

# Adaptive Neural Networks for Reduced Order Modeling

**Davide Torlo, Federico Pichi**

Dipartimento di Matematica “Guido Castelnuovo”, Università di Roma La Sapienza, Italy  
[davidetorlo.it](http://davidetorlo.it)

Trieste - 2nd September 2025



**SAPIENZA**  
UNIVERSITÀ DI ROMA

# Table of contents

---

① Adaptive NN

② Adaptive NN for Reduced Order Modeling  
Simulations

③ Conclusions

# Table of contents

---

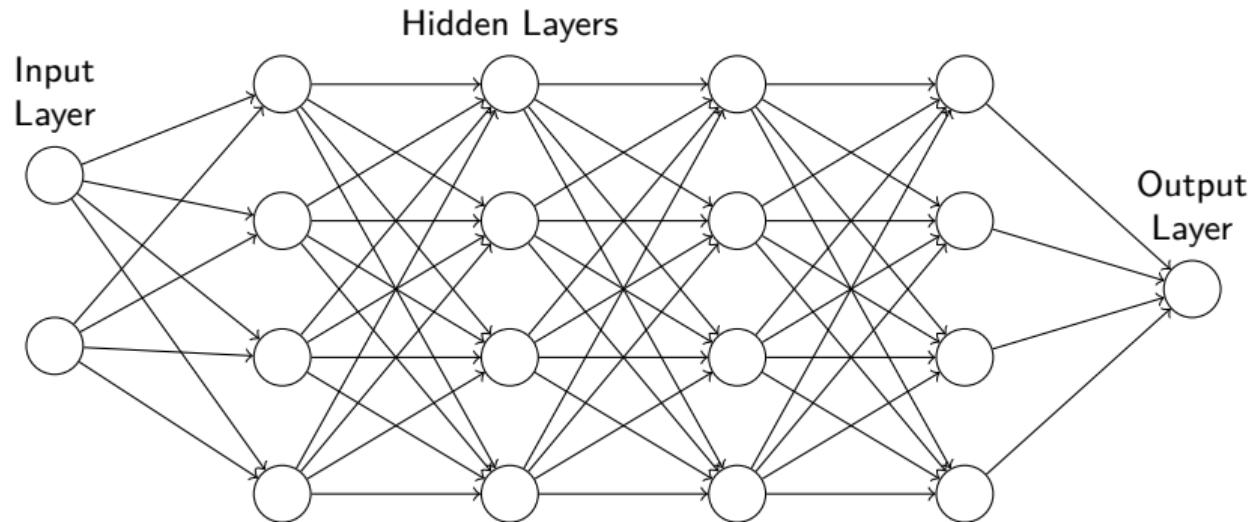
① Adaptive NN

② Adaptive NN for Reduced Order Modeling  
Simulations

③ Conclusions

## Motivation: too deep too wide networks

---



- Universal approximators
- Adding many layers increase expressibility
- Countless applications
- Is this optimal?
- Do we really need so many layers?
- Could we save some energy?

