} \in \Omega_x \mapsto \boldsymbol{y} \in \Omega_y, \quad \Omega_x \in \mathbb{R}^{N_x},\ \Omega_y \in \mathbb{R}^{N_y}$$

For example, the heat conduction PDE model for temperature field u

$$\begin{aligned} \frac{\partial u(\boldsymbol{s}, t)}{\partial t} - \nabla \cdot (\kappa(\boldsymbol{s}) \nabla u(\boldsymbol{s}, t)) = f(\boldsymbol{s}), &\qquad \forall\ (\boldsymbol{s}, t) \in (0,1)^2 \times (0, T] \\ u(\xi, 0) = u_0(\boldsymbol{s}), &\qquad \forall\ \boldsymbol{s} \in (0,1)^2 \\ u(\xi, t) = 0, &\qquad \forall\ \boldsymbol{s} \in \partial(0,1)^2 \times (0, T] \end{aligned}$$

Inverse problem: Given $u(\boldsymbol{s}, T)$ infer $u_0(\boldsymbol{s})$



Forward and inverse problems

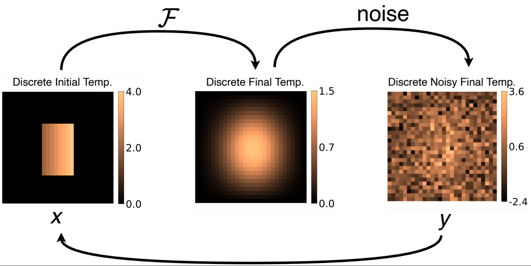
Consider a forward problem

$$\mathcal{F} : \boldsymbol{x} \in \Omega_x \mapsto \boldsymbol{y} \in \Omega_y, \quad \Omega_x \in \mathbb{R}^{N_x}, \ \Omega_y \in \mathbb{R}^{N_y}$$

For example, the heat conduction PDE model for temperature field u

$$\begin{aligned} \frac{\partial u(\boldsymbol{s}, t)}{\partial t} - \nabla \cdot (\kappa(\boldsymbol{s}) \nabla u(\boldsymbol{s}, t)) &= f(\boldsymbol{s}), & \forall\, (\boldsymbol{s}, t) \in (0,1)^2 \times (0, T] \\ u(\xi, 0) &= u_0(\boldsymbol{s}), & \forall\, \boldsymbol{s} \in (0,1)^2 \\ u(\xi, t) &= 0, & \forall\, \boldsymbol{s} \in \partial(0,1)^2 \times (0, T] \end{aligned}$$

Inverse problem: Given noisy $u(\boldsymbol{s}, T)$ infer $u_0(\boldsymbol{s})$



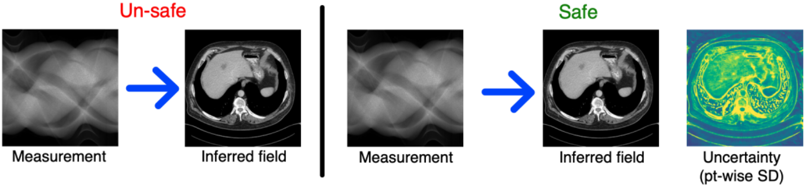
Forward and inverse problems

Challenges with inverse problems:

- Inverse map is not well posed.
- Noisy measurements.
- Need to encode prior knowledge about $\boldsymbol{x}$.

Uncertainty in inferred field critical for applications with high-stake decisions.

Example: Medical imaging to detect liver lesions



[Adler et al., 2018]

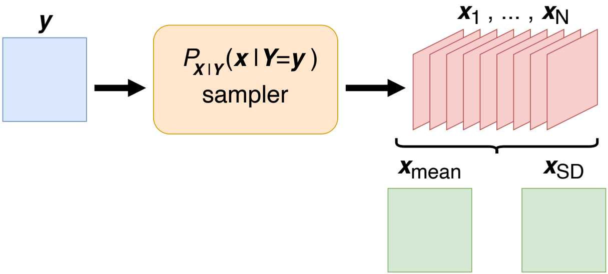
Bayesian framework

Assume $\boldsymbol{x}$ and $\boldsymbol{y}$ are modelled by random variables $\boldsymbol{X}$ and $\boldsymbol{Y}$.

