Using deep learning to overcome algorithmic bottlenecks

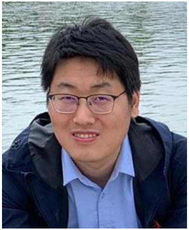
Deep Ray
Institute of Mathematics
Email: deep.ray@epfl.ch
Website: deepray@github.io

NumHyp19
Malaga, 22 June 2019

École
polytecnique
fédérale
de Lausanne

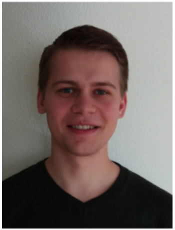
In collaboration with ...

Jan S. Hesthaven
Director MCSS
EPFL



Qian Wang
Postdoc
EPFL

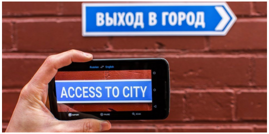
Niccolò Discacciati
Doctoral student
EPFL

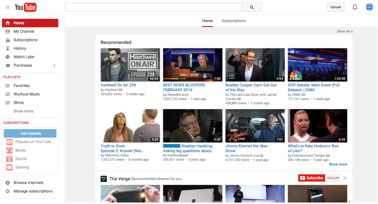


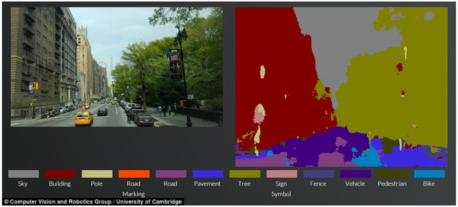
Lukas Schwander
Master student
ETH Zürich

Deep learning in action

Deep learning in action







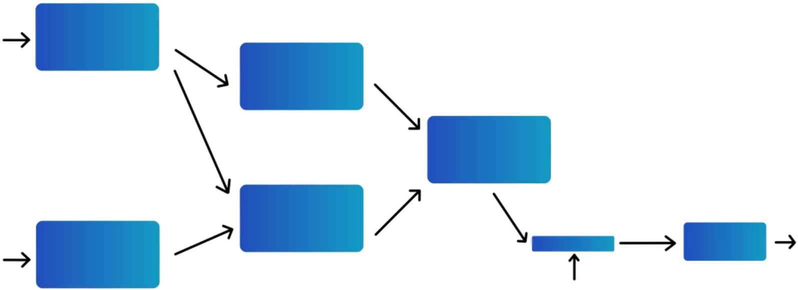
Blending deep learning and numerics

Blending deep learning and numerics

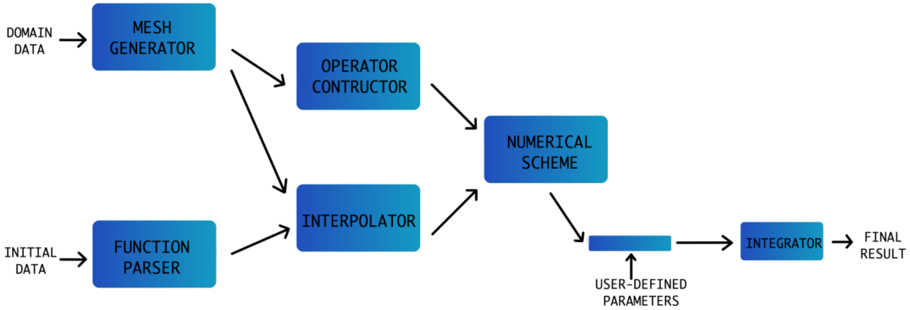
- Solving differential equations [Lagaris et al., 1998; Golak et al., 2007; Rudd et al., 2015; Tompson et al., 2017; Long et al. 2017; Raissi et al., 2018, Magiera et al., 2019]
- Subgrid scale modelling/LES/RANS [Tracey et al., 2013; Zhang et al., 2015; Ling et al., 2016; Kurian et al. 2018; Beck et al.; Duraisamy et al., 2019; Maulik et al., 2019]
- Uncertainty quantification [Tripathy et al., 2018, Kwon et al. 2018; Schwab et al, 2018; Mishra et al., 2019; Wang et al., 2019]
- Reduced order modelling [Kutz et al, 2016; Hartman et al., 2017; Carlberg et al. 2018; Willcox et al., 2019; Wang et al., 2019]
- Inverse problems [Schönlieb et al. 2017, Lunz et al., 2018; Chang et al., 2018; Raissi et al. 2019]
- Shape optimization [Timnaka et al., 2017; Baque at al., 2018; Duraisamy et al., 2019, Sasaki et al. 2019]
- ...

Motivation

Motivation

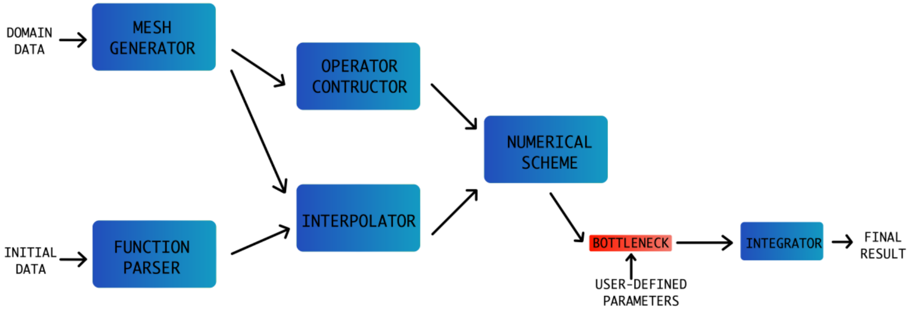


Motivation

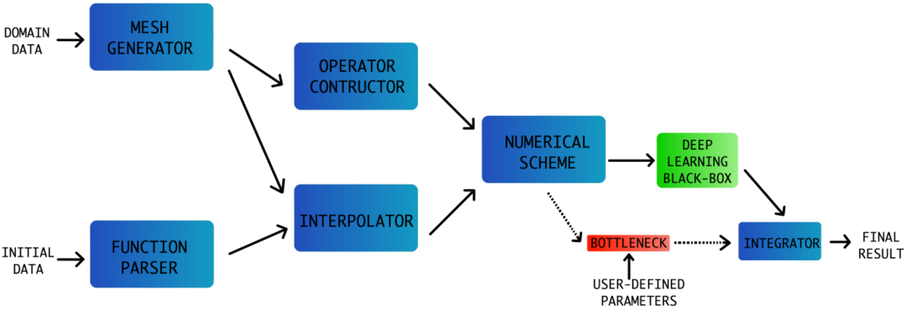


DOMAIN DATA
MESH GENERATOR
OPERATOR CONTRUCTOR
INITIAL DATA
FUNCTION PARSER
INTERPOLATOR
NUMERICAL SCHEME
USER-DEFINED PARAMETERS
INTEGRATOR
FINAL RESULT

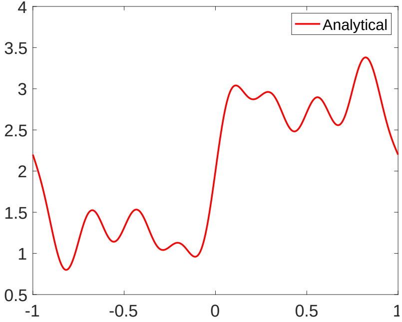
Motivation

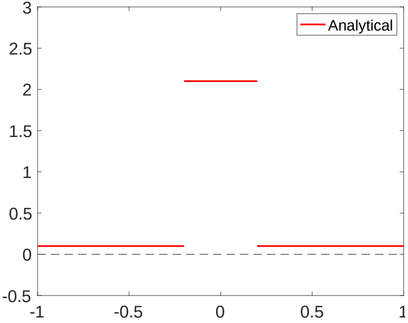

DOMAIN
DATA
MESH
GENERATOR
OPERATOR
CONTRUCTOR
NUMERICAL
SCHEME
INTERPOLATOR
INITIAL
DATA
FUNCTION
PARSER
BOTTLENECK
USER-DEFINED
PARAMETERS
INTEGRATOR
FINAL
RESULT

Motivation


DOMAIN DATA
MESH GENERATOR
OPERATOR CONTRUCTOR
INITIAL DATA
FUNCTION PARSER
INTERPOLATOR
NUMERICAL SCHEME
DEEP LEARNING BLACK-BOX
BOTTLENECK
USER-DEFINED PARAMETERS
INTEGRATOR
FINAL RESULT

Today's bottleneck – controlling spurious oscillations!

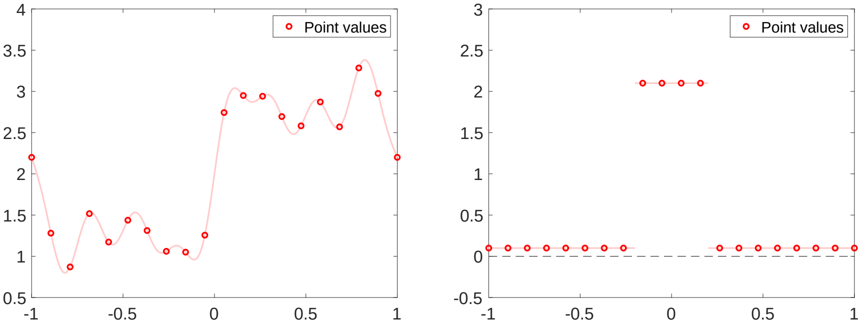




Today’s bottleneck – controlling spurious oscillations!