## Rediscovering shallow NN: ReLU 1 hidden layer

### 1 hidden layer NN

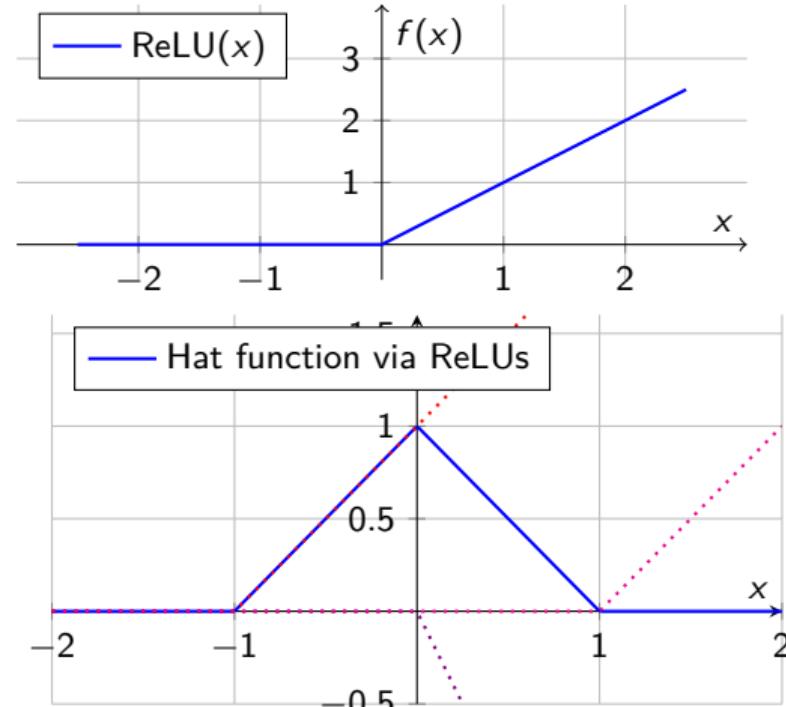
- 1 hidden layer
- few neurons
- ReLU activation function  $\max(x, 0) = x^+$
- Hat function: 1 hidden layer, 3 neurons

$$h(x) = (x + 1)^+ - 2(x)^+ + (x - 1)^+$$

- Every breaking point one neuron
- Exploit Finite Element knowledge  
approximation functions: Piecewise linear functions
- Also in more dimensions, e.g.

$$(ax + by)^+$$

- In D dimensions breaking manifolds are hyper-planes (lines in 2D)



# Rediscovering shallow NN: ReLU 1 hidden layer

## 1 hidden layer NN

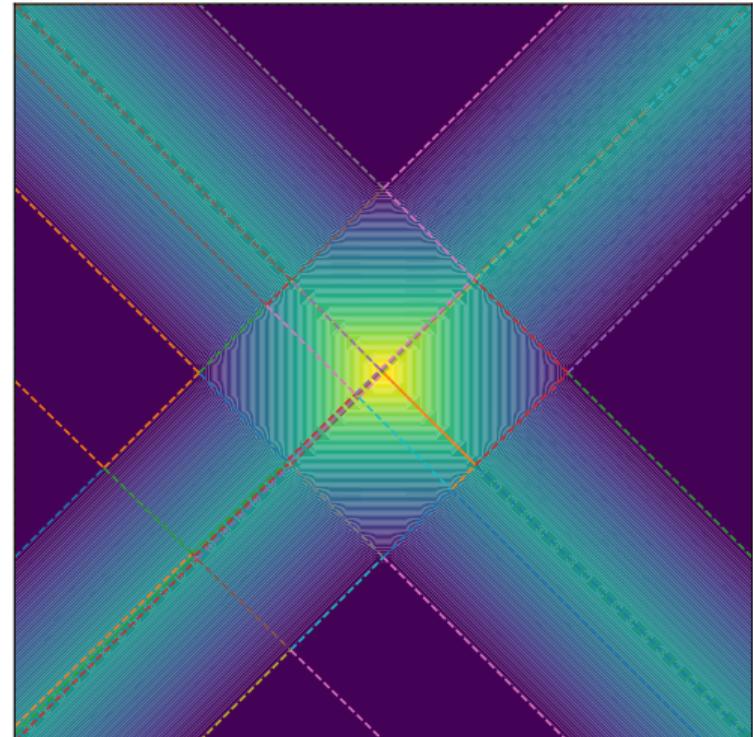
- 1 hidden layer
- few neurons
- ReLU activation function  $\max(x, 0) = x^+$
- Hat function: 1 hidden layer, 3 neurons

$$h(x) = (x + 1)^+ - 2(x)^+ + (x - 1)^+$$

- Every breaking point one neuron
- Exploit Finite Element knowledge  
approximation functions: Piecewise linear functions
- Also in more dimensions, e.g.

$$(ax + by)^+$$

- In D dimensions breaking manifolds are hyper-planes (lines in 2D)