AIM: Given a measurement $\boldsymbol{Y} = \boldsymbol{y}$ approximate the conditional (posterior) distribution

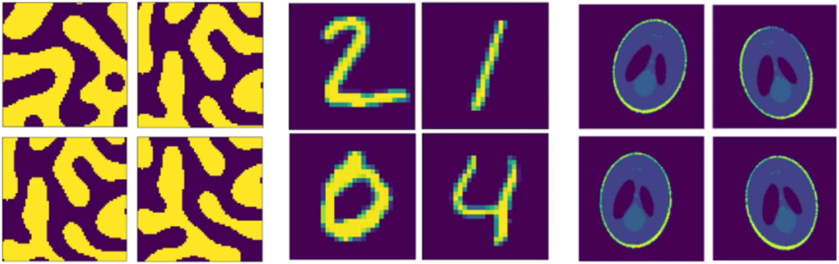
$$P_{\boldsymbol{X}|\boldsymbol{Y}}(\boldsymbol{x}|\boldsymbol{Y} = \boldsymbol{y})$$

and sample from it.



- Posterior sampling techniques, such as Markov Chain Monte Carlo, are prohibitively expensive when N_x is large.
- Characterization of priors for complex data

Examples of prior data snapshots for $\boldsymbol{x}$:



Representing this data using simple distributions is hard!

Resolve both issues using deep learning

A neural network is a parametrized mapping

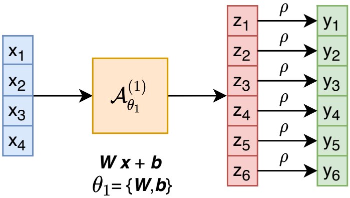
$$NN_{\boldsymbol{\theta}} : \Omega_x \to \Omega_y$$

typically formed by alternating composition

$$NN_{\boldsymbol{\theta}} := \rho \circ \mathcal{A}^{(L+1)}_{\boldsymbol{\theta}_{L+1}} \circ \rho \circ \mathcal{A}^{(L)}_{\boldsymbol{\theta}_L} \circ \rho \circ \mathcal{A}^{(L-1)}_{\boldsymbol{\theta}_{L-1}} \circ \cdots \circ \rho \circ \mathcal{A}^{(1)}_{\boldsymbol{\theta}_1}$$

where

$$\begin{aligned} \boldsymbol{\theta} = \{\boldsymbol{\theta}_k\}_{k=1}^{L} &\longrightarrow \text{ trainable weights and biases of the network} \\ \mathcal{A}^{(k)}_{\boldsymbol{\theta}_k} &\longrightarrow \text{ parametrized affine transformation} \\ \rho &\longrightarrow \text{ non-linear activation function} \end{aligned}$$



Deep neural network

A neural network is a parametrized mapping

$$NN_{\boldsymbol{\theta}} : \Omega_x \to \Omega_y$$

typically formed by alternating composition

$$NN_{\boldsymbol{\theta}} := \rho \circ \mathcal{A}^{(L+1)}_{\boldsymbol{\theta}_{L+1}} \circ \rho \circ \mathcal{A}^{(L)}_{\boldsymbol{\theta}_L} \circ \rho \circ \mathcal{A}^{(L-1)}_{\boldsymbol{\theta}_{L-1}} \circ \cdots \circ \rho \circ \mathcal{A}^{(1)}_{\boldsymbol{\theta}_1}$$

where

$$\begin{aligned} \boldsymbol{\theta} = \{\boldsymbol{\theta}_k\}_{k=1}^{L} &\longrightarrow \text{ trainable weights and biases of the network} \\ \mathcal{A}^{(k)}_{\boldsymbol{\theta}_k} &\longrightarrow \text{ parametrized affine transformation} \\ \rho &\longrightarrow \text{ non-linear activation function} \end{aligned}$$

Usage:

- Let $\boldsymbol{x}$ and $\boldsymbol{y}$ are related in some manner, say $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$.
- We are only given $\mathcal{S} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$.
- $NN_{\boldsymbol{\theta}}$ can be used to learn $\boldsymbol{f}$.

