Transfer and Multi-Task Learning in Physics-Based Applications with Deep Neural Operators

Somdatta Goswami
Assistant Professor (Research)
Applied Mathematics, Brown University
Can Neural Networks Predict in Real-Time (Autonomy)?

Courtesy: MIT, Michael Triantafyllou
Can we predict the rupture of aneurysm?
Can we predict the rupture of aneurysm?

Courtesy: Boston Children's Hospital, J. Marsden
Data + Laws of Physics

**The 5D Law:** Dinky, Dirty, Dynamic, Deceptive Data

Three scenarios of Physics-Informed Learning Machines

- **Lots of Physics:** Small Data, FEM
- **Some Physics:** Some Data, PINNs
- **No Physics:** Big Data, Neural Operators

Scientific ML
Motivation

\[ k \left( \frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} \right) = f(x, y) \]

\[ \Omega = [0,1] \times [0,1] \]

**Domain**

To Generalize: Need an approach to learn to predict the solution for unseen \( f(x, y) \).

**Aim**

\[ f_i(x, y)_{i=1}^{n} \rightarrow \Phi \rightarrow u_i(x, y)_{i=1}^{n} \]

\[ \mathcal{F} \rightarrow \Phi \rightarrow \mathcal{U} \]
Operator learning

Input-output map \( \Phi: \mathcal{F} \to \mathcal{U} \) \( \mathcal{F}, \mathcal{U} \) are infinite dimensional function space

Data \( \{\mathcal{F}_i, \mathcal{U}_i\}_{i=1}^n \) \( \mathcal{U}_i = \Phi(\mathcal{F}_i) \), \( \mathcal{F}_i \sim \mu \ i.i.d \)

Operator learning \( \Psi: \mathcal{F} \times \Theta \to \mathcal{U} \) such that \( \Psi(. , \theta^*) \approx \Phi \)

Training \( \theta^* = \arg\min_{\theta} l(\{\mathcal{F}_i, \Psi(\mathcal{U}_i, \theta)\}) \)

Universal Approximation Theorem for Operators
\( \mathcal{G}: f \to \mathcal{G}(f), \mathcal{G}(f): y \in \mathbb{R}^d \to \mathbb{R} \)
Deep Operator Network (DeepONet)

- Generalized Universal Approximation Theorem for Operator [Chen ’95, Lu et al. ’19]
- **Branch net:** Input \( \{ f(x_i) \}_{i=1}^m \), output: \( [b_1, b_2, \ldots, b_p]^T \in \mathbb{R}^p \)
- **Trunk net:** Input \( y \), output: \( [t_1, t_2, \ldots, t_p]^T \in \mathbb{R}^p \)
- Input \( f \) is evaluated at locations \( \{ y_i \}_{i=1}^m \)

\[
\hat{u} = G_\theta(f)(y) = \sum_{i=1}^p b_i(f(x_1), f(x_2), \ldots, f(x_m)) \cdot t_{ri}(y)
\]

Minimize loss

\[
\theta^* = \arg\min_{\theta} \mathcal{L}_r(\theta) + \mathcal{L}_t(\theta)
\]
Constructing the DeepONet model

Testing Boundary conditions $f(x, y)$

Query Points during testing

Sensors $(m) = 10$

$$k \left( \frac{\partial^2 u(x, y)}{\partial x^2} + \frac{\partial^2 u(x, y)}{\partial y^2} \right) = f(x, y)$$

$\mathcal{G}_\theta: f(x, y) \rightarrow u(x, y)$

DeepONet is data hungry
Transfer learning

• Transfer learning (TL) allows us to learn from a source data distribution a well performing model on a different but related target data distribution

• TL addresses the expense of data acquisition and labelling, potential computational power limitations and dataset distribution mismatches

• Problem setup:

\[ \mathcal{D}^S = \{(x_i^S, y_i^S)\} \text{ labeled data sampled from } P \]

\[ \mathcal{D}^T = \{(x_i^T)\} \text{ (un)labeled data samples from } Q \]

\( P \neq Q \)

\( S \): source domain, \( T \): target domain
Covariate vs conditional shift

\[ u_t = \left( \frac{u^2}{2} \right)_x + v u_{xx}. \]

\( \mathcal{D}^S = \{(x^S_i, y^S_i)\} \quad \text{sufficient labeled data} \)
\( \mathcal{D}^T = \{(x^T_i)\} \quad \text{unlabeled data} \)

Computer vision problems:

- \( \mathcal{S} \): source domain
- \( \mathcal{T} \): target domain

Nonlinear PDE problems:

- \( \mathcal{S} \): source domain
- \( \mathcal{T} \): target domain

\( \mathcal{D}^S = \{(x^S_i, y^S_i)\} \quad \text{sufficient labeled data} \)
\( \mathcal{D}^T = \{(x^T_i, y^T_i)\} \quad \text{few labeled data} \)

- \( P(x_s) \neq P(x_t) \)
- \( P(y_s | x_s) = P(y_t | x_t) \)

“covariate shift”

- \( P(x_s) = P(x_t) \)
- \( P(y_s | x_s) \neq P(y_t | x_t) \)

“conditional shift”

\( \mathcal{S} \): source domain, \( \mathcal{T} \): target domain
Learning the solution of PDE on multiple domains

\[ \mathcal{X} = \mathcal{X}(\Omega; \mathbb{R}^{d_x}), \mathcal{Y} = \mathcal{Y}(\Omega; \mathbb{R}^{d_y}) \]

Nonlinear operator \( \mathcal{G} : \mathcal{X} \rightarrow \mathcal{Y} \)

Neural operator \( \mathcal{G}_\theta : \mathcal{X} \rightarrow \mathcal{Y}, \ \theta \in \Theta \)

Training data \( \{x_i, y_i\}_{i=1}^N \)

### Conditional shift

- \( P(x_s) = P(x_t) \)
- \( P(y_s|x_s) \neq P(y_t|x_t) \)

*\( x_s \): Model inputs; \( y_s \): Model outputs

\( \mathcal{S} \): source domain

\( \mathcal{D}^S = \{(x_i^S, y_i^S)\}_{i=1}^{N_s} \)

\( N_s \gg N_t \)

\( \mathcal{T} \): target domain

\( \mathcal{D}^T = \{(x_i^T, y_i^T)\}_{i=1}^{N_t} \)

Few labeled data

- Learning surrogate models in \textit{isolation} is expensive
- Training a surrogate with very few data can lead to overfitting
- Networks used for similar tasks (datasets) should be similar
- Leverage learned information between source and target models
  - Remove the need for big data for every new problem
  - Accelerate learning by fast fine-tuning of target network
Transfer learning approach

Nonlinear operator $\mathcal{G} : X \rightarrow Y$
Neural operator $\mathcal{G}_\theta : X \rightarrow Y$, $\theta \in \Theta$
Training data $\{x_i, y_i\}_{i=1}^N$

$S$: source domain

$sufficient labeled data$
$\mathcal{D}^S = \{(x_i^S, y_i^S)\}_{i=1}^{N_S}$

$\mathcal{T}$: target domain

$few labeled data$
$\mathcal{D}^T = \{(x_i^T, y_i^T)\}_{i=1}^{N_T}$

$\mathcal{N}_S \gg \mathcal{N}_T$

Branch net ($!$)
Trunk net ($"$)

Minimize Loss:
$\mathcal{L}(\theta^S) = \mathcal{L}_r(\theta^S)$

Output of the convolution layers for $X^S$

$G_{\theta^S}(X^S)(\zeta^S)$

$\theta^{S*}$

$G_{\theta^S}(X^T)(\zeta^T)$

$\mathcal{L}_{\text{CED}}(\theta^T)$
$\mathcal{L}_r(\theta^T)$

Discrepancy Loss

Minimize Loss

Match individual samples

Preserve global properties of target distribution

$\mathcal{S}$: source domain
$sufficient labeled data$
$\mathcal{D}^S = \{(x_i^S, y_i^S)\}_{i=1}^{N_S}$

CEOD: Conditional embedding operator discrepancy

Given two datasets: \( \mathcal{D}_p = \{(x_1, y_1), \ldots, (x_{N_1}, y_{N_1})\} \), \( \mathcal{D}_q = \{(x_1, y_1), \ldots, (x_{N_2}, y_{N_2})\} \)

\[
D_{\text{CEOD}}(\mathcal{D}_p, \mathcal{D}_q) = \left\| \hat{C}_{Y|X_p} - \hat{C}_{Y|X_q} \right\|_{HS}^2
= \left\| \Phi(Y_p) \left( K_{X_p X_p} + \lambda N_1 I \right)^{-1} Y^T(X_p) \\
- \Phi(Y_q) \left( K_{X_q X_q} + \lambda N_2 I \right)^{-1} Y^T(X_q) \right\|_{HS}^2
= \text{Tr} \left\{ \left( K_{X_p X_p} + \lambda N_1 I \right)^{-1} K_{Y_p Y_p} \left( K_{X_p X_p} + \lambda N_1 I \right)^{-1} K_{X_p X_p} \right\}
+ \text{Tr} \left\{ \left( K_{X_q X_q} + \lambda N_2 I \right)^{-1} K_{Y_q Y_q} \left( K_{X_q X_q} + \lambda N_2 I \right)^{-1} K_{X_q X_q} \right\}
- 2 \text{Tr} \left\{ \left( K_{X_p X_p} + \lambda N_1 I \right)^{-1} K_{Y_p Y_q} \left( K_{X_q X_q} + \lambda N_2 I \right)^{-1} K_{X_q X_p} \right\}
\]