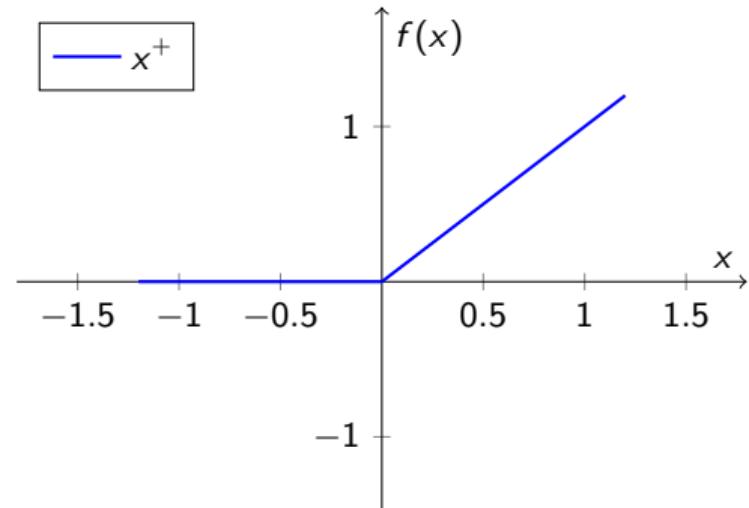


### 2 hidden layers NN

- Speed up the process of geometrical subdivision
- In 1D, for example, easy to discretize discontinuities up to  $\varepsilon$  with 2 hidden layers 1 neuron each

$$N(x) = 1 - \left(1 - \frac{1}{\varepsilon}(x - x_d)^+\right)^+$$

- Less sensitive to hyperparameters (to do the same with 1 hidden layer the different weights have to match exactly)
- **Fully interpretable**

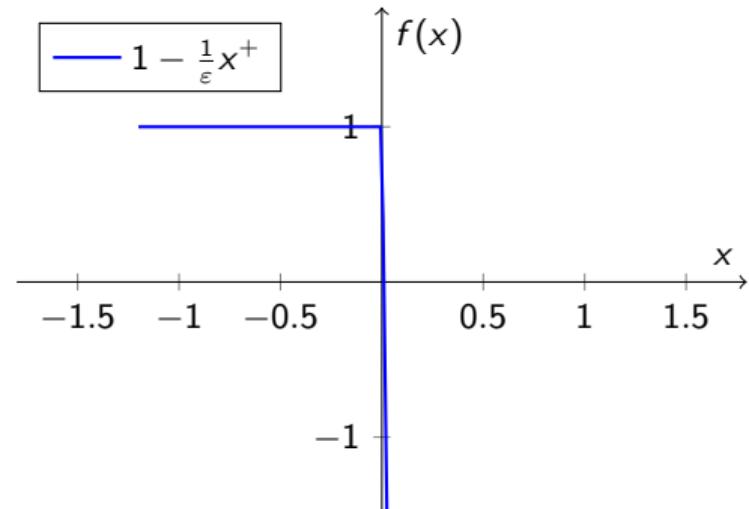


### 2 hidden layers NN

- Speed up the process of geometrical subdivision
- In 1D, for example, easy to discretize discontinuities up to  $\varepsilon$  with 2 hidden layers 1 neuron each

$$N(x) = 1 - \left(1 - \frac{1}{\varepsilon}(x - x_d)^+\right)^+$$

- Less sensitive to hyperparameters (to do the same with 1 hidden layer the different weights have to match exactly)
- **Fully interpretable**