Consider a suitable measure

$$\mu : \Omega_x \times \Omega_y \times \Omega_x \times \Omega_y \to \mathbb{R}$$

s.t. $\mu(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{x}, \underbrace{NN_{\boldsymbol{\theta}}(\boldsymbol{x})}_{=\widehat{\boldsymbol{y}}})$ is the error/discrepancy between $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}$ and $(\boldsymbol{x}, \widehat{\boldsymbol{y}})$.

Define the loss/objective function

$$\Pi(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \mu(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{x}_i, NN_{\boldsymbol{\theta}}(\boldsymbol{x}_i))$$

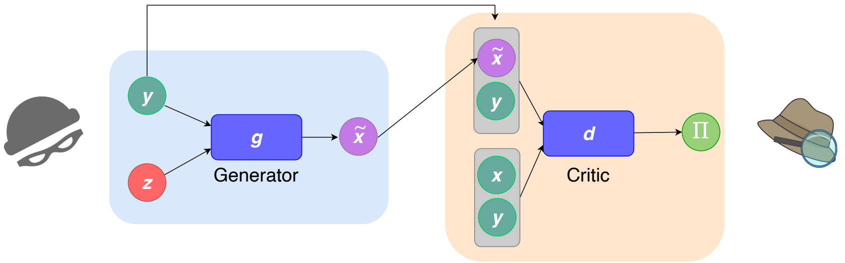
Solve the non-convex optimization problem

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \Pi(\boldsymbol{\theta})$$

Then $NN_{\boldsymbol{\theta}^*} \approx \boldsymbol{f}$

Also need to tune network hyper-parameters:
- Width
- Depth (L)
- Activation function ρ
- Optimizer
- Loss function
- Dataset

- Learning distributions conditioned on another field.
- Comprises two neural networks, $\boldsymbol{g}$ and d.
- Flipping role of $\boldsymbol{x}$ and $\boldsymbol{y}$ for inverse problems.



Generator network:

- $\boldsymbol{g} : \Omega_z \times \Omega_y \to \Omega_x$.
- Latent variable $\boldsymbol{z} \sim P_Z$, $N_z \ll N_x$.
- $(\boldsymbol{x}, \boldsymbol{y}) \sim P_{XY}$

Critic network:

- $d : \Omega_x \times \Omega_y \to \mathbb{R}$.
- $d(\boldsymbol{x}, \boldsymbol{y})$ large for real $\boldsymbol{x}$, small otherwise.

- Objective function

$$\Pi(\boldsymbol{g}, d) = \mathop{\mathbb{E}}_{\substack{(\boldsymbol{x},\boldsymbol{y})\sim P_{XY} \\ \boldsymbol{z}\sim P_Z}} \left[d(\boldsymbol{x}, \boldsymbol{y}) - d\big(\boldsymbol{g}(\boldsymbol{z}, \boldsymbol{y}), \boldsymbol{y}\big) \right]$$

- $\boldsymbol{g}$ and d determined (with constraint $\|d\|_{\text{Lip}} \leq 1$) through

$$(\boldsymbol{g}^*, d^*) = \arg\min_{\boldsymbol{g}} \ \arg\max_{d} \Pi(\boldsymbol{g}, d)$$

- Adler et al. (2018) proved that the minmax problem is equivalent to

$$\boldsymbol{g}^*(., \boldsymbol{y}) = \arg\min_{\boldsymbol{g}} W_1(P_{X|Y}, \boldsymbol{g}_{\#}(., \boldsymbol{y})P_Z) \quad \text{given } \boldsymbol{y} \sim p_Y$$

  where W_1 is the Wasserstein-1 distance.