Approximating the function using a smooth basis

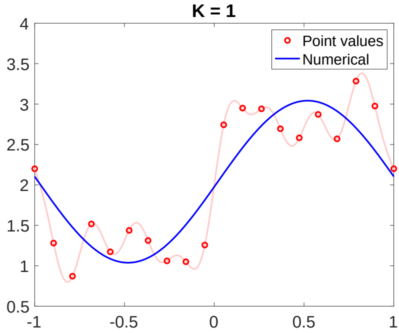
$$u(x) \approx \sum_{p=1}^{K} a_p \phi_p(x)$$

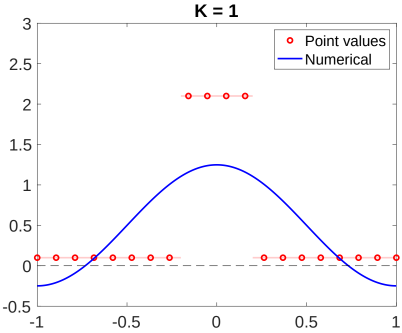


Today's bottleneck – controlling spurious oscillations!

Approximating the function using a smooth basis

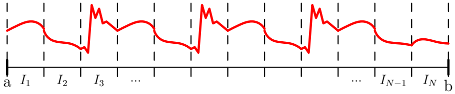
$$u(x) \approx \sum_{p=1}^{K} a_p \phi_p(x)$$





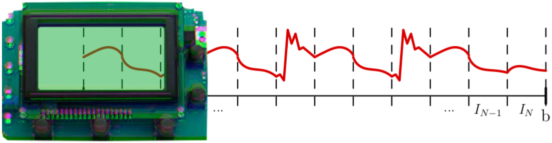
Today's bottleneck – controlling spurious oscillations!

Mechanism to detect discontinuity/smoothness



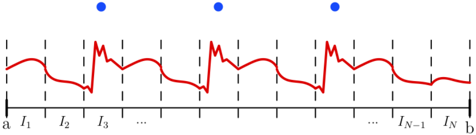
Today's bottleneck – controlling spurious oscillations!

Mechanism to detect discontinuity/smoothness



Today's bottleneck – controlling spurious oscillations!

Mechanism to detect discontinuity/smoothness

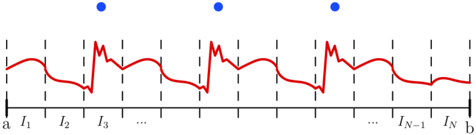


Based on the local analysis of data, we can

- Limit the local solution
- Adapt the local polynomial/basis order
- Add artificial viscosity

Today's bottleneck – controlling spurious oscillations!

Mechanism to detect discontinuity/smoothness



Based on the local analysis of data, we can

- Limit the local solution
- Adapt the local polynomial/basis order
- Add artificial viscosity

Problems with existing methods

- Specification of problem-dependent parameters
- Computationally-expensive
- Too conservative

Overcome bottlenecks using deep learning

We wish to approximate the function

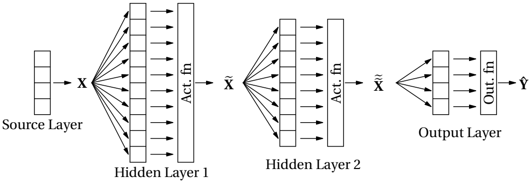
$$\mathbf{F} : \mathbf{X} \mapsto \mathbf{Y}, \quad \mathbf{X} \in \mathbb{R}^n \ \mathbf{Y} \in \mathbb{R}^m$$

Overcome bottlenecks using deep learning

We wish to approximate the function

$$\mathbf{F} : \mathbf{X} \mapsto \mathbf{Y}, \quad \mathbf{X} \in \mathbb{R}^n \ \mathbf{Y} \in \mathbb{R}^m$$

We train a suitable multilayer perceptron

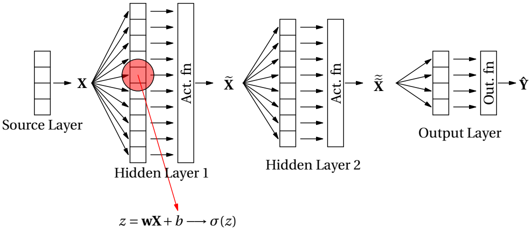


Overcome bottlenecks using deep learning

We wish to approximate the function

$$\mathbf{F} : \mathbf{X} \mapsto \mathbf{Y}, \quad \mathbf{X} \in \mathbb{R}^n \; \mathbf{Y} \in \mathbb{R}^m$$

We train a suitable multilayer perceptron

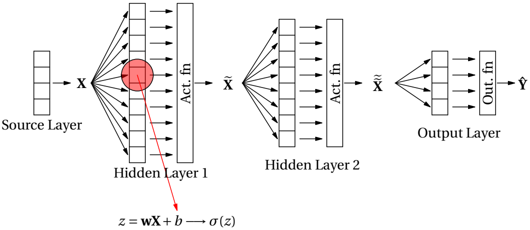


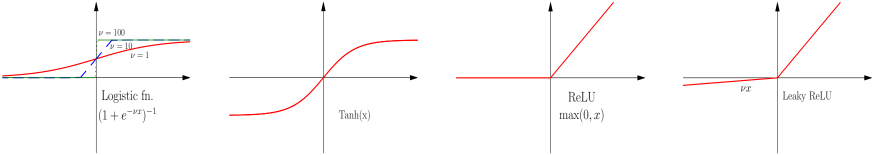
Overcome bottlenecks using deep learning

We wish to approximate the function

$$\mathbf{F} : \mathbf{X} \mapsto \mathbf{Y}, \quad \mathbf{X} \in \mathbb{R}^n \; \mathbf{Y} \in \mathbb{R}^m$$

We train a suitable multilayer perceptron



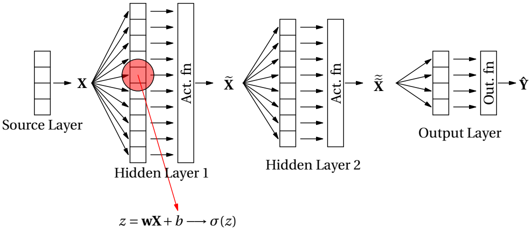


Overcome bottlenecks using deep learning

We wish to approximate the function

$$\mathbf{F} : \mathbf{X} \mapsto \mathbf{Y}, \quad \mathbf{X} \in \mathbb{R}^n \ \mathbf{Y} \in \mathbb{R}^m$$

We train a suitable multilayer perceptron



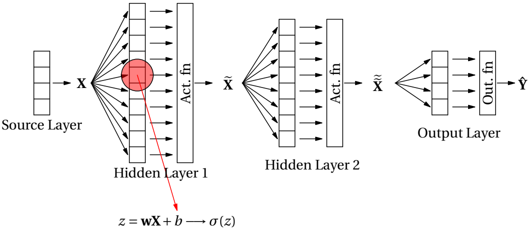
$$\hat{\mathbf{Y}} = \mathcal{O} \circ \sigma \circ H^L \circ \sigma \circ H^{L-1} \circ \ldots \circ \sigma \circ H^1(\mathbf{X}), \quad H^l(\tilde{\mathbf{X}}) = \mathbf{W}^l \tilde{\mathbf{X}} + \mathbf{b}^l$$

Overcome bottlenecks using deep learning

We wish to approximate the function

$$\mathbf{F} : \mathbf{X} \mapsto \mathbf{Y}, \quad \mathbf{X} \in \mathbb{R}^n \; \mathbf{Y} \in \mathbb{R}^m$$

We train a suitable multilayer perceptron



Find parameters $\theta = \{\mathbf{W}^l, \mathbf{b}^l\}$ that minimizes the loss function

$$\mathcal{L}(\mathbf{Y}_i, \hat{\mathbf{Y}}_i), \qquad (\mathbf{X}_i, \mathbf{Y}_i) \in \mathbb{T}.$$

Then $\hat{\mathbf{F}}(\theta) \approx \mathbf{F}$.

The hyperparameters

Based on some theory and prior experience, choose

- Network size – depth and width
- Activation function
- Loss function
- Regularization technique – to avoid overfitting
- Training and validation datasets
- Stopping criteria for training
- Optimizer
- Your expectation from the network!

Some applications we will see today

- Troubled-cell detection (Hesthaven and R.)
- Artificial viscosity (Discacciati, Hesthaven and R.)
- p-adaption (Wang, Hesthaven and R.)
- Local viscosity for spectral methods (Schwander, Hesthaven and R.)

Some applications we will see today

- Troubled-cell detection (Hesthaven and R.)
- Artificial viscosity (Discacciati, Hesthaven and R.)
- p-adaption (Wang, Hesthaven and R.)
- Local viscosity for spectral methods (Schwander, Hesthaven and R.)

The network is always trained offline!

Troubled-cell detector

Conservation laws

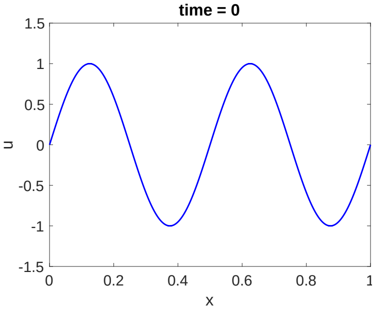
Consider

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

Non-linearity

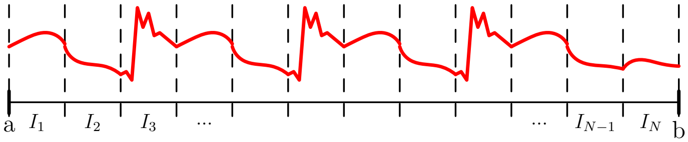
$\Longrightarrow$

Discontinuities in finite time



Detecting and limiting

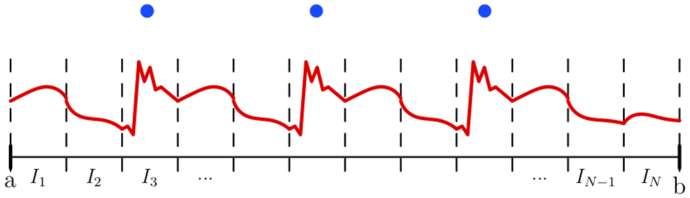
Strategy in the context of RKDG schemes:



Detecting and limiting

Strategy in the context of RKDG schemes:

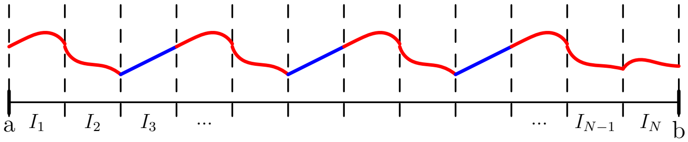
1. Find troubled-cells



Detecting and limiting

Strategy in the context of RKDG schemes:

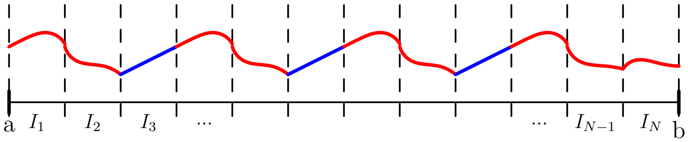
1. Find troubled-cells
2. Limit solution in flagged cells



Detecting and limiting

Strategy in the context of RKDG schemes:

1. Find troubled-cells
2. Limit solution in flagged cells



Issues:

- Problem-dependent parameters
- If insufficient cells marked $\longrightarrow$ re-appearance of Gibbs oscillations
- If excessive cells marked
  - Unnecessary computational cost
  - Loss of accuracy for strong limiters

Available troubled-cell indicators

- Minmod-based TVB limiter [Cockburn and Shu; Math. Comp. ‘98]
- Moment limiter [Biswas et al.; Appl. Numer. Math. ‘94]
- Modified moment limiter [Burbeau; JCP ‘01]
- Monotonicity preserving limiter [Suresh and Huynh; JCP ‘97]
- Modified MP limiter [Rider and Margolin; JCP ‘01]
- KXRCF indicator [Krivodonova et al.; App. Numer. Math. ‘04]
- Polynomial degree based limiter [Fu and Shu; JCP ‘17]
- Outlier detection using Tukey’s boxplot method [Vuik and Ryan; J. Sci. Comp. ‘16]
- ...

Available troubled-cell indicators

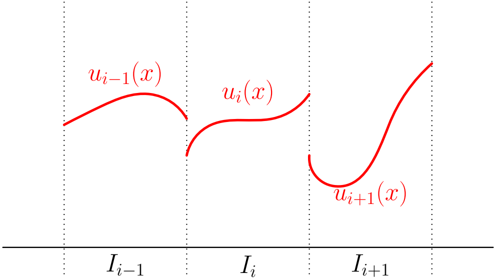
- Minmod-based TVB limiter [Cockburn and Shu; Math. Comp. ‘98]
- Moment limiter [Biswas et al.; Appl. Numer. Math. ‘94]
- Modified moment limiter [Burbeau; JCP ‘01]
- Monotonicity preserving limiter [Suresh and Huynh; JCP ‘97]
- Modified MP limiter [Rider and Margolin; JCP ‘01]
- KXRCF indicator [Krivodonova et al.; App. Numer. Math. ‘04]
- Polynomial degree based limiter [Fu and Shu; JCP ‘17]
- Outlier detection using Tukey’s boxplot method [Vuik and Ryan; J. Sci. Comp. ‘16]
- ...

An MLP-based detector

- Input $\mathbf{X} = ?$

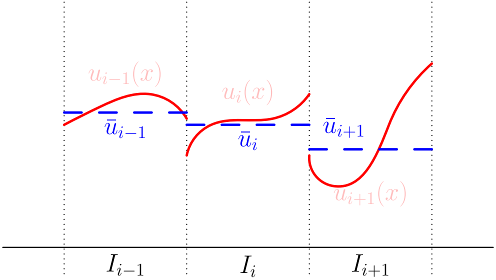
An MLP-based detector

- Input $\mathbf{X} = ?$



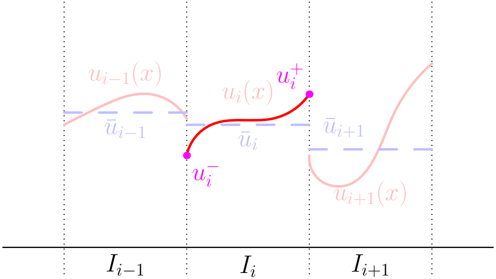
An MLP-based detector

- Input $\mathbf{X} = [\overline{u}_{i-1}, \overline{u}_i, \overline{u}_{i+1} \qquad]$



An MLP-based detector

- Input $\mathbf{X} = [\overline{u}_{i-1}, \overline{u}_i, \overline{u}_{i+1}, u_i^-, u_i^+] \in \mathbb{R}^5$



An MLP-based detector

- Input $\mathbf{X} = [\overline{u}_{i-1}, \overline{u}_i, \overline{u}_{i+1}, u_i^-, u_i^+] \in \mathbb{R}^5$ (Scaled)

An MLP-based detector

- Input $\mathbf{X} = [\overline{u}_{i-1}, \overline{u}_i, \overline{u}_{i+1}, u_i^-, u_i^+] \in \mathbb{R}^5$ (Scaled)
- 5 Hidden Layers with width 256, 128, 64, 32, 16

An MLP-based detector

- Input $\mathbf{X} = [\overline{u}_{i-1}, \overline{u}_i, \overline{u}_{i+1}, u_i^-, u_i^+] \in \mathbb{R}^5$ (Scaled)
- 5 Hidden Layers with width 256, 128, 64, 32, 16
- Leaky ReLU activation function with $\nu = 10^{-3}$

An MLP-based detector

- Input $\mathbf{X} = [\overline{u}_{i-1}, \overline{u}_i, \overline{u}_{i+1}, u_i^-, u_i^+] \in \mathbb{R}^5$ (Scaled)
- 5 Hidden Layers with width 256, 128, 64, 32, 16
- Leaky ReLU activation function with $\nu = 10^{-3}$
- Softmax output function

$$\hat{Y}^{(k)} \leftarrow \frac{e^{\hat{Y}^{(k)}}}{\sum_j e^{\hat{Y}^{(j)}}} \quad \in \quad [0,1] \quad \longrightarrow \quad \text{probabilities}$$

Output $\hat{\mathbf{Y}} = [\hat{Y}^{(0)}, \hat{Y}^{(1)}] \in [0,1]^2$

Troubled-cell if $\hat{Y}^{(0)} > 0.5$

An MLP-based detector

- Input $\mathbf{X} = [\overline{u}_{i-1}, \overline{u}_i, \overline{u}_{i+1}, u_i^-, u_i^+] \in \mathbb{R}^5$ (Scaled)
- 5 Hidden Layers with width 256, 128, 64, 32, 16
- Leaky ReLU activation function with $\nu = 10^{-3}$
- Softmax output function

$$\hat{Y}^{(k)} \leftarrow \frac{e^{\hat{Y}^{(k)}}}{\sum_j e^{\hat{Y}^{(j)}}} \quad \in \quad [0,1] \quad \longrightarrow \quad \text{probabilities}$$

Output $\hat{\mathbf{Y}} = [\hat{Y}^{(0)}, \hat{Y}^{(1)}] \in [0,1]^2$

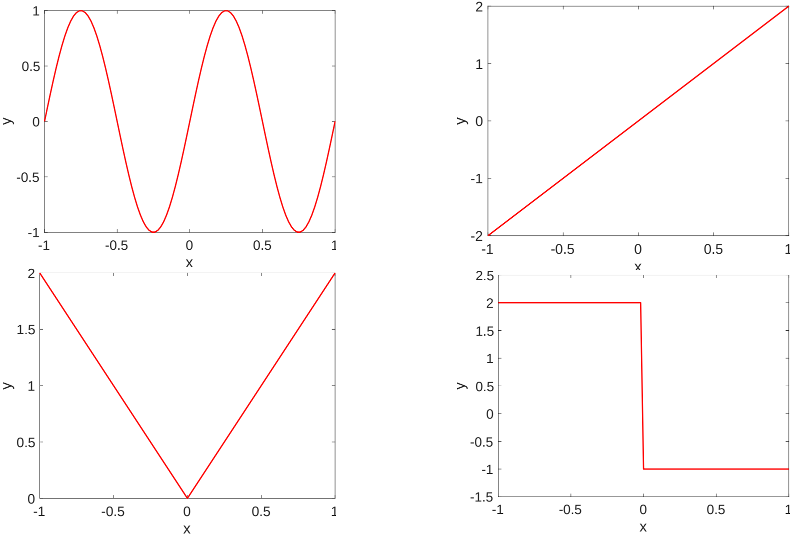
Troubled-cell if $\hat{Y}^{(0)} > 0.5$

- Cost functional: L2 regularized cross-entropy

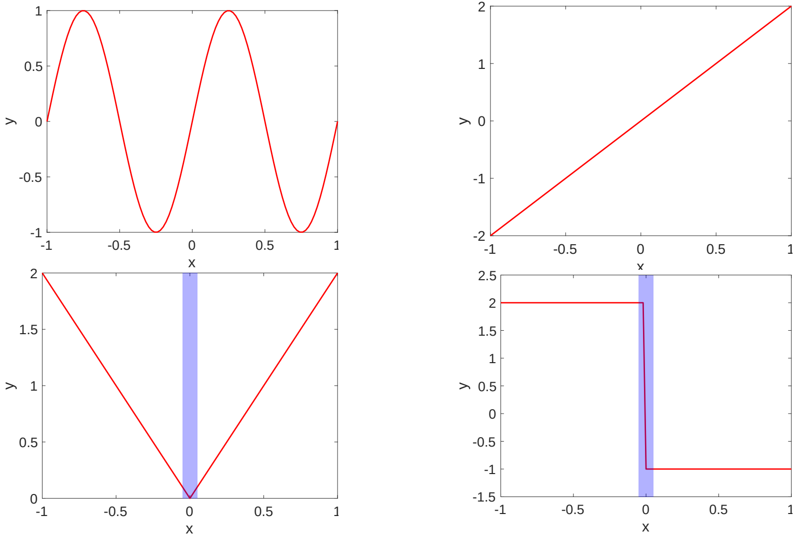
$$\mathcal{L} = -\sum_{i=1}^{K} \sum_{j=0}^{1} Y_i^{(j)} \log(\hat{Y}_i^{(j)}) \; + \; \lambda \|\mathbf{W}\|_2^2$$