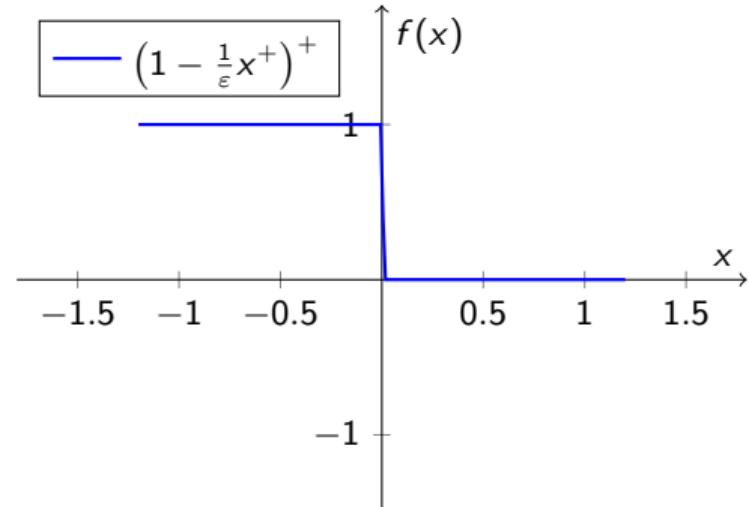


### 2 hidden layers NN

- Speed up the process of geometrical subdivision
- In 1D, for example, easy to discretize discontinuities up to  $\varepsilon$  with 2 hidden layers 1 neuron each

$$N(x) = 1 - \left(1 - \frac{1}{\varepsilon}(x - x_d)^+\right)^+$$

- Less sensitive to hyperparameters (to do the same with 1 hidden layer the different weights have to match exactly)
- **Fully interpretable**

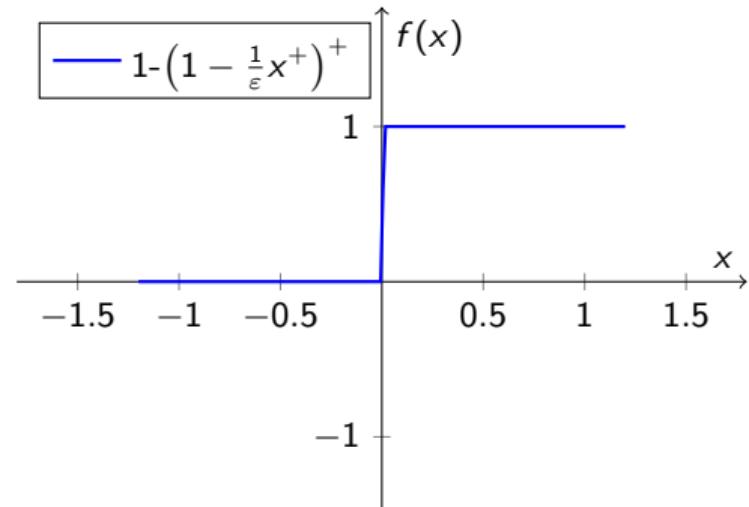


### 2 hidden layers NN

- Speed up the process of geometrical subdivision
- In 1D, for example, easy to discretize discontinuities up to  $\varepsilon$  with 2 hidden layers 1 neuron each

$$N(x) = 1 - \left(1 - \frac{1}{\varepsilon}(x - x_d)^+\right)^+$$

- Less sensitive to hyperparameters (to do the same with 1 hidden layer the different weights have to match exactly)
- **Fully interpretable**

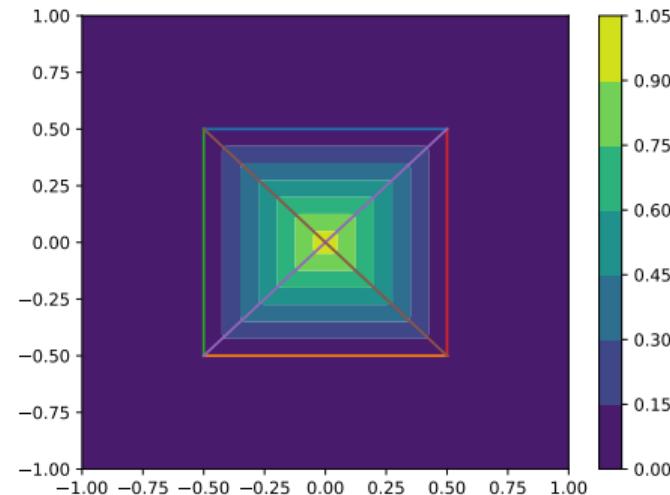


## 2 hidden layers NN

- Speed up the process of geometrical subdivision
- In 1D, for example, easy to discretize discontinuities up to  $\varepsilon$  with 2 hidden layers 1 neuron each