- Convergence in W_1 implies weak convergence

$$\mathop{\mathbb{E}}_{\boldsymbol{x}\sim P_{X|Y}} [\ell(\boldsymbol{x})] = \mathop{\mathbb{E}}_{\boldsymbol{z}\sim P_Z} [\ell(\boldsymbol{g}^*(\boldsymbol{z}, \boldsymbol{y}))], \quad \forall\, \ell \in C_b(\Omega_X).$$

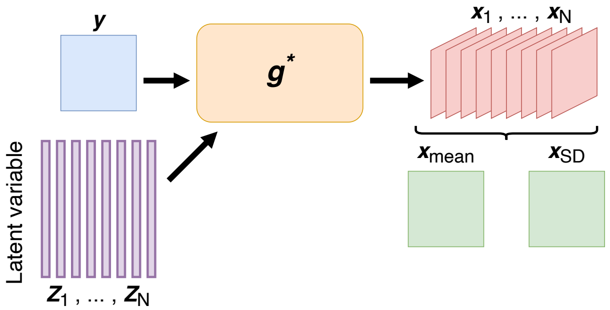
$\Longrightarrow$ conditional statistics converge!

Steps:

- Acquire samples $\mathcal{S}_x = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$, where $\boldsymbol{x}_i \sim P_X^{\text{prior}}$.
- Use forward map $\mathcal{F}$ to generate paired dataset

$$\mathcal{S} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), ..., (\boldsymbol{x}_N, \boldsymbol{y}_N)\} \quad \text{where} \quad \boldsymbol{y}_n = \mathcal{F}(\boldsymbol{x}_n) + \text{ noise.}$$

- Train a cGAN on $\mathcal{S}$
- For a new test measurement $\boldsymbol{y}$, generate samples using $\boldsymbol{g}^*$.
- Evaluate statistics using Monte Carlo



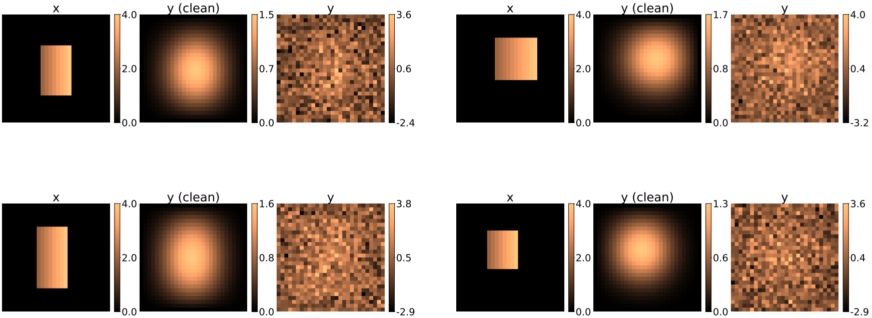
Consider the PDE

$$\begin{aligned} \frac{\partial u(\boldsymbol{s},t)}{\partial t} - \nabla \cdot (\kappa(\boldsymbol{s}) \nabla u(\boldsymbol{s},t)) &= f(\boldsymbol{s}), && \forall\, (\boldsymbol{s},t) \in (0,1)^2 \times (0,T] \\ u(\xi,0) &= u_0(\boldsymbol{s}), && \forall\, \boldsymbol{s} \in (0,1)^2 \\ u(\xi,t) &= 0, && \forall\, \boldsymbol{s} \in \partial(0,1)^2 \times (0,T] \end{aligned}$$

- $\boldsymbol{x}$: discrete initial temperature field.
- $\boldsymbol{y}$: noisy discrete final temperature field.
- $\mathcal{F}$: Finite difference solver for the PDE.
- We will assume a constant κ and f.