Generating training data

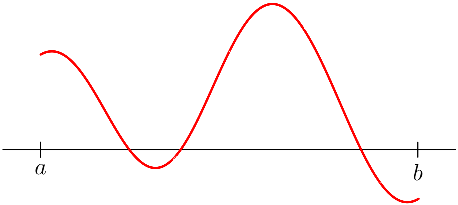
Types of functions used:



Types of functions used:



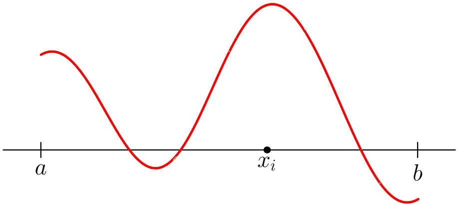
Generating training data



Data sampling is achieved by

- Choose a known function $u(x)$

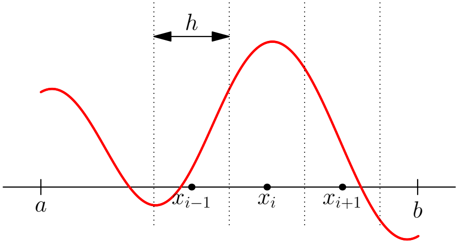
Generating training data



Data sampling is achieved by

- Choose a known function $u(x)$
- Pick a point x_i

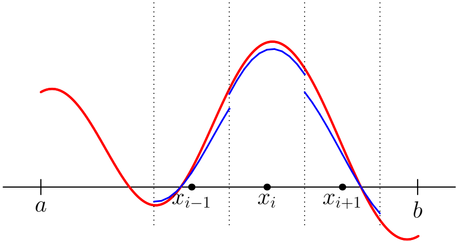
Generating training data



Data sampling is achieved by

- Choose a known function $u(x)$
- Pick a point x_i
- Pick a cell size h and make stencil

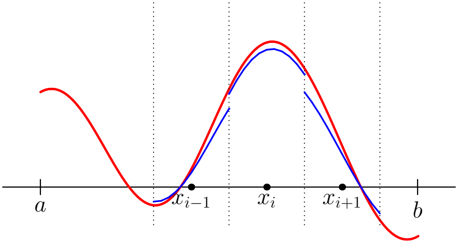
Generating training data



Data sampling is achieved by

- Choose a known function $u(x)$
- Pick a point x_i
- Pick a cell size h and make stencil
- Pick a degree r and approximate

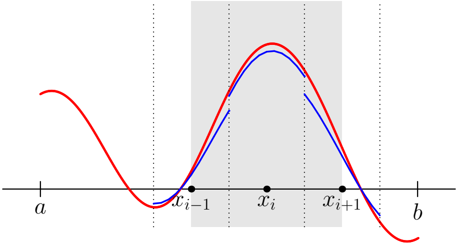
Generating training data



Data sampling is achieved by

- Choose a known function $u(x)$
- Pick a point x_i
- Pick a cell size h and make stencil
- Pick a degree r and approximate
- Extract needed data $[\overline{u}_{i-1}, \overline{u}_i, \overline{u}_{i+1}, u^+_{i-\frac{1}{2}}, u^-_{i+\frac{1}{2}}]$

Generating training data



Data sampling is achieved by

- Choose a known function $u(x)$
- Pick a point x_i
- Pick a cell size h and make stencil
- Pick a degree r and approximate
- Extract needed data $[\overline{u}_{i-1}, \overline{u}_i, \overline{u}_{i+1}, u^+_{i-\frac{1}{2}}, u^-_{i+\frac{1}{2}}]$
- Flag cell if discontinuity in $[x_{i-\frac{1}{2}} - h/2, x_{i+\frac{1}{2}} + h/2]$

Numerical setup

- Comparison with TVB limiter by setting parameter M
- In flagged cells, perform limited linear reconstruction with MUSCL limiter
- Legendre basis (Jacobi polynomials in 2D) with degree r
- Local Lax-Friedrich numerical flux
- Time integration with SSP-RK3

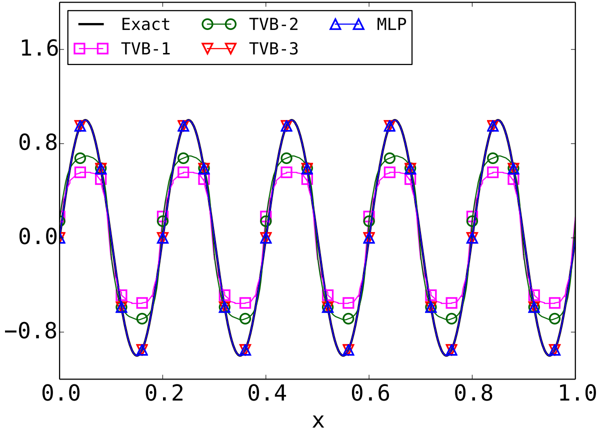
An artificial neural network as a troubled-cell indicator, by R. and Hesthaven; JCP vol. 367, pp. 166–191, 2018.

Detecting troubled-cells on two-dimensional unstructured grids using a neural network, by R. and Hesthaven, 2019 (submitted).

Linear advection: $u_t + u_x = 0$

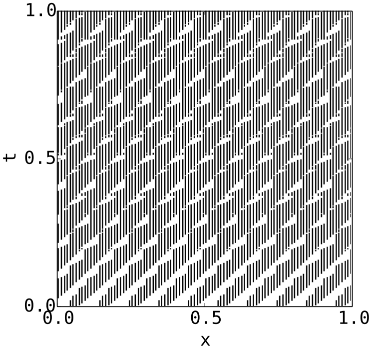
$$u_0(x) = \sin(10\pi x), \quad x \in [0,1], \quad T_f = 1, \quad N = 100, \quad r = 4$$

TVB-1 $\longrightarrow$ M=10
TVB-2 $\longrightarrow$ M=100
TVB-3 $\longrightarrow$ M=1000

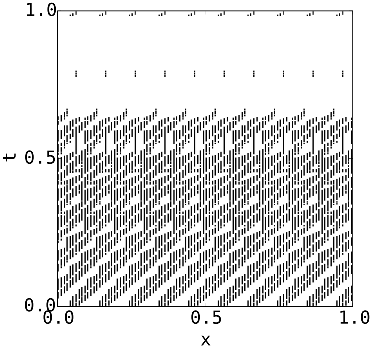


Linear advection: $u_t + u_x = 0$

$$u_0(x) = \sin(10\pi x), \quad x \in [0,1], \quad T_f = 1, \quad N = 100, \quad r = 4$$