$$N(x) = 1 - \left(1 - \frac{1}{\varepsilon}(x - x_d)^+\right)^+$$

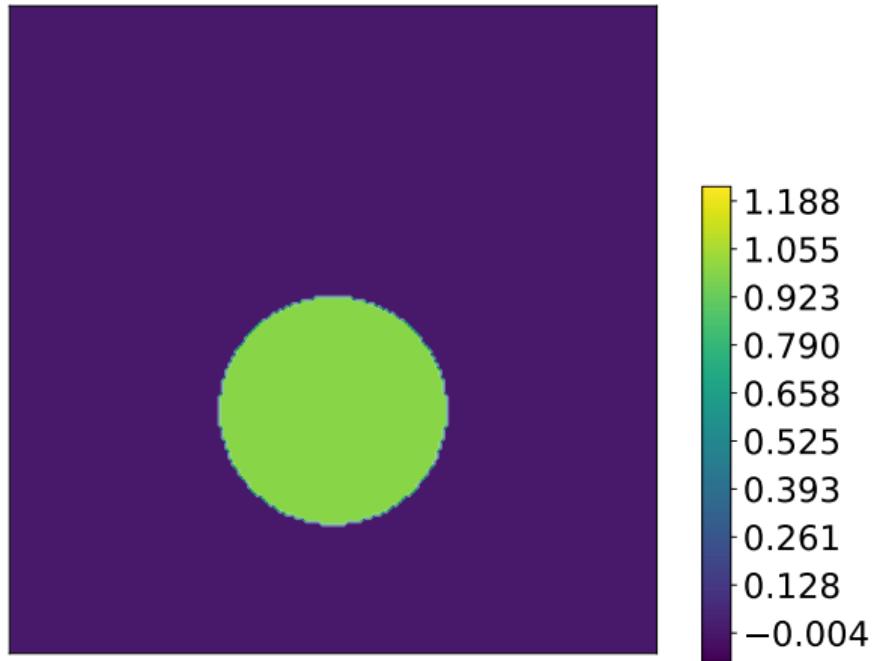
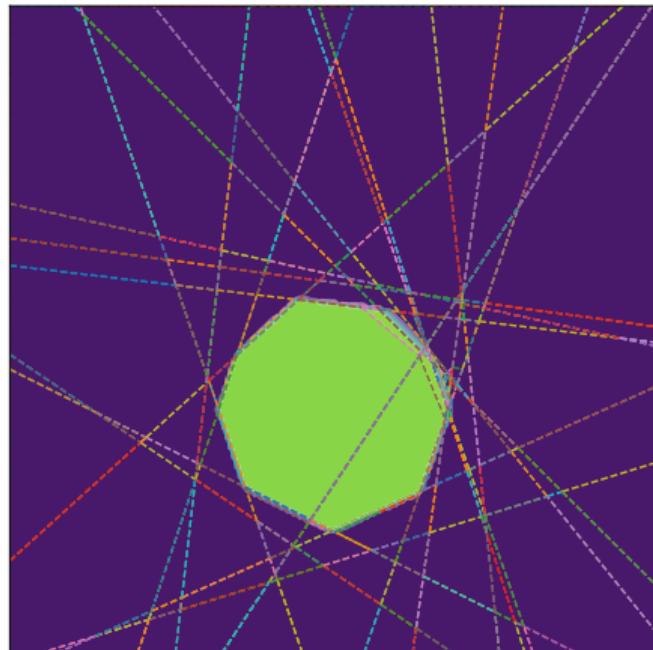
- Less sensitive to hyperparameters (to do the same with 1 hidden layer the different weights have to match exactly)
- **Fully interpretable**
- Easy 2D Hat function:  
$$(1-(x-y)^+-(y-x)^+-(x+y)^+--(-x-y)^+)^+$$