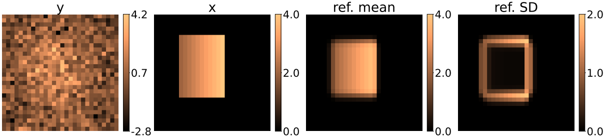
Assuming $\boldsymbol{x}$ to given by a rectangular inclusion and $N_x = N_y = 28 \times 28 = 784$

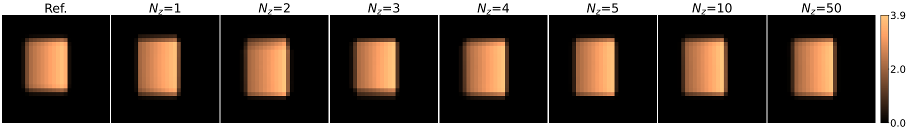
Training samples:

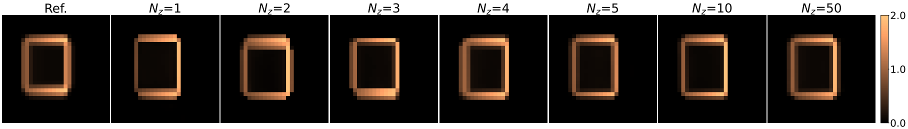


We never actually have clean $\boldsymbol{y}$!

Testing trained cGAN (statistics with 800 $\boldsymbol{z}$ samples)

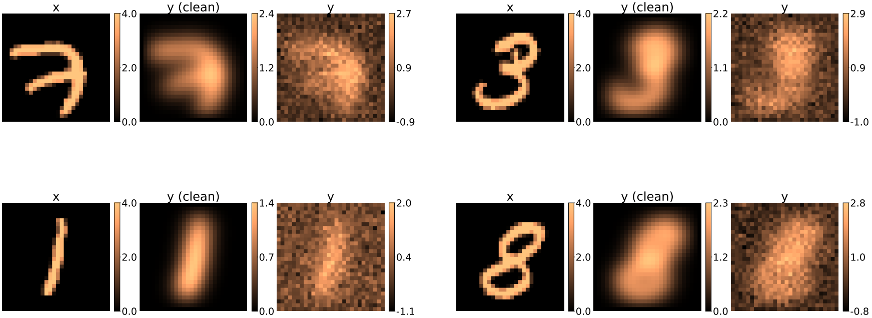






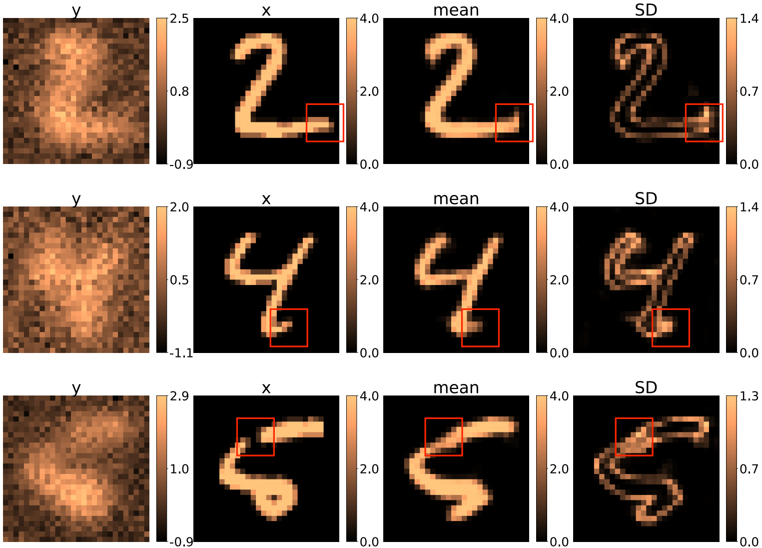
Assuming $\boldsymbol{x}$ to given by MNIST handwritten digits and $N_x = N_y = 28 \times 28 = 784$