- Inspired by: \( D_{\text{MMD}}(\mathcal{D}_p, \mathcal{D}_q) = \left\| \hat{\mu}_{X_p} - \hat{\mu}_{X_q} \right\|_{HS}^2 \)
- CEOD measures the conditional distribution discrepancy in a reproducing kernel Hilbert space (RKHS)
- Constructs a Hilbert–Schmidt norm of the empirical conditional embedding operators of two distributions
- Based on the theory of kernel embeddings of conditional distributions

\( \mathcal{C}_{Y|X} \): operator of inputs to outputs on a RKHS
\( \Phi, \Psi \): embed from original space to RKHS
\( K_{XX'} \): Gram matrix calculated with a Gaussian kernel \( k \)
\( \lambda \): regularization term to avoid overfitting

Liu, X. et al. (2021). Knowledge-Based Systems
CEOD: Conditional embedding operator discrepancy

RVS: $X, Y$ Original space $\Omega_X, \Omega_Y$

<table>
<thead>
<tr>
<th>$x \in \Omega_X$</th>
<th>$X \sim P_X$</th>
<th>$Y \sim P_Y$</th>
<th>$(X, Y) \sim P_{XY}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>conditioning on $X = x$</td>
<td>embed $\varphi(X) : \Omega_X \rightarrow \mathcal{H}$</td>
<td>embed $\psi(Y) : \Omega_Y \rightarrow \mathcal{F}$</td>
<td>condition on $X = x$</td>
</tr>
<tr>
<td>$(Y</td>
<td>X = x) \sim P_{Y</td>
<td>X=x}$</td>
<td>$\varphi(Y), \varphi(X)$</td>
</tr>
<tr>
<td>* $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Given a dataset: $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$:

$$\hat{C}_{Y|X} = \hat{C}_{YY} \hat{C}_{XX}^{-1} = \Phi (K + \lambda NI)^{-1} Y^T$$

where $\Phi := (\psi(y_1), ..., \psi(y_N))$

$Y := (\varphi(x_1), ..., \varphi(x_N))$

$K = Y^T Y$ Gram matrix

$\lambda$: regularization parameter

Given two datasets $\mathcal{D}_p, \mathcal{D}_q$:

$$D_{CEOD}(\mathcal{D}_p, \mathcal{D}_q) = \left\| \hat{C}_{Y_p|x_p} - \hat{C}_{Y_q|x_q} \right\|_{HS}^2$$
Transfer learning DeepONet loss function

Hybrid loss function

\[ \mathcal{L}(\theta^T) = \lambda_1 \mathcal{L}_r(\theta^T) + \lambda_2 \mathcal{L}_{CEOD}(\theta^T) \]

\[ = \lambda_1 \frac{\|f_T(x^{tL}) - y^{tL}\|_2}{\|y^{tL}\|_2} + \lambda_2 \left\| \hat{C}_{Y,tL|x_{tL}} - \hat{C}_{Y,tU|x_{tU}} \right\|_{HS}^2 \]

\[ \lambda_1, \lambda_2: \text{Trainable coefficients updated through backpropagation}^{1} \]

Regression loss

- \( \mathcal{D}_t = \{(x_i^{tL}, y_i^{tL})\}_{i=1}^{N_t} \)

CEOD loss

- \( \mathcal{D}_t^L = \{(x_{b1i}^{tL}, y_i^{tL})\}_{i=1}^{N_t} \)
- \( \mathcal{D}_t^U = \{(x_{b1i}^{tU}, f_T(y_i^{tU}))\}_{i=1}^{N_u} \)

