Solving PDEs Using Conditional Generative Models

Chinmay Hegde
October 24, 2019

Iowa State → NYU
Today’s talk

PDEs
Today’s talk

PDEs

Generative Models
Today’s talk

PDEs

Generative Models

Conditional
Materials design → exploring the space of \textit{plausible} microstructures

<table>
<thead>
<tr>
<th>PROCESS</th>
<th>STRUCTURE</th>
<th>PROPERTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process 1 (e.g., slow spinning)</td>
<td>Microstructure 1</td>
<td></td>
</tr>
<tr>
<td>Process 2 (e.g., hot roll to roll)</td>
<td>Microstructure 2</td>
<td></td>
</tr>
<tr>
<td>Process 3 (e.g., fast spinning)</td>
<td>Microstructure 3</td>
<td></td>
</tr>
<tr>
<td>Process 4 (e.g., cold roll to roll)</td>
<td>Microstructure 4</td>
<td></td>
</tr>
</tbody>
</table>

Expensive, time-consuming; need to create \textit{digital twin}
However, simulations are themselves extremely slow and costly.
However, simulations are themselves extremely slow and costly.

Ex: Two-component fluid mixtures that undergo a phase separation:
Solve the Cahn-Hilliard Equation $\rightarrow$ 4th order nonlinear PDE
However, simulations are themselves extremely slow and costly.

Ex: Two-component fluid mixtures that undergo a phase separation:
Solve the Cahn-Hilliard Equation → 4th order nonlinear PDE

Question: Are there alternatives to numerically solving PDEs?
Inverse problems

Canonical challenge in machine learning, scientific computing, engineering design

Parameters $\rightarrow$ Forward model $\rightarrow$ Observations

$y = A(x) + \text{noise}$.

Goal: Given $y$ and (possibly) $A$, recover an estimate of $x$. 
Inverse problems

Canonical challenge in machine learning, scientific computing, engineering design

\[ y = \mathcal{A}(x) + \text{noise}. \]
Inverse problems

Canonical challenge in machine learning, scientific computing, engineering design

\[ y = A(x) + \text{noise}. \]

\( A \): given by nature, or manually designed
Inverse problems

Canonical challenge in machine learning, scientific computing, engineering design

\[ y = \mathcal{A}(x) + \text{noise}. \]

\( \mathcal{A} \): given by nature, or manually designed

**Goal:** Given \( y \) and (possibly) \( \mathcal{A} \), recover an estimate of \( x \).
Inverse problems

Canonical challenge in machine learning, scientific computing, engineering design

Parameters → Forward model → Observations

Imaging / CV

- Imaging
- (De)compression
- Confocal microscopy
- MRI/CT
Inverse problems

Canonical challenge in machine learning, scientific computing, engineering design

Parameters \rightarrow Forward model \rightarrow Observations

**Imaging / CV**
- Imaging
- (De)compression
- Confocal microscopy
- MRI/CT

**Systems, design, prediction**
- System identification
- Inverse design
- Prognostics
- NDE
Estimation typically ill-posed; additional information / priors necessary
Classical approach

Estimation typically ill-posed; additional information / priors necessary

Estimation typically posed in terms of (constrained) optimization:

\[ \hat{x} = \min_x F(x|y, A) \]

\[ \text{s.t. } x \in S \]
Classical approach

Estimation typically ill-posed; additional information / priors necessary

Estimation typically posed in terms of (constrained) optimization:

\[ \hat{x} = \min_x F(x|y, A) \]

\[ \text{s.t. } x \in S \]

\( S \) denotes hypothesis class (prior) for “true” \( x \):
Classical approach

Estimation typically ill-posed; additional information / priors necessary

Estimation typically posed in terms of (constrained) optimization:

$$\hat{x} = \min_x F(x | y, A)$$

s.t. $x \in S$

$S$ denotes hypothesis class (prior) for “true” $x$:

- Bounded total variation
- Smoothness
- RKHS
- Sparsity in a basis/dictionary (synthesis, analysis, . . .)
- . . .
Given training data samples, learn a \textit{neural generative prior} as the hypothesis class
Neural Priors

Given training data samples, learn a *neural generative prior* as the hypothesis class

Example prior: Generative Adversarial Networks (GANs)

[Goodfellow et al, 2014]

\[ G(z) \] is a convolutional neural network (CNN) that models the distribution of \( S \)

\( z \): random variable
Promise of GANs

[Brock, Donahue, Simonyan, 2018]

BigGAN, \( \text{dim}(z) = 32, x = G(z), \text{dim}(x) = 256^2 \)
Promise of GANs

[Brock, Donahue, Simonyan, 2018]

BigGAN, $\dim(z) = 32$, $x = G(z)$, $\dim(x) = 256^2$
Using neural priors to solve PDEs

**Intuition:** Think of solving a PDE as a (non)linear mapping from one space (parameters/boundary conditions) to another space (domain)
Intuition: Think of solving a PDE as a (non)linear mapping from one space (parameters/boundary conditions) to another space (domain)

Why this is possible: Neural networks can approximate arbitrary high dimensional maps if architecture sufficiently wide/deep

[Cybenko ’89], [Funahashi ’89], [Hornik et al ’89], [Kurkova ’92]
Intuition: Think of solving a PDE as a (non)linear mapping from one space (parameters/boundary conditions) to another space (domain).