Training samples:



Inferring initial condition: non-parametric prior

Testing trained cGAN (statistics with 800 ***z*** samples)



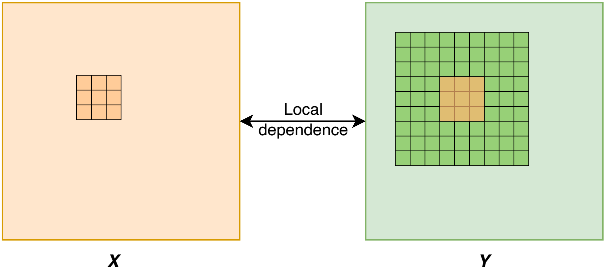
Generalization

Generalization: Network trained on Set A gives good predictions on another distinct Set B, possibility sampled from a different distribution.

We can prove[1] the following theorem on generalizability: Assume

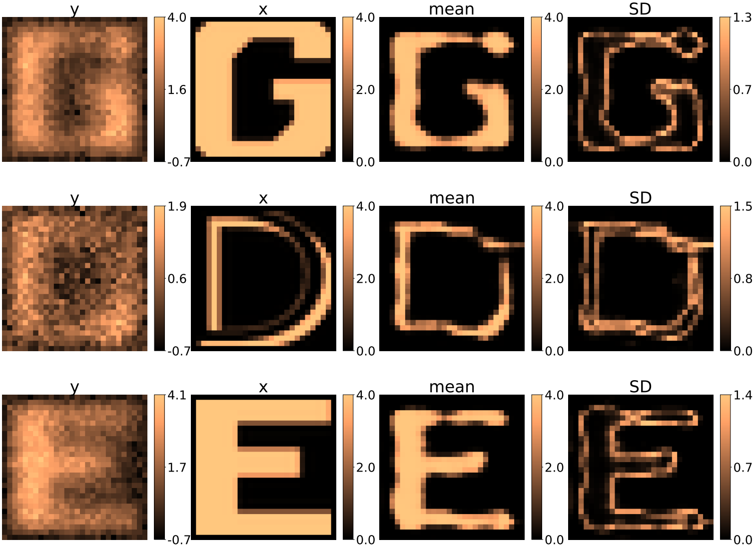
- The true (regularized) inverse map $\mathcal{F}^{-1}$ is spatially local.
- Set A and Set B contain samples with similar local spatial features.
- A cGAN train on Set A, and is also spatially local.

Then the cGAN can generalize well to Set B.



1: *The efficacy and generalizability of conditional GANs for posterior inference in physics-based inverse problems* (D. Ray, D. Patel, H. Ramaswamy, A. A. Oberai); preprint 2022.

cGAN trained on MNIST, tested on notMNIST (locally similar features)

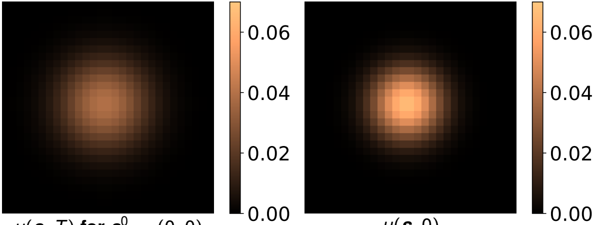


Locality of (regularized) inverse-heat equation:

- Consider final temperature as $T = 1$ as Gaussian bump

$$u(\boldsymbol{s}, T) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{|\boldsymbol{s} - \boldsymbol{s}^0|^2}{2\sigma^2}\right), \quad \sigma = 0.7,$$

- Solve inverse problem using FFT but killing higher-modes (hyper-diffusion).