## Rediscovering shallow NN: ReLU 2 hidden layers in 2D

### 2 hidden layers NN

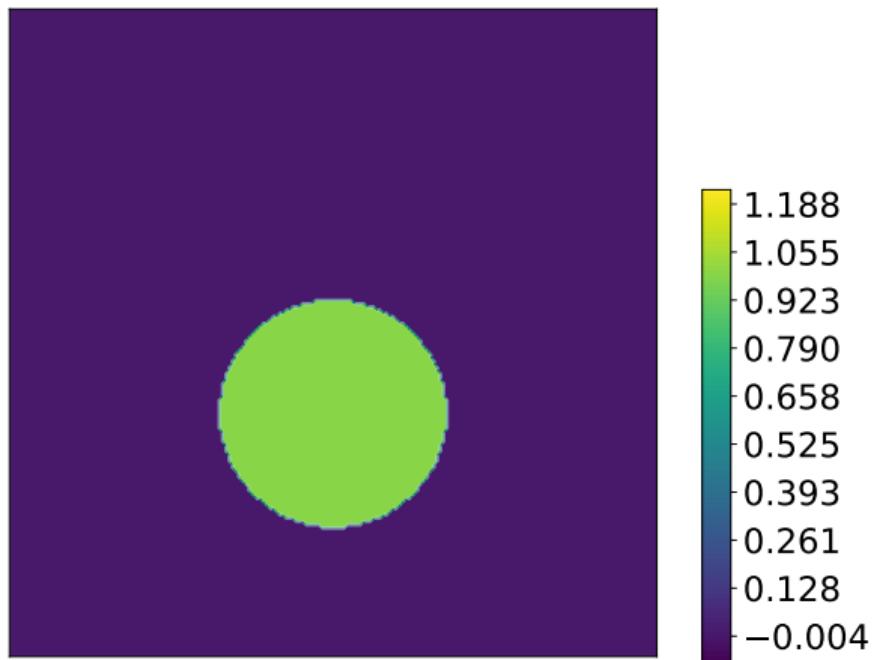
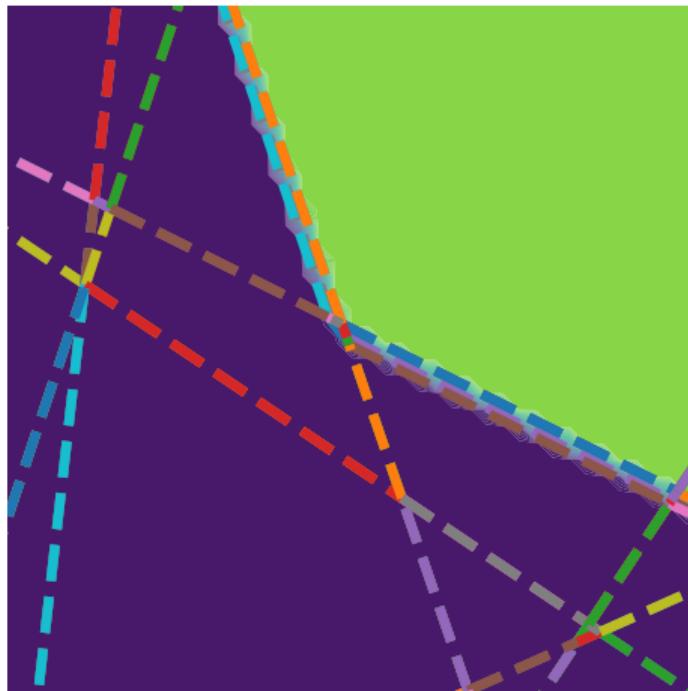
- More easily gets steep gradients
- Speed up the process of geometrical subdivision



## Rediscovering shallow NN: ReLU 2 hidden layers in 2D

### 2 hidden layers NN

