A Deep Learning Framework for p-Adaptation

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

Joint work with Qian Wang (EPFL)

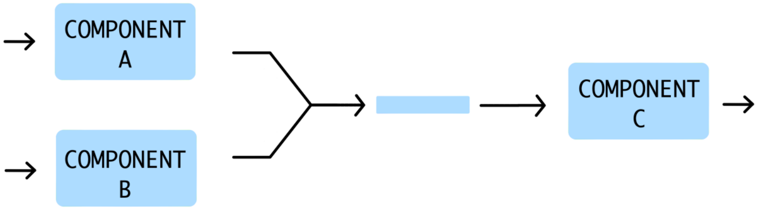
SIAM CSE
March 5, 2021

USC Viterbi
School of Engineering

Motivation: data-driven assist

Theoretically sound techniques available for

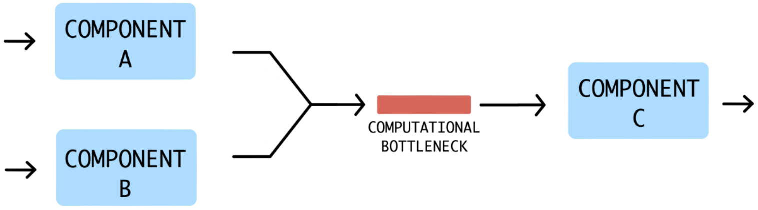
- Solving differential equations
- Optimization and control
- Uncertainty quantification
- Reduced order modelling
- Inverse problems
- ...



Motivation: data-driven assist

Theoretically sound techniques available for

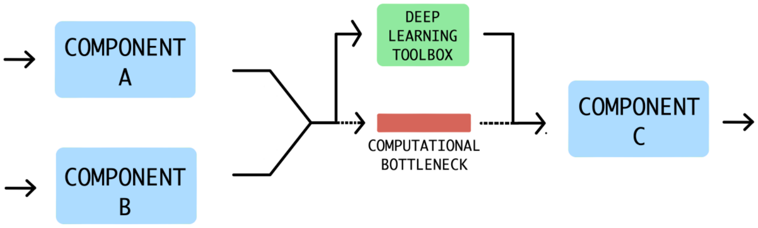
- Solving differential equations
- Optimization and control
- Uncertainty quantification
- Reduced order modelling
- Inverse problems
- ...



Motivation: data-driven assist

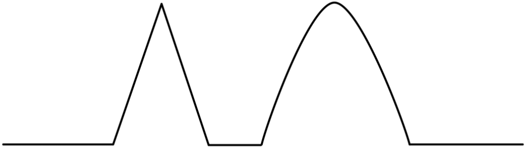
Theoretically sound techniques available for

- Solving differential equations
- Optimization and control
- Uncertainty quantification
- Reduced order modelling
- Inverse problems
- ...



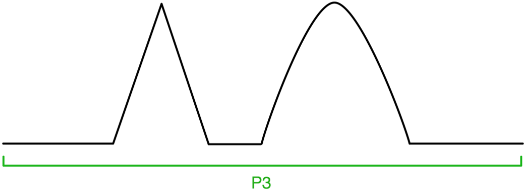
Motivation: p-adaptation

Approximate



Motivation: p-adaptation

Approximate

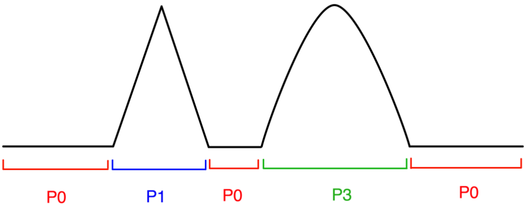


Using finite-element based methods, in each element E_i:

$$u_i^h(x) = \sum_{j=1}^{N_p} \hat{u}_j^i \phi_j(x), \qquad \text{update/solve coef. } \hat{u}_j^i$$

Motivation: p-adaptation

Approximate



Using finite-element based methods, in each element E_i:

$$u_i^h(x) = \sum_{j=1}^{N_p} \hat{u}_j^i \phi_j(x), \qquad \text{update/solve coef. } \hat{u}_j^i$$

Adapt p**:** $u_i^h(x) = \sum_{j=1}^{N_{p_i}} \hat{u}_j^i \phi_l(x), \;\; 0 \leq p_i \leq p_{max} \quad \Longrightarrow$ lower cost

Issue: Existing methods use problem-dependent parameters.