\( x_{b1i} \): output of the 1st FNN layer of branch net

1 Kontolati, K., Goswami, S., et al. (2023). *Journal of Computational Physics*
## Transfer learning applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Input Function</th>
<th>Model Output</th>
<th>Domain Visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Darcy Flow</td>
<td>Random input conductivity field ( K(x) \sim GP(0, \mathcal{K}(x, x')) )</td>
<td>( \nabla \cdot (K(x) \nabla h(x)) = 1 )</td>
<td><img src="T11" alt="Source" /> <img src="T12" alt="Target" /> <img src="T13" alt="Target" /></td>
</tr>
<tr>
<td></td>
<td>( \mathcal{K}(x, x') = \exp \left[ -\frac{(x-x')^2}{2l^2} \right] )</td>
<td>( h(x) = 0 \ \forall \ x \in \partial \Omega )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( l = 0.25, x, x' \in [0,1]^2 )</td>
<td>( G_\theta: K(x) \rightarrow h(x) )</td>
<td></td>
</tr>
<tr>
<td>Elasticity Model</td>
<td>Random boundary conditions ( f(x) \sim GP(0, \mathcal{K}(x, x')) )</td>
<td>( \nabla \sigma + f(x) = 0 )</td>
<td><img src="U1" alt="Source" /> <img src="U2" alt="Target" /> <img src="U3" alt="Target" /></td>
</tr>
<tr>
<td></td>
<td>( \mathcal{K}(x, x') = \exp \left[ -\frac{(x-x')^2}{2l^2} \right] )</td>
<td>( (u, v) = 0 \ \forall \ x = 0 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( l = 0.12, x, x' \in [0,1] )</td>
<td>( G_\theta: f(x) \rightarrow (u, v) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( u: X\text{-Displacement} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( v: Y\text{-Displacement} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Material properties</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \varepsilon_s = 300 \cdot 10^5 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \nu_s = 0.3 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \varepsilon_s = 410 \cdot 10^5, \nu_s = 0.35 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \varepsilon_s = 410 \cdot 10^5, \nu_s = 0.45 )</td>
<td></td>
</tr>
<tr>
<td>Brusselator Diffusion-Reaction System</td>
<td>Random initial condition ( h_2(x) \sim GP(h_2(x)</td>
<td>\mu(x), \mathcal{K}(x, x')) )</td>
<td>( \partial u / \partial t = D_0 \nabla^2 u + a - (1 - b)u + vu^2 )</td>
</tr>
<tr>
<td></td>
<td>( v(x, y, t = 0) = h_2(x, y) \geq 0 )</td>
<td>( \partial v / \partial t = D_1 \nabla^2 v + bu - vu^2 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \mathcal{K}(x, x') = \sigma^2 \exp \left[ -\frac{(x-x')^2}{2l^2} \right] )</td>
<td>( x \in [0,1]^2, t \in [0,1] )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( l_x = 0.12, l_y = 0.4, \sigma^2 = 0.15 )</td>
<td>( G_\theta: h_2(x, y) \rightarrow v(x, y, t) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Model parameter</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( b_5 = 2.2 )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( b_{T_1} = 1.7, )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( b_{T_2} = 3.0 )</td>
<td></td>
</tr>
</tbody>
</table>

- **geometric domain shift**
- **geometric domain + model parameter shift**
- **model dynamics shift**
Results: Darcy Flow (TL3)

Objective:
\[ \mathcal{G}: K(x) \rightarrow h(x) \]

Random input conductivity field
\[ K(x) \sim \mathcal{GP}(0, \mathcal{K}(x, x')) \]
\[ \mathcal{K}(x, x') = \exp \left[ -\frac{(x-x')^2}{2l^2} \right] \]
\[ l = 0.25, x, x' \in [0,1]^2 \]

Hydraulic head
\[ \nabla(K(x)\nabla h(x)) = 1 \]
\[ h(x) = 0 \ \forall \ x \in \partial \Omega \]

Transfer learning scenario:
Results: Darcy Flow (TL4)

Objective:

\[ G: K(x) \rightarrow h(x) \]

Random input conductivity field

\[ K(x) \sim \mathcal{GP}(0, \mathcal{K}(x, x')) \]
\[ \mathcal{K}(x, x') = \exp \left[-\frac{(x-x')^2}{2l^2}\right] \]
\[ l = 0.25, x, x' \in [0,1]^2 \]

Hydraulic head

\[ \nabla \cdot (K(x) \nabla h(x)) = 1 \]
\[ h(x) = 0 \quad \forall \quad x \in \partial \Omega \]