TVB-1

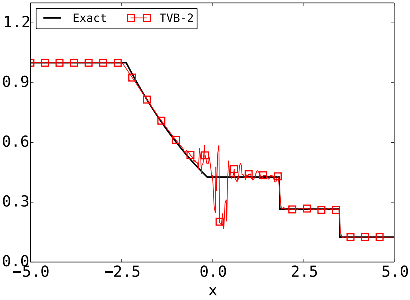


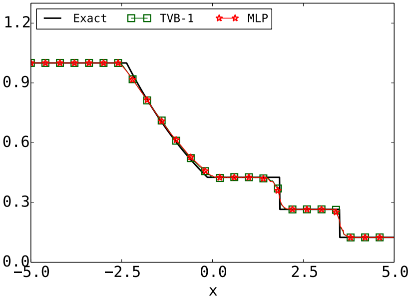
TVB-2

MLP and TVB-3 do not flag any cell

Euler equations: Sod shock tube

$$T_f = 2, \quad N = 100, \quad r = 4$$

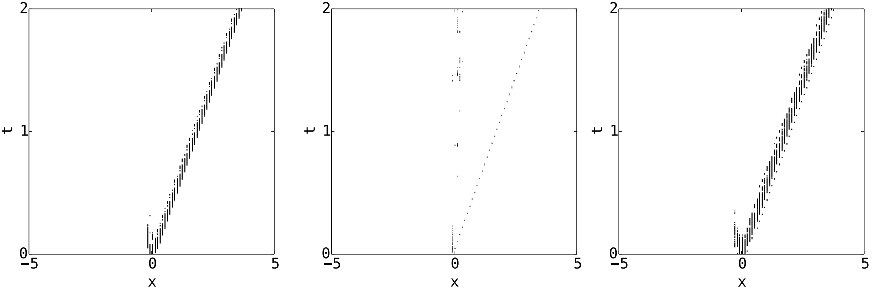




Loss of positivity with TVB-3

Euler equations: Sod shock tube

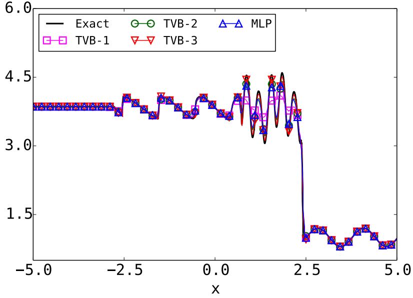
$$T_f = 2, \quad N = 100, \quad r = 4$$

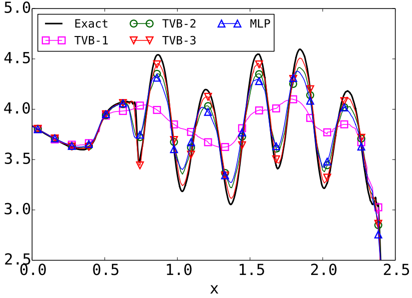


TVB-1 **TVB-2** **MLP**

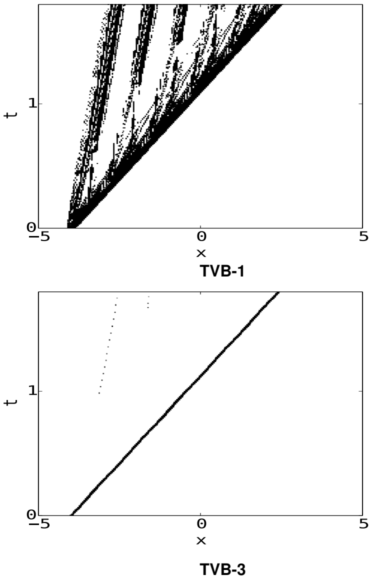
1D Euler equations: Shu-Osher problem

$$T_f = 1.8, \quad N = 256, \quad r = 4$$



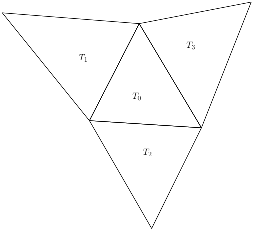


1D Euler equations: Shu-Osher problem



Extension to 2D

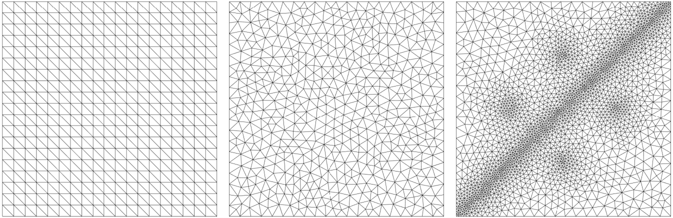
We consider unstructured triangular grids



MLP network with 5 hidden layers of width 20 each

Extension to 2D

We consider unstructured triangular grids

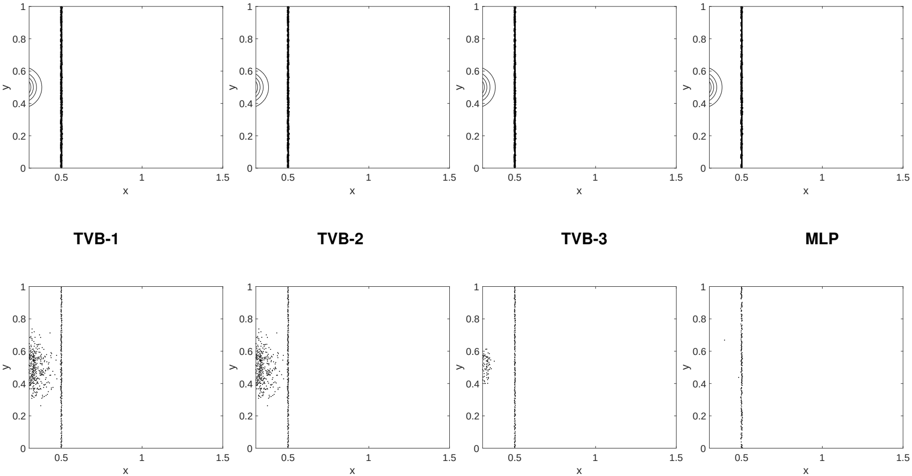


MLP network with 5 hidden layers of width 20 each
Training data constructed by:

- Interpolating functions to patch of 4 triangles.
- Extracting linear components, with $\mathbf{X} \in \mathbb{R}^{12}$.
- Label cell as troubled-cell if discontinuity present within circumscribed circle.

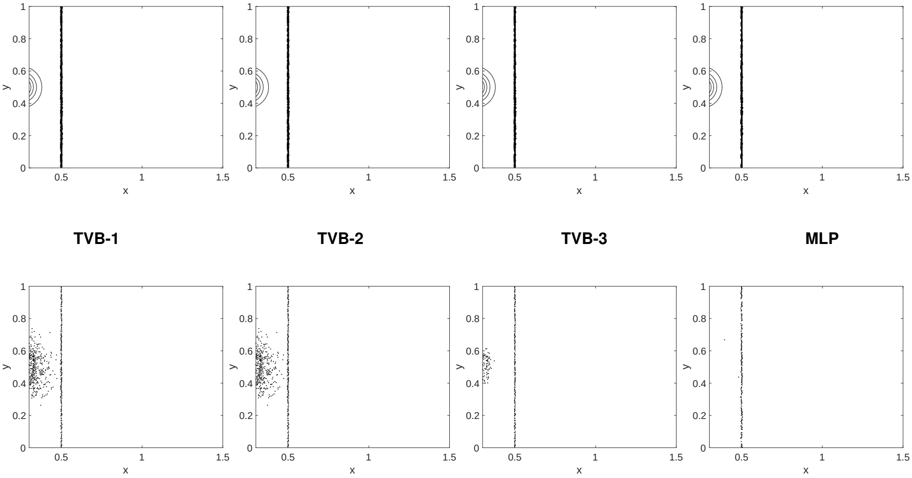
2D Euler equations: Shock-vortex

$T_f = 0.8, \quad h = 0.01, \quad r = 3,$ Barth-Jesperson limiter



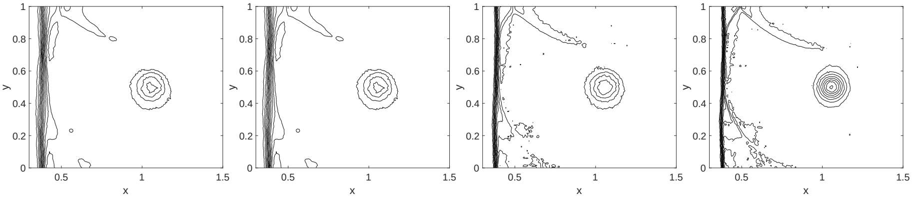
2D Euler equations: Shock-vortex

$$T_f = 0.8, \quad h = 0.01, \quad r = 3, \quad \text{Barth-Jesperson limiter}$$

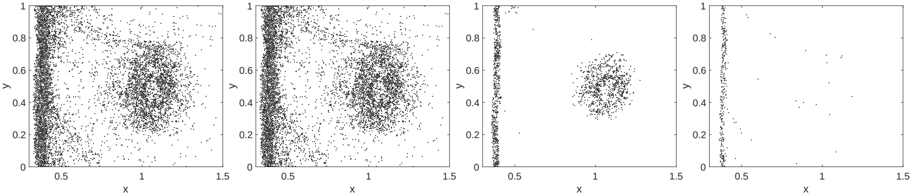


2D Euler equations: Shock-vortex

$T_f = 0.8, \quad h = 0.01, \quad r = 3,$ Barth-Jesperson limiter



TVB-1 **TVB-2** **TVB-3** **MLP**



Predicting artificial viscosity

Adding artificial viscosity

Now consider the problem

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{f}(\mathbf{u}) = \nabla \cdot \mathbf{g}, \quad \mathbf{g} = \mu \mathbf{w}, \quad \mathbf{w} = \nabla \mathbf{u}$$

Viscosity depends on **u** and locally controls oscillations.

$$\mu \sim h|\mathbf{f}'(\mathbf{u})|$$

Several artificial viscosity models exist

- MDH: Highest Modal Decay [Persson and Peraire; 14th AIAA meet, 2016]
- MDA: Averaged Modal Decay [Klöckner et al.; Math. Mod. Nat. Phen., 2011]
- EV: Entropy Viscosity [Guermond et al.; JCP, 2011]

Dependent on problem specific parameters

Training an MLP

Network architecture:

- Input: Full data from each element (no neighbours)
- 5 hidden layers of width 10 each with Leaky ReLU
- Softplus output function, i.e., $f(x) = \log(1 + e^x)$
- Mean Squared Error cost function

Training an MLP

Network architecture:

- Input: Full data from each element (no neighbours)
- 5 hidden layers of width 10 each with Leaky ReLU
- Softplus output function, i.e., $f(x) = \log(1 + e^x)$
- Mean Squared Error cost function

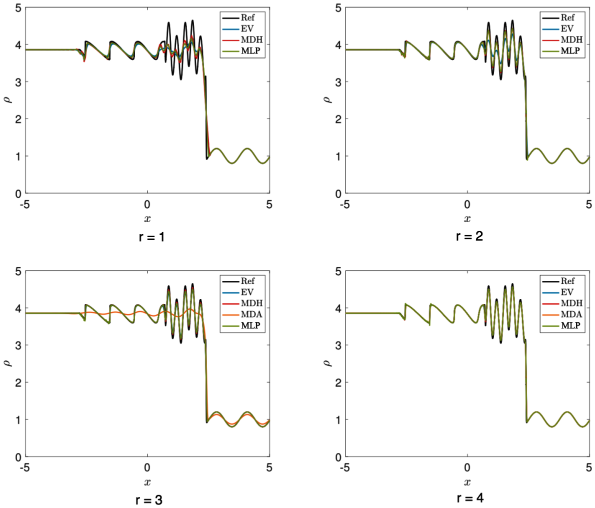
Training/validation sets:

- Using numerical solution of conservation laws (only linear adv. and Burgers)
- Target viscosity: viscosity corresponding to "best" model
- Different network for each degree r

Controlling oscillations in high-order Discontinuous Galerkin schemes using artificial viscosity tuned by neural networks, by Discacciti, Hesthaven and R.; 2019 (submitted).