Hyperbolic conservation laws

$$\frac{\partial \mathbf{u}(\mathbf{x}, t)}{\partial t} + \nabla \cdot \mathbf{f}(\mathbf{u}(\mathbf{x}, t)) = 0$$
$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0(\mathbf{x})$$

- Shallow water equations
- Euler equations
- MHD equations

Hyperbolic conservation laws

$$\frac{\partial \mathbf{u}(\mathbf{x},t)}{\partial t} + \nabla \cdot \mathbf{f}(\mathbf{u}(\mathbf{x},t)) = 0$$
$$\mathbf{u}(\mathbf{x},0) = \mathbf{u}_0(\mathbf{x})$$

- Shallow water equations
- Euler equations
- MHD equations

Discontinuous Galerkin schemes + deep learning:

- Troubled-cell detector (classification): [Ray et al., 2018; Ray et al., 2019; Beck et al., 2020; Feng et al., 2020]
- Artificial viscosity (regression): [Discacciati et al., 2020]

Advantage:

- Universal strategy – free from problem-dependent parameters.
- More bang for the buck!

Hyperbolic conservation laws

$$\frac{\partial \mathbf{u}(\mathbf{x}, t)}{\partial t} + \nabla \cdot \mathbf{f}(\mathbf{u}(\mathbf{x}, t)) = 0$$
$$\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0(\mathbf{x})$$

- Shallow water equations
- Euler equations
- MHD equations

Discontinuous Galerkin schemes + deep learning:

- Troubled-cell detector (classification): [Ray et al., 2018; Ray et al., 2019; Beck et al., 2020; Feng et al., 2020]
- Artificial viscosity (regression): [Discacciati et al., 2020]

Advantage:

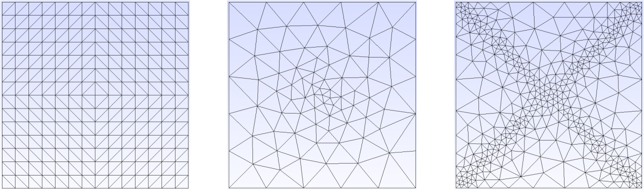
- Universal strategy – free from problem-dependent parameters.
- More bang for the buck!

AIM: Train a network to predict the local order

NN-based p-adapter

Datasets:

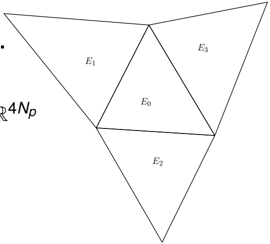
- Choose a function $u(\mathbf{x})$ with known regularity.
- Interpolate on a mesh:



- Extract modal coefficients from local stencils.
- Create the input vector and label:

$\mathbf{X} = [u_1^0, ..., u_{N_p}^0, u_1^1, ..., u_{N_p}^1, u_1^2, ..., u_{N_p}^2, u_1^3, ..., u_{N_p}^3]^\top \in \mathbb{R}^{4N_p}$

$\mathbf{Y} \in \{0, ..., p\}$ (Classification)



NN based p-adapter ($p_{\max} = 3$)

Training set:

Function	**Target p**	**Samples**
Constants	0	135,000
P1	1	140,400
P2	2	140,400
P3	3	56,160
$a_1 + a_2 \sin(a_3 \pi x) + a_4 \cos(a_5 \pi y)$	3	42,120
$a_1 + a_2 \sin(a_3 \pi (a_4 x + a_5 y))$	3	42,120
		556,200

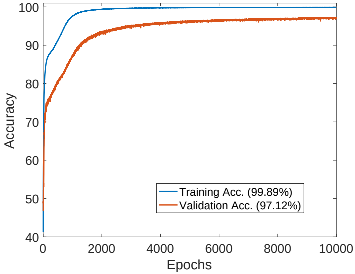
Validation set:

Function	**Target p**	**Samples**
Constants	0	21,620
P1	1	23,374
P2	2	23,374
P4	3	6,032
P5	3	6,032
P6	3	6,032
$a_1 + \sin(a_2 \pi x) \cos(a_3 \pi y) + \sin(a_4 \pi x) \cos(a_5 \pi y)$	3	6,032
		92,496

NN based p-adapter ($p_{\max} = 3$)

- Ensemble training to find suitable hyper-parameters of feedforward network (144 nets.)
- Best network:

Depth	6
Width	20
Activation	Leaky ReLU ($\alpha = 1e-3$)
Regularization	L2 ($\lambda = 1e-7$)
Learning rate	$1e-3$



Training Acc. = 99.89%

Validation Acc. = 97.12%

Numerical results

- Demonstration with $p_{\max} = 3 \implies \mathbf{X} \in \mathbb{R}^{40}$
- Modal DG scheme
- Compressible Euler equations:

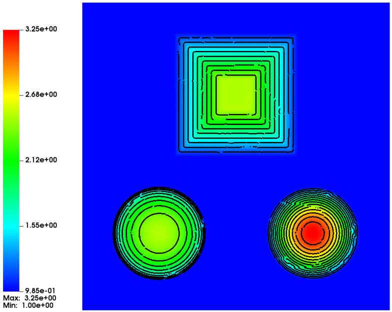
$$\mathbf{u} = \begin{bmatrix} \rho \\ \rho v_1 \\ \rho v_2 \\ E \end{bmatrix}, \quad \mathbf{f}_1 = \begin{bmatrix} \rho v_1 \\ \Pi + \rho v_1^2 \\ \rho v_1 v_2 \\ (E + \Pi) v_1 \end{bmatrix}, \quad \mathbf{f}_2 = \begin{bmatrix} \rho v_2 \\ \rho v_1 v_2 \\ \Pi + \rho v_2^2 \\ (E + \Pi) v_2 \end{bmatrix}$$

$$E = \rho \left(\frac{|\mathbf{v}|^2}{2} + e \right), \qquad e = \frac{\Pi}{(\gamma - 1)\rho}, \qquad \gamma = 1.4$$

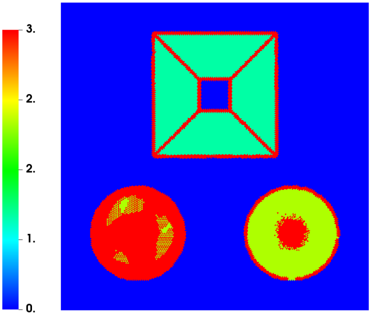
$$\begin{aligned} \rho &\longrightarrow \text{fluid density} \\ \mathbf{v} = (v_1, v_2) &\longrightarrow \text{velocity} \\ \Pi &\longrightarrow \text{pressure} \\ E &\longrightarrow \text{total energy} \\ e &\longrightarrow \text{internal energy} \end{aligned}$$

Advecting shapes

$\mathbf{v} = (1, 0)$, $\quad \Pi = 1$, $\quad K = 57,492$, $\quad$ adaptation based on ρ



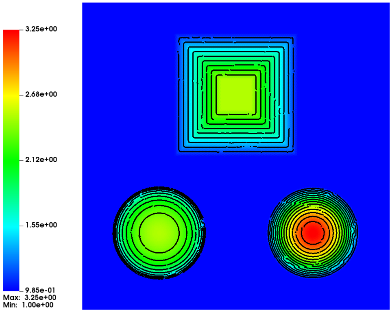
Density



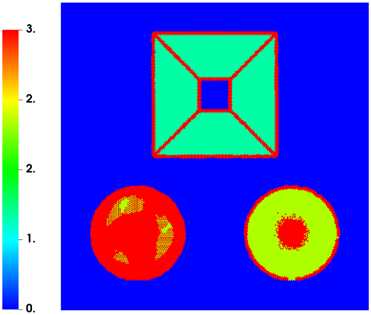
Order

Advecting shapes

$\mathbf{v} = (1,0), \quad \Pi = 1, \quad K = 57,492,$ adaptation based on ρ



Density



Order

Advecting shapes

CPU times (minutes):

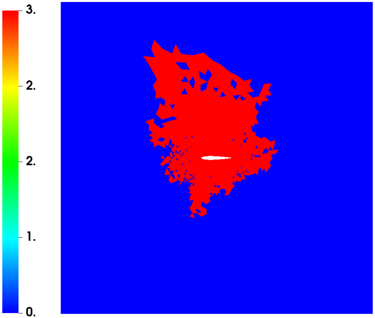
Scheme	Adaptation	Flux+Source	Time-stepping	Total
Full P3	0.0	8,377.1	73.3	8,450.4
NN Adapt.	79.6	2,882.7	23.3	2,985.6

Speed up $\approx$ 2.83!

Scheme	L^1 Error	L^∞ Error
Full P3	$2.29e-4$	$2.97e-2$
NN Adapt.	$2.44e-4$	$3.29e-2$

Subsonic flow past NACA-0012 airfoil

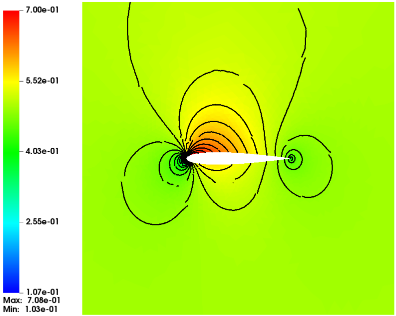
Mach Number = 0.5, a.o.a = 2°, $K = 10382$