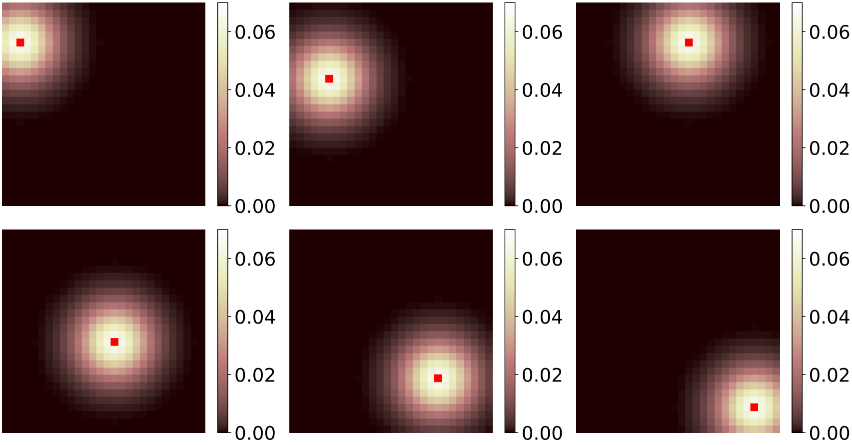


$u(\boldsymbol{s}, T)$ **for** $\boldsymbol{s}^0 = (0, 0)$ $\quad$ $u(\boldsymbol{s}, 0)$

- Move Gaussian center $\boldsymbol{s}^0$ and repeat.

Generalization

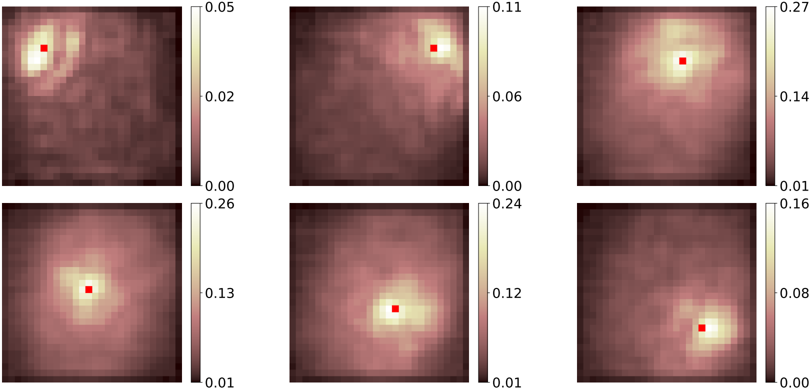
Vizualize $u(\boldsymbol{s}, 0)$ at fixed $\boldsymbol{s} = \boldsymbol{s}^1$ (marked in red) as $\boldsymbol{s}^0$ moved in 28×28 grid.



Influence of $u(\boldsymbol{s}, T)$ on $u(\boldsymbol{s}, 0)$ weakens as $\boldsymbol{s}^0$ moves away from $\boldsymbol{s}^1$.

Gradient of k-th component (marked in red) of $\boldsymbol{g}^*$ wrt input $\boldsymbol{y}$

$$\overline{\text{grad}}_k = \frac{1}{1000}\sum_{i=1}^{100}\sum_{j=1}^{10}\left|\frac{\partial g_k}{\partial \boldsymbol{y}}(\boldsymbol{z}^{(j)}, \boldsymbol{y}^{(i)})\right|, \ \boldsymbol{y}^{(i)} \sim P_Y, \quad \boldsymbol{z}^{(j)} \sim P_Z, \ 1 \le k \le N_X.$$



Gradient concentrated near k-th component of $\boldsymbol{y}$ $\implies$ locality of $\boldsymbol{g}^*$.

Conclusion

- What do we gain?
  - Ability to represent and encode complex prior data.
  - Dimension reduction since $N_z \ll N_x$.
  - Sampling from cGAN is quick and easy.
- Generalizability – training on smaller dataset.
- Algorithm has been tested for many other physic-based applications.