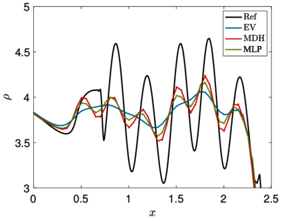
1D Euler equations: Shu-Osher

$$T_f = 1.8, \quad h = 10/200$$

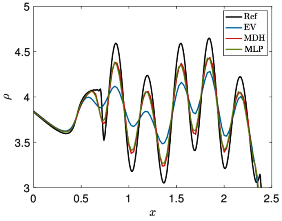


1D Euler equations: Shu-Osher

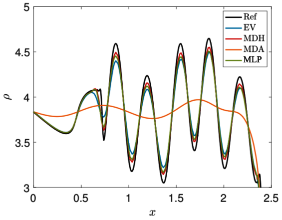
$$T_f = 1.8, \quad h = 10/200$$



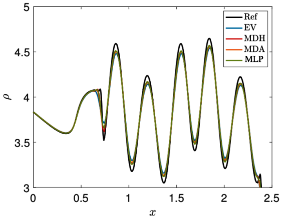
r = 1



r = 2



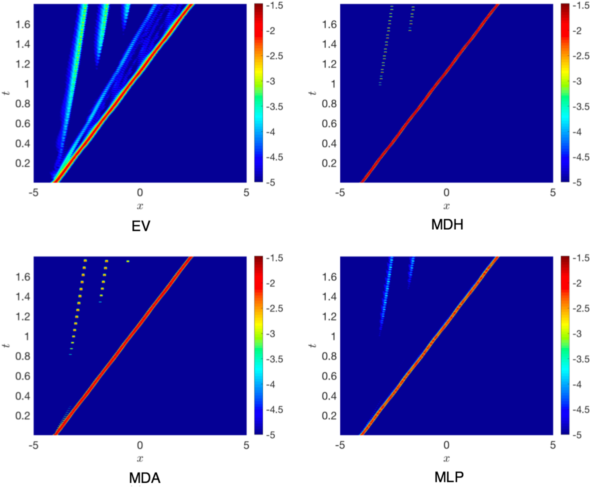
r = 3



r = 4

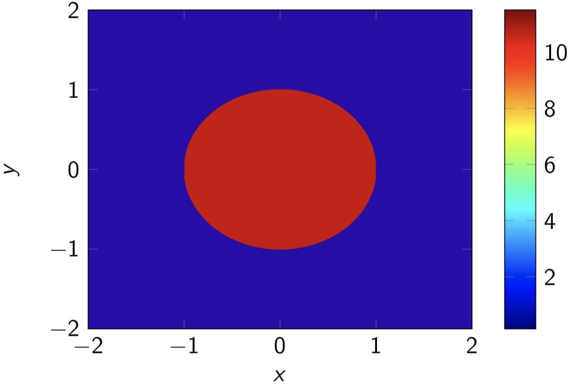
1D Euler equations: Shu-Osher

$$T_f = 1.8, \quad h = 10/200, \quad r = 4$$



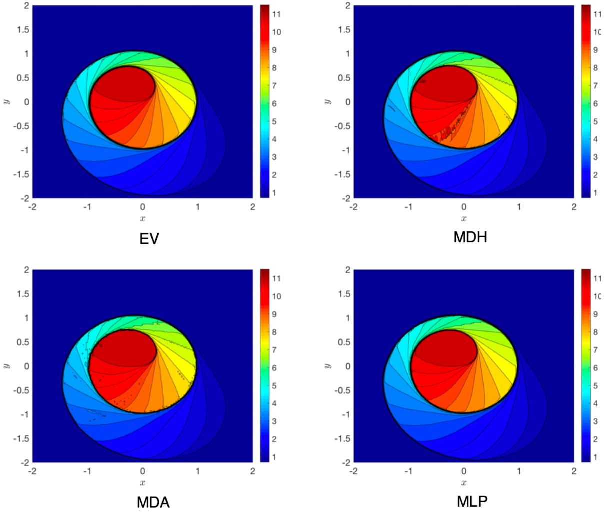
2D KPP equations: Rotating wave

$$\mathbf{f}(u) = (\sin u, \cos u), \quad T_f = 1, \quad h = 4\sqrt{2}/120, \quad r = 4$$

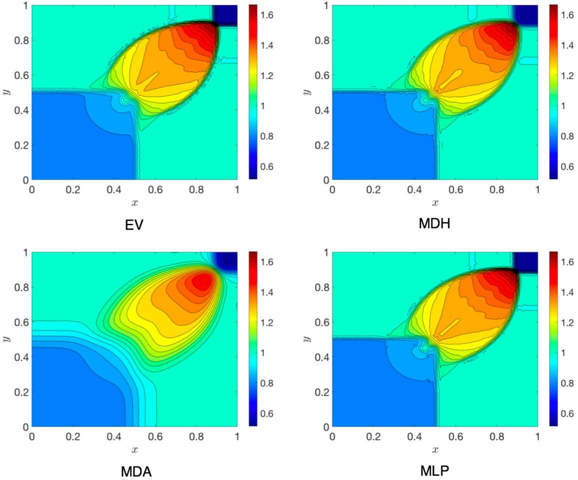


2D KPP equations: Rotating wave

$$\mathbf{f}(u) = (\sin u, \cos u), \quad T_f = 1, \quad h = 4\sqrt{2}/120, \quad r = 4$$

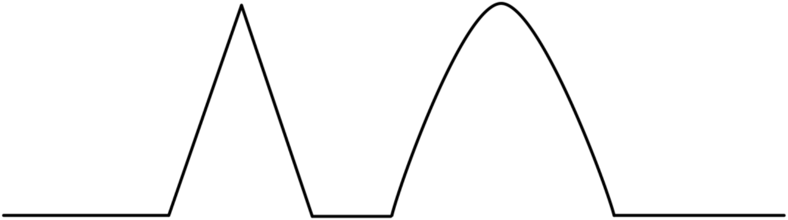


2D Euler equations: Riemann Problem config. 12 ($r = 3$)

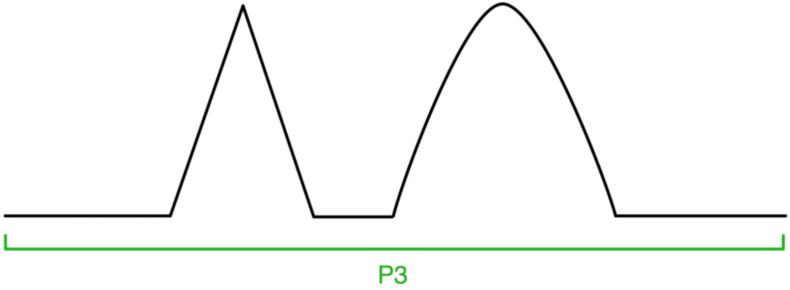


Learning p-adaption

Idea of p-adaption

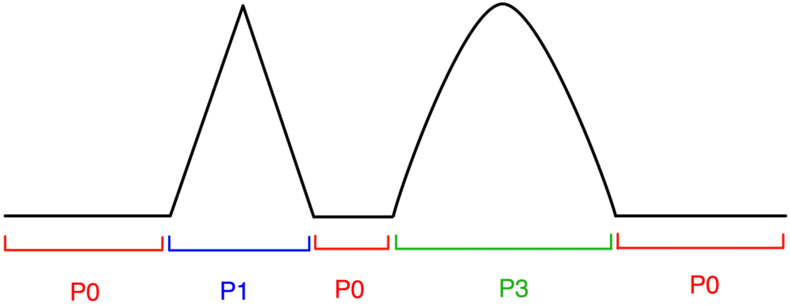


Idea of p-adaption



In each element: $u_i^h(x) = \sum_{l=0}^{p_{max}} \hat{u}_l^i \phi_l(x)$

Idea of p-adaption



In each element: $u_i^h(x) = \sum_{l=0}^{p_{max}} \hat{u}_l^i \phi_l(x)$

Adapt P: $u_i^h(x) = \sum_{l=0}^{p_i} \hat{u}_l^i \phi_l(x), \;\; 0 \leq p_i \leq p_{max}$

How to choose p?

Existing methods:

- p-refinement indicator based on a posteriori estimator [Naddei et al., JCP, 2019]
- Dynamic p-adaption based on flow gradients [Burbeau and Sagaut, Comp. Fluids, 2005; Kubatko et al., Comp. Meth. App. Mech. Engg., 2009]

Dependent on problem specific parameters

How to choose p?

MLP based p refinement for a pre-defined p_{max}:

- **Input:** Full modal coefficients of 3-cell patch

$$\mathbf{X} = [\hat{u}_0^{i-1}, ..., \hat{u}_{p_{max}}^{i-1},\ \ \hat{u}_0^{i}, ..., \hat{u}_{p_{max}}^{i},\ \ \hat{u}_0^{i+1}, ..., \hat{u}_{p_{max}}^{i+1}] \in \mathbb{R}^{3(p_{max}+1)}$$

- **Output:** $\hat{\mathbf{Y}} \in \mathbb{R}^{p_{max}+2}$, where

$$\begin{aligned} \hat{\mathbf{Y}}_j,\ 0 \le j \le p_{max} \quad &\longrightarrow \quad \text{order } p_j \ \ \text{(smooth)} \\ \hat{\mathbf{Y}}_{p_{max}+1} \quad &\longrightarrow \quad \text{discontinuity} \end{aligned}$$

How to choose p?

MLP based p refinement for a pre-defined p_{max}:

- **Input:** Full modal coefficients of 3-cell patch

$$\mathbf{X} = [\hat{u}_0^{i-1}, ..., \hat{u}_{p_{max}}^{i-1},\ \ \hat{u}_0^{i}, ..., \hat{u}_{p_{max}}^{i},\ \ \hat{u}_0^{i+1}, ..., \hat{u}_{p_{max}}^{i+1}] \in \mathbb{R}^{3(p_{max}+1)}$$

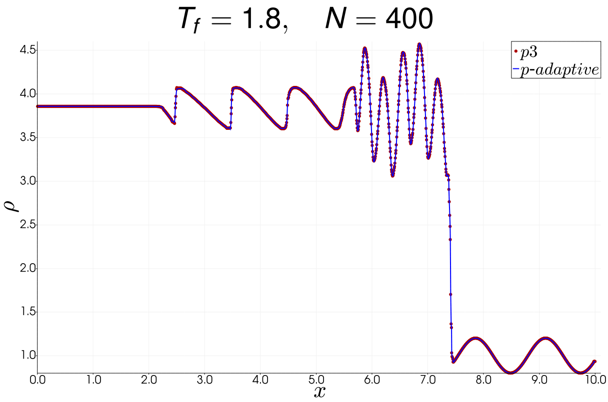
- **Output:** $\hat{\mathbf{Y}} \in \mathbb{R}^{p_{max}+2}$, where

$$\begin{aligned} \hat{\mathbf{Y}}_j,\ 0 \le j \le p_{max} \quad &\longrightarrow \quad \text{order } p_j \ \ \text{(smooth)} \\ \hat{\mathbf{Y}}_{p_{max}+1} \quad &\longrightarrow \quad \text{discontinuity} \end{aligned}$$

For $p_{max} = 3$:

- MLP with 5 hidden layers of width 20 each.
- Trained on samples generated from polynomials and generalized (linear) Riemann data.