Transfer learning scenario:
Representative result: Darcy Flow

Representative results of TL-DeepONet for Darcy’s problem

TL3
- Conductivity
- Pressure: Truth
- Error

TL4
- Conductivity
- Pressure: Truth
- Error
Representative result: Darcy Flow

Table: GPU time (s) for all Darcy flow transfer learning scenarios

<table>
<thead>
<tr>
<th>Training DeepONet</th>
<th>N_t</th>
<th>TL1</th>
<th>TL2</th>
<th>TL3</th>
<th>TL4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(source)</td>
<td>2,000</td>
<td>15,260</td>
<td>15,260</td>
<td>15,260</td>
<td>2,261</td>
</tr>
<tr>
<td>(target)</td>
<td>2,000</td>
<td>12,880</td>
<td>18,200</td>
<td>18,080</td>
<td>3,978</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>129</td>
<td>116</td>
<td>112</td>
<td>139</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>416</td>
<td>399</td>
<td>302</td>
<td>289</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>439</td>
<td>437</td>
<td>351</td>
<td>300</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>459</td>
<td>439</td>
<td>378</td>
<td>302</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>462</td>
<td>480</td>
<td>406</td>
<td>304</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>531</td>
<td>528</td>
<td>586</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>2,000</td>
<td>595</td>
<td>601</td>
<td>653</td>
<td>350</td>
</tr>
</tbody>
</table>

* Simulations performed on single NVIDIA RTX A6000 GPU
Results: Linear elasticity (TL5)

Objective: \( G: f(x) \rightarrow [u(x), v(x)] \)

Random RHS

\( f(x) \sim GP(0, \mathcal{K}(x, x')) \)

\( \mathcal{K}(x, x') = \exp \left[ -\frac{(x-x')^2}{2l^2} \right] \)

\( l = 0.12, x, x' \in [0,1]^2 \)

Displacement

\[ \nabla \sigma + f(x) = 0 \]

\[ u(x) = v(x) = 0 \quad \forall x = 0 \]

Relative \( L_2 \) error

\begin{align*}
\text{X-displacement} & \quad \text{Y-displacement} \\
N & \quad N \\
[0, 140] & \quad [0, 140] \\
\end{align*}

Transfer learning scenario:

\begin{align*}
E_s &= 300e5 \\
\nu_s &= 0.3 \\
E_T &= 410e3 \\
\nu_T &= 0.35
\end{align*}
Results: Limitations of TL-DeepONet

- Different internal, external boundaries and material properties;
- Predicted displacements field deviates significantly with the response predicted by training DeepONet from scratch

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (E_s = 300 \cdot 10^5, \nu_s = 0.3) )</td>
<td>( (E_s = 410 \cdot 10^5, \nu_s = 0.35) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( N_t )</th>
<th>( u(x) )</th>
<th>( L_2 (%) )</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training DeepONet (source)</td>
<td>1,900</td>
<td>2.30 ± 0.49</td>
<td>3.22 ± 0.48</td>
</tr>
<tr>
<td>Training DeepONet (target)</td>
<td>1,900</td>
<td>2.72 ± 0.26</td>
<td>1.92 ± 0.41</td>
</tr>
<tr>
<td>Training TL-DeepONet</td>
<td>5</td>
<td>60.28 ± 1.95</td>
<td>57.48 ± 2.54</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>29.14 ± 0.31</td>
<td>21.42 ± 3.20</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>16.4 ± 2.01</td>
<td>18.72 ± 1.18</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>11.37 ± 0.34</td>
<td>14.15 ± 0.96</td>
</tr>
<tr>
<td></td>
<td>150</td>
<td>9.66 ± 0.26</td>
<td>11.95 ± 0.61</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>5.52 ± 0.20</td>
<td>10.94 ± 0.20</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>4.08 ± 0.06</td>
<td>9.18 ± 0.14</td>
</tr>
<tr>
<td></td>
<td>1,900</td>
<td>3.82 ± 0.20</td>
<td>7.89 ± 0.21</td>
</tr>
</tbody>
</table>

* Simulations performed on single NVIDIA RTX A6000 GPU
Key takeaways

• Learn operators on multiple PDE domains via transfer learning and treat trained models as reusable building blocks

• TL-DeepONet:
  • Performs well under small-data regimes
  • TL-DeepONet accelerates learning via fine-tuning pre-trained models
  • Enhances generalizability in neural operators

• Code availability: https://github.com/katiana22/TL-DeepONet.git
References


* denotes equal contribution
The team

Dr. George Karniadakis
Professor
Brown University

Dr. Somdatta Goswami
Assistant Professor (Research)
Brown University

Dr. Michael Shields
Associate Professor
Johns Hopkins University

Dr. Katiana Kontolati
Johns Hopkins University

Computing support:
CCV @Brown
RockFish @JHU
Positions available

Ph.D. -2 *(starting Spring)*
Postdocs – 1 *(immediately)*

in the Department of Civil and Systems Engineering at Johns Hopkins University

Topics: SciML for materials and mechanics

Email: somdatta89@gmail.com