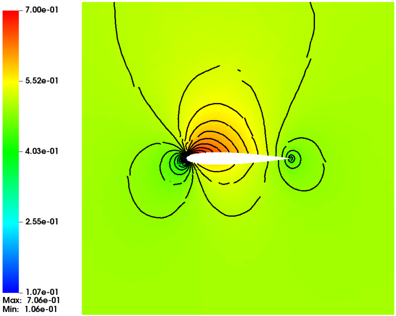


Order

Subsonic flow past NACA-0012 airfoil

Mach Number = 0.5, a.o.a = 2°, $K = 10382$





Full P3

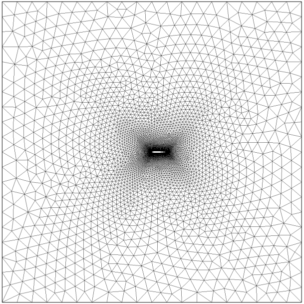
NN Adapt.

Subsonic flow past NACA-0012 airfoil

CPU times (minutes):

Scheme	Adaptation	Flux+Source	Time-stepping	Total
Full P3	0.0	14,261.6	126.5	14,388.1
NN Adapt.	148.0	10,493.0	83.3	10,724.3

Speed up $\approx$ 1.34



Mesh is adapted!

Double Mach reflection

$K = 80000$, using NN-based troubled-cell indicator

Density

Order

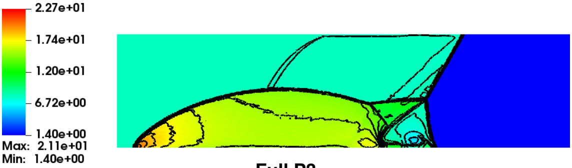
Double Mach reflection

$K = 80000$, using NN-based troubled-cell indicator

Density

Order

Double Mach reflection



Full P3

NN Adapt.

CPU times (minutes):

Scheme	Adaptation	Flux+Source	Limiting	Time-stepping	Total
Full P3	0.0	7,304.7	67.7	65.6	7,438.1
NN Adapt.	75.0	2,174.9	62.0	19.8	2,331.7

Speed up $\approx$ 3.19!

Concluding remarks

- Demonstrated that networks can be trained to predict local order.
- Network trained offline – one network for all problems.
- No additional tuning required.

Concluding remarks

- Demonstrated that networks can be trained to predict local order.
- Network trained offline – one network for all problems.
- No additional tuning required.

Next steps:

- Comparison with existing p-adapting algorithms [Burbeau et al., 2005; Naddei et al., 2019]
- hp-adaption – can we get a single network?
- Extension to hybrid grids and 3D.

Concluding remarks

- Demonstrated that networks can be trained to predict local order.
- Network trained offline – one network for all problems.
- No additional tuning required.

Next steps:

- Comparison with existing p-adapting algorithms [Burbeau et al., 2005; Naddei et al., 2019]
- hp-adaption – can we get a single network?
- Extension to hybrid grids and 3D.

Questions?

Confusion matrices

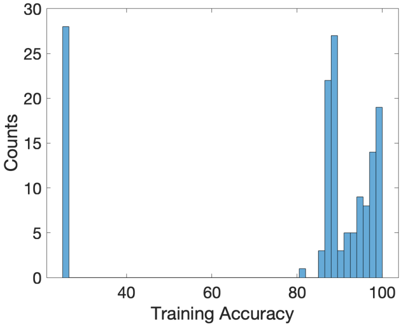
Training set:

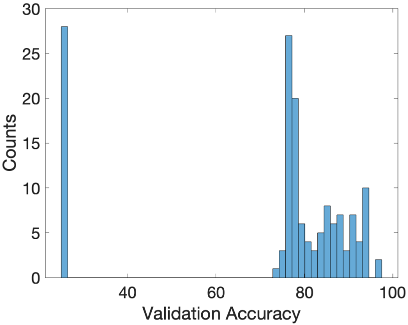
True / Pred.	0	1	2	3
0	135,000	0	0	0
1	0	140,397	3	0
2	0	124	140,143	133
3	17	24	280	140,079

Validation set:

True / Pred.	0	1	2	3
0	21,620	0	0	0
1	88	23,030	0	256
2	4	312	22,843	215
3	172	27	1589	22,340

Training histograms





Additional runtime details

- All training and simulations performed on the EPFL Fidis cluster, using Intel® Broadwell processors @2.6GHz
- Network training: 144 nets trained on 48 CPUs, approx. 12 hrs.
- Shapes: 84 CPUs, NACA: 28 CPUs, Double Mach: 28 CPUs
- Simulation times exclude MPI wait times (need dynamic partitioning)