Why this is possible: Neural networks can approximate arbitrary high dimensional maps if architecture sufficiently wide/deep.

More recent developments: Refinements to reasonable width but (very) deep networks.
Intuition: Think of solving a PDE as a (non)linear mapping from one space (parameters/boundary conditions) to another space (domain)

Why this is possible: Neural networks can approximate arbitrary high dimensional maps if architecture sufficiently wide/deep

[Cybenko '89], [Funahashi '89], [Hornik et al '89], [Kurkova '92]

More recent developments: Refinements to reasonable width but (very) deep networks

[Lu et al, '17], [Hardt-Ma '17], [Lin-Jegelka '18]

Our approach: Train neural networks that can solve PDEs
Intuition: Think of solving a PDE as a (non)linear mapping from one space (parameters/boundary conditions) to another space (domain)

Why this is possible: Neural networks can approximate arbitrary high dimensional maps if architecture sufficiently wide/deep

[Cybenko '89], [Funahashi '89], [Hornik et al '89], [Kurkova '92]

More recent developments: Refinements to reasonable width but (very) deep networks

[Lu et al, '17], [Hardt-Ma '17], [Lin-Jegalka '18]

Our approach: Train neural networks that can solve PDEs

But not in the standard way..
Learning based solutions to PDEs

[Raissi, Perdikaris, Karniadakis 2018]
[Yang, Zhang, Karniadakis 2018]
[Lu, Meng, Mao, Karniadakis 2019]
[Zhou, Zabaras, Koutsourelakis, Perdikaris 2019]
and many others . . .
Hybrid neural prior that combines data and physics

\[ L_{\text{Inv}}(W) = \|y - A(x)\|^2, \quad x = G_W(z) \]
DiffNets: Formulation

Hybrid neural prior that combines data and physics

\[ L_{\text{Inv}}(W) = \|y - A(x)\|^2, \quad x = G_W(z) \]

\[ L_{\text{GAN}}(W, \Psi) = \mathbb{E}_{x' \sim P_{\text{data}}} [\phi(D_{\Psi}(x'))] + \mathbb{E}_{x \sim P_{x}} [\phi(-D_{\Psi}(x))] \]

Optimizing the min-max two-player game:

\[ \min_W \max_{\Psi} L_{\text{GAN}} + \mu L_{\text{Inv}} \]
Example: Elliptic PDEs

Assume a stochastic elliptic PDE of the form:

\[ A(u) = f + \xi, \]

Example:

\[ \nabla (K \circ \nabla (u)) = f + \xi, \quad \text{(Stochastic heat equation).} \]
Example: Elliptic PDEs

Assume a stochastic elliptic PDE of the form:

\[ \mathcal{A}(u) = f + \xi, \]

Example:

\[ \nabla(K \circ \nabla(u)) = f + \xi, \quad \text{(Stochastic heat equation).} \]

Discretize (using finite differences):

\[ \nabla^2 u \approx \frac{1}{4h^2} (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}) \]

This gives a linear system of equations (assuming period BC):

\[ (I - P_{\Omega})Au = (I - P_{\Omega})f + \xi, \]

\[ P_{\Omega}u = P_{\Omega}b. \]
DiffNet: Elliptic PDEs

Periodic boundary conditions

Upshot: comparable accuracy as numerical solvers; solution for new boundary conditions only requires a forward pass through the generator CNN.
We can explicitly write down the solution space as:

\[ u = \begin{bmatrix} A^{-1} P_{\Omega_c} f \\ P_{\Omega} b \end{bmatrix} + \begin{bmatrix} z \\ 0 \end{bmatrix} \]  

(1)

where \( z \) is normally distributed with covariance \( A^{-1} P_{\Omega_c} A (-1)^T \).
We can explicitly write down the solution space as:

\[ u = \begin{bmatrix} A^{-1} P_{\Omega^c} f \\ P_{\Omega} b \end{bmatrix} + \begin{bmatrix} z \\ 0 \end{bmatrix} \]  

(1)

where \( z \) is normally distributed with covariance \( A^{-1} P_{\Omega^c} A(-1)^T \).

Setup:

- We do not know the form of the PDE (i.e., \( A \) is not known), but have access to training data sampled from distribution of \( u \).
- We enforce boundary conditions as invariances
- Noise is gaussian
Generator: \( z \mapsto \Theta z + \lambda \); Discriminator: \( x \mapsto x^T P_{\Omega_c} \Psi P_{\Omega_c} x \)

Solve: \( \min_W \max_\Psi L_{\text{GAN}} + \mu \mathbb{E}_{u \sim G(W(\mathcal{N}))} \| P_\Omega (u - b) \|_2^2 \).
Generator: $z \mapsto \Theta z + \lambda$; Discriminator: $x \mapsto x^T P_{\Omega^c} \Psi P_{\Omega^c} x$

Solve: $\min_W \max_{\Psi} L_{GAN} + \mu \mathbb{E}_{u \sim G_W(N)} \| P_{\Omega} (u - b) \|^2_2$.