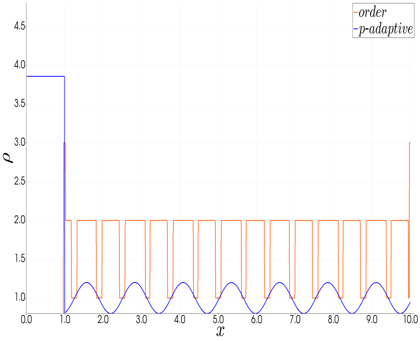
Euler equations: Shu-Osher

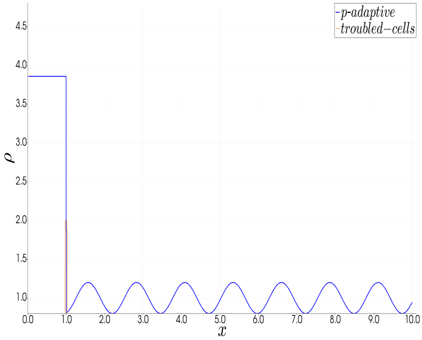


Multi-dimensional WBAP limiter [Li et al., JCP, 2011] used near discontinuities.

Euler equations: Shu-Osher

$$T_f = 1.8, \quad N = 400$$

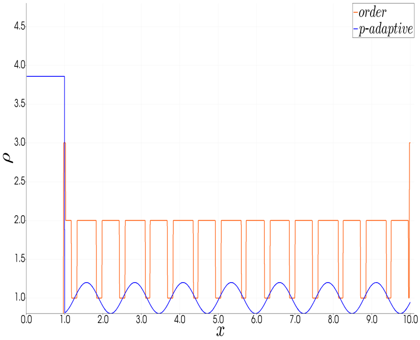


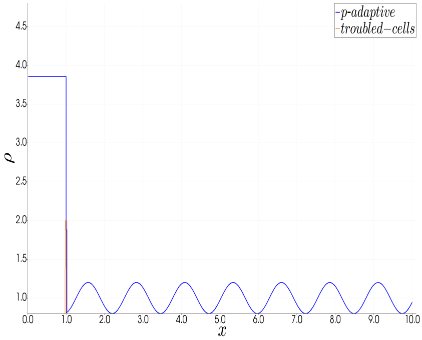


Multi-dimensional WBAP limiter [Li et al., JCP, 2011] used near discontinuities.

Euler equations: Shu-Osher

$$T_f = 1.8, \quad N = 400$$

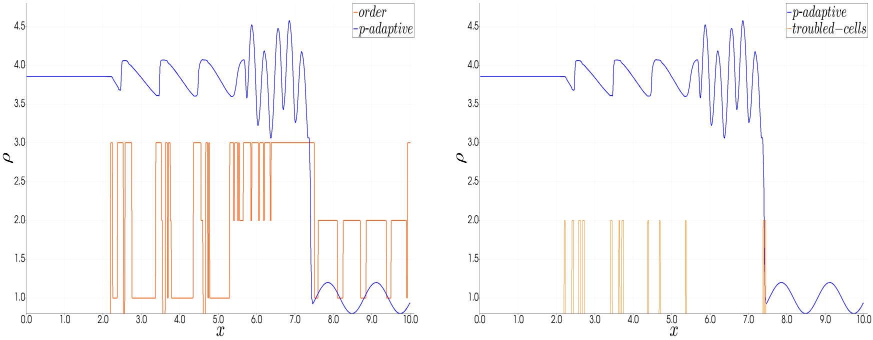




Multi-dimensional WBAP limiter [Li et al., JCP, 2011] used near discontinuities.

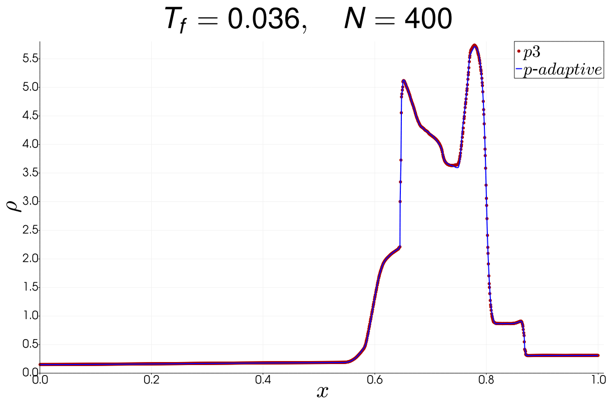
Euler equations: Shu-Osher

$$T_f = 1.8, \quad N = 400$$



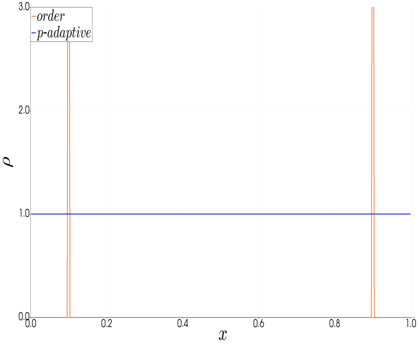
Multi-dimensional WBAP limiter [Li et al., JCP, 2011] used near discontinuities.

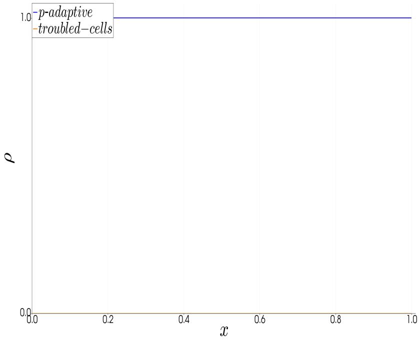
Euler equations: Blastwave



Euler equations: Blastwave

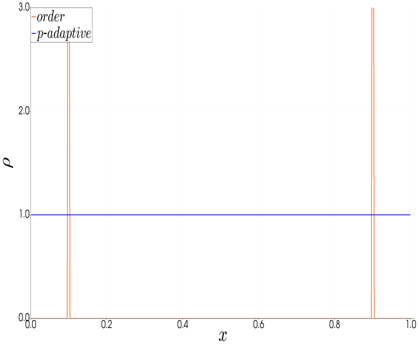
$$T_f = 0.036, \quad N = 400$$

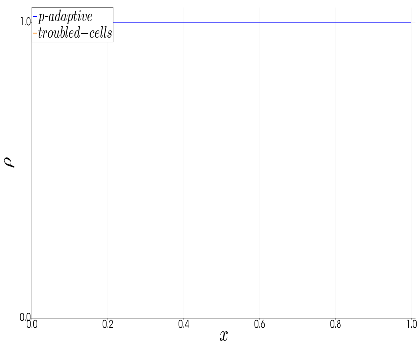




Euler equations: Blastwave

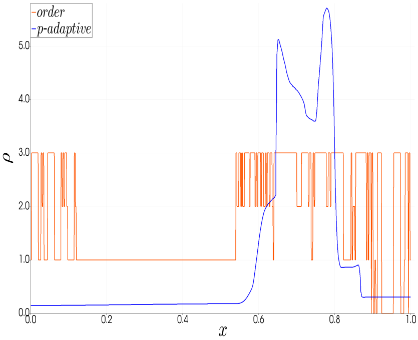
$$T_f = 0.036, \quad N = 400$$

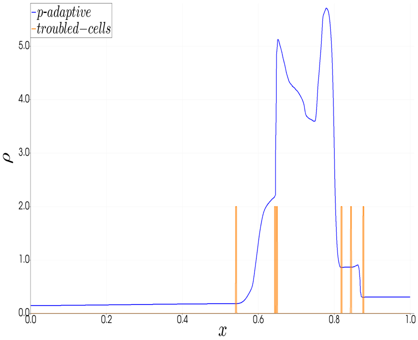




Euler equations: Blastwave

$$T_f = 0.036, \quad N = 400$$





Computational cost

Table: Computational cost for the Blast Waves problem. Unit of cost: second (s).

	total	p-adaption	residual	limiter	time-stepping
p3	155.17	0	31.68	116.26	6.44
p-adaptive	246.12	113.98	14.69	113.52	3.34

Table: Computational cost for the Shu-Osher problem. Unit of cost: second (s).

	total	p-adaption	residual	limiter	time-stepping
p3	101.76	0	21.00	76.20	4.05
p-adaptive	165.72	75.57	11.10	75.88	2.71

Local smoothening for global schemes

Fourier spectral methods

Assuming periodic boundary conditions on $[0, L]$

$$u(x,t) \approx u_h(x,t) = \sum_{n=-N}^{N} \tilde{u}_n(t) \exp\left(i\, n\, 2\pi \frac{x}{L}\right) \quad \longrightarrow \quad \text{global}$$

Define $2N+1$ collocation points

$$x_j = \frac{L}{2N+1} j, \quad j = 0, \ldots 2N$$

Solve for

$$\frac{\mathrm{d}\mathbf{u}(t)}{\mathrm{d}t} + \mathbf{D}\mathbf{u}(t) = 0$$

where $\mathbf{u}(t) = [u_h(x_0,t), \ldots, u_h(x_{2N},t)]^\top$.