**Theorem:** At Nash equilibrium, we get:

\[
\Psi_{\Omega^c} = 0, \\
\Theta_{\Omega} = 0, \\
\Theta_{\Omega^c} \Theta_{\Omega^c}^T = A^{-1} P_{\Omega^c} A^{-1}.
\]

In other words: the generator

- provably learns unknown dynamics (inverse of $A$) up to rotation
- provably enforces known constraints (boundary conditions)
Example: (In)viscid Burgers Equation

\[ \frac{\partial u}{\partial t} + u \cdot \nabla u = (\nu + \xi) \nabla^2 u, \]

\[ u_{t=0} = 1 - \cos \frac{2\pi cx}{L} \]
Solving nonlinear PDEs

Example: (In)viscid Burgers Equation

\[
\frac{\partial u}{\partial t} + u \cdot \nabla u = (\nu + \xi) \nabla^2 u, \\
u_{t=0} = 1 - \cos \frac{2\pi cx}{L}
\]

Upshot: Far quicker solutions (about 1500X speedup) than standard numerical solvers; effective surrogate in lieu of solving stochastic PDEs.
Solving nonlinear PDEs

Example: (In)viscid Burgers Equation

\[
\frac{\partial u}{\partial t} + u \cdot \nabla u = (\nu + \xi) \nabla^2 u,
\]

\[
u_{t=0} = 1 - \cos \frac{2\pi cx}{L}
\]

**Upshot:** Far quicker solutions (about 1500X speedup) than standard numerical solvers; effective surrogate in lieu of solving stochastic PDEs
Physics-aware conditional generative models

Going beyond PDEs: instead of enforcing PDE constraints, one can use other prior information (statistics, geometry, etc)
Physics-aware conditional generative models

Going beyond PDEs: instead of enforcing PDE constraints, one can use other prior information (statistics, geometry, etc)

- Example invariance: mass fraction; $\alpha = E_r x(r)$
- Example invariance: 2-point correlation; $\beta(r) = E_{r_1, r_2} x(r_1) x(r_2)$
- Example invariance: connectivity or shape information
Physics-aware conditional generative models

Going beyond PDEs: instead of enforcing PDE constraints, one can use other prior information (statistics, geometry, etc)

- Example invariance: mass fraction; $\alpha = \mathbb{E}_r x(r)$
- Example invariance: 2-point correlation; $\beta(r) = \mathbb{E}_{r_1, r_2} x(r_1) x(r_2)$
- Example invariance: connectivity or shape information

Model: Invariance Network (InvNet). Same as above, but enforce above properties.

InvNets: Results

Invariance: P1 (volume fraction)

Desired: 0.42  
Actual: 0.45

Desired: 0.52  
Actual: 0.55

Desired: 0.61  
Actual: 0.64

Desired: 0.72  
Actual: 0.72

Invariance: 2 point correlation distance (in pixels)

Desired: 23 pix  
Actual: 25 pix

Desired: 33 pix  
Actual: 31 pix

Desired: 42 pix  
Actual: 42 pix

Desired: 50 pix  
Actual: 48 pix
InvNets: Results

Invariance: P1 (volume fraction)

<table>
<thead>
<tr>
<th>Desired</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.42</td>
<td>0.45</td>
</tr>
<tr>
<td>0.52</td>
<td>0.55</td>
</tr>
<tr>
<td>0.61</td>
<td>0.64</td>
</tr>
<tr>
<td>0.72</td>
<td>0.72</td>
</tr>
</tbody>
</table>

Invariance: 2 point correlation distance (in pixels)

<table>
<thead>
<tr>
<th>Desired</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>23 pix</td>
<td>25 pix</td>
</tr>
<tr>
<td>33 pix</td>
<td>31 pix</td>
</tr>
<tr>
<td>42 pix</td>
<td>42 pix</td>
</tr>
<tr>
<td>50 pix</td>
<td>48 pix</td>
</tr>
</tbody>
</table>

Upshot: Far quicker microstructure reconstruction (nearly 1500X speedup in amortized time)
Summary

This talk: Solving PDEs using neural networks.

- Conditional generative models
- Wide Applicability
- (Preliminary) Theoretical guarantees
This talk: Solving PDEs using neural networks.

- **Conditional generative models**
  Allow for user tuning of inputs/boundary conditions

- **Wide Applicability**
  Works for linear, non-linear PDEs

- **(Preliminary) Theoretical guarantees**
  Nash equilibrium provably learns the inverse

Open problems:

- Properly modeling grid mismatch (ideas from multi-scale modeling?)
- Faster training (ideas from PDE pre-conditioning?)
- Limited samples (ideas from transfer learning?)
- Guarantees (ideas from optimization and statistics?)
Thanks

Funded by the DARPA AIE (AIRA) Program; Dec 2018 - present.

- Baskar Ganapathysubramanian, Soumik Sarkar (ISU)
- Daniel Sparkman (AFRL)
- Payel Das, Youssef Mroueh (IBM)