Handling Gibbs oscillations:

- Add hyper-viscosity
  - Second-order: cures oscillations but smears solution
  - High-order: better accuracy but local oscillations remain
- Post-processing exponential filter – same issue as above
- Use Fourier-Padé reconstruction – oscillations still persist

Fourier spectral methods

Handling Gibbs oscillations:

- Add hyper-viscosity
  - Second-order: cures oscillations but smears solution
  - High-order: better accuracy but local oscillations remain
- Post-processing exponential filter – same issue as above
- Use Fourier-Padé reconstruction – oscillations still persist

A simpler approach using MLPs

$$\frac{\mathrm{d}\mathbf{u}(t)}{\mathrm{d}t} + \mathbf{D}\mathbf{u}(t) = \mathbf{D}[\mu \otimes \mathbf{D}\mathbf{u}]$$

where μ varies locally based on local-regularity

Local viscosity with an MLP

Evaluate viscosity as a classification problem [Klöckner et al., 2011; Yu et al., 2018]

The network predicts the local regularity τ based on a seven-point stencil

τ	1	2	3	4
Reg.	Disc.	C0/C1	C1/C2	C2

Local viscosity with an MLP

Evaluate viscosity as a classification problem [Klöckner et al., 2011; Yu et al., 2018]

The network predicts the local regularity τ based on a seven-point stencil

τ	1	2	3	4
Reg.	Disc.	C0/C1	C1/C2	C2

$$\mu = \mu_{max} \begin{cases} 1 - (\tau - 1)/2 & \text{if } 1 \leq \tau \leq 3 \\ 0 & \text{if } \tau = 4 \end{cases}$$

where $\mu_{max} = \frac{h}{2} \max(|f'(u)|)$.

Local viscosity with an MLP

Network architecture:

- Input: $\mathbf{X} \in \mathbb{R}^7$; Output: $\hat{\mathbf{Y}} \in \mathbb{R}^4$
- 3 hidden layers of width 16 each
- A dropout layer is used for regularization

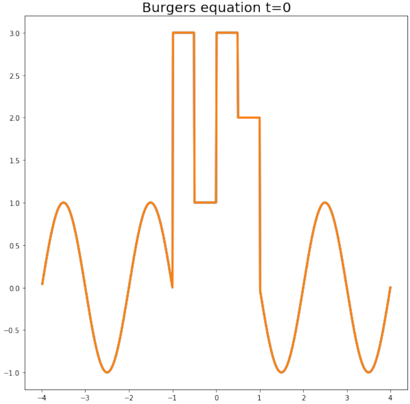
Training/validation data sets created used functions of varying regularity.

Viscosity used along with a high-order exponential filter

- order 12 if $\tau = 1, 2, 3$
- order 14 if $\tau = 4$

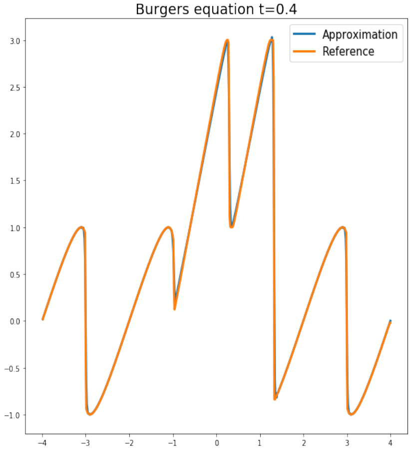
Burgers equation

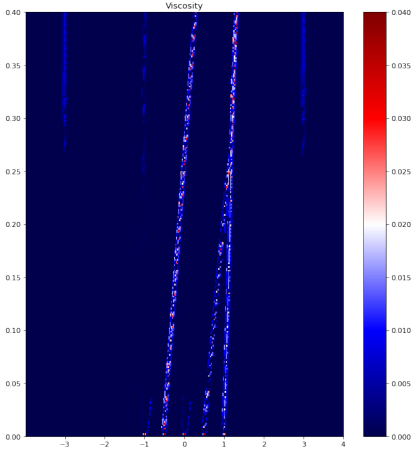
$$N = 600, \quad T_f = 0.4$$



Burgers equation

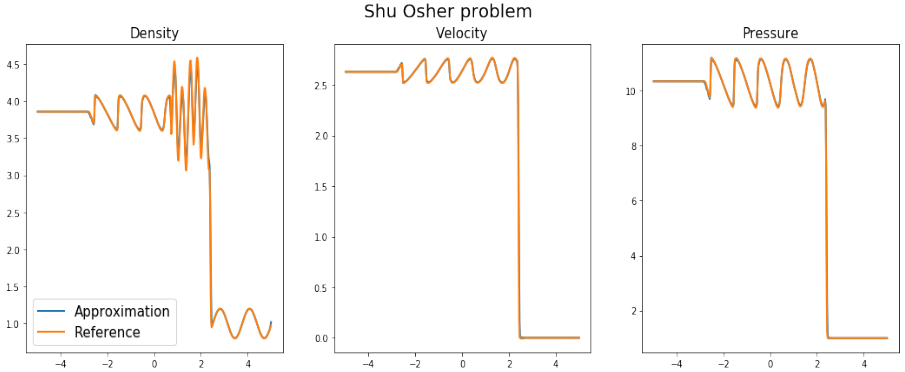
$$N = 600, \quad T_f = 0.4$$





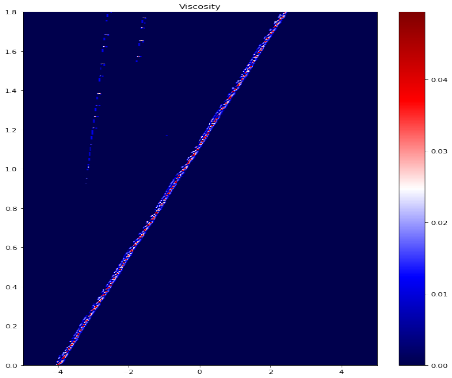
Euler equations: Shu-Osher

$$N = 600, \quad T_f = 1.8$$



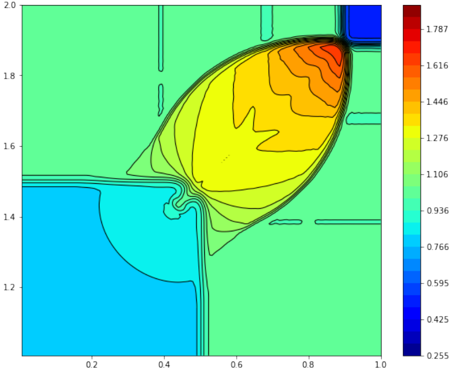
Euler equations: Shu-Osher

$$N = 600, \quad T_f = 1.8$$



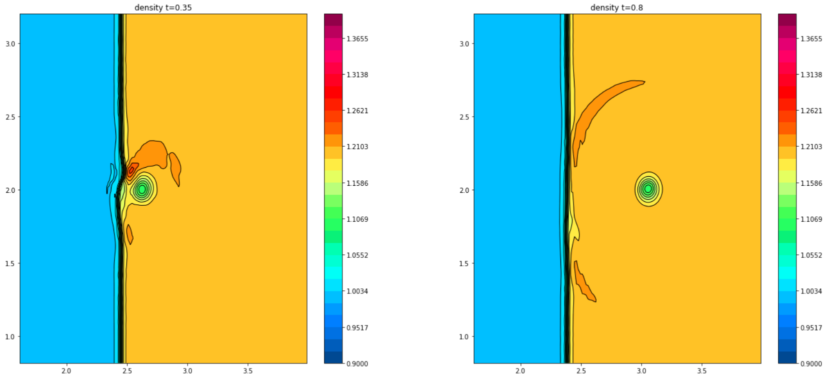
2D Euler equations: Riemann problem config. 12

$$N_x = N_y = 100$$



2D Euler equations: Shock-vortex

$$N_x = N_y = 200$$



Conclusions and future scope

- Demonstrated that deep learning can be used as a surrogate – both for classification and regression
- Networks trained (once) offline and then used for any conservation law
- Useful in constructing methods free of problem-dependent parameters
- Need to use domain-knowledge to construct training sets

Conclusions and future scope

- Demonstrated that deep learning can be used as a surrogate – both for classification and regression
- Networks trained (once) offline and then used for any conservation law
- Useful in constructing methods free of problem-dependent parameters
- Need to use domain-knowledge to construct training sets

Troubled-cell detector: what next?

- Extension to 3D
- Network for general hybrid meshes – how to handle variable input size?
- Possible to estimate angle of discontinuity?

Conclusions and future scope

- Demonstrated that deep learning can be used as a surrogate – both for classification and regression
- Networks trained (once) offline and then used for any conservation law
- Useful in constructing methods free of problem-dependent parameters
- Need to use domain-knowledge to construct training sets

Artificial viscosity: what next?

- Extension to 3D and hybrid grids
- Explore more sophisticated subcell resolution based on network prediction
- Automate learning – perhaps using reinforcement learning

Conclusions and future scope

- Demonstrated that deep learning can be used as a surrogate – both for classification and regression
- Networks trained (once) offline and then used for any conservation law
- Useful in constructing methods free of problem-dependent parameters
- Need to use domain-knowledge to construct training sets

p-**adaption:** what next?

- Extension to higher-dimensions and hybrid grids
- Use network for *hp*-adaption

Conclusions and future scope

- Demonstrated that deep learning can be used as a surrogate – both for classification and regression
- Networks trained (once) offline and then used for any conservation law
- Useful in constructing methods free of problem-dependent parameters
- Need to use domain-knowledge to construct training sets

Local viscosity for global methods: what next?

- Test on non-uniform grids
- Explore similar ideas for ROM.

Don't replace but enhance existing numerical frameworks with deep networks

* Don't replace but enhance existing numerical frameworks with deep networks

*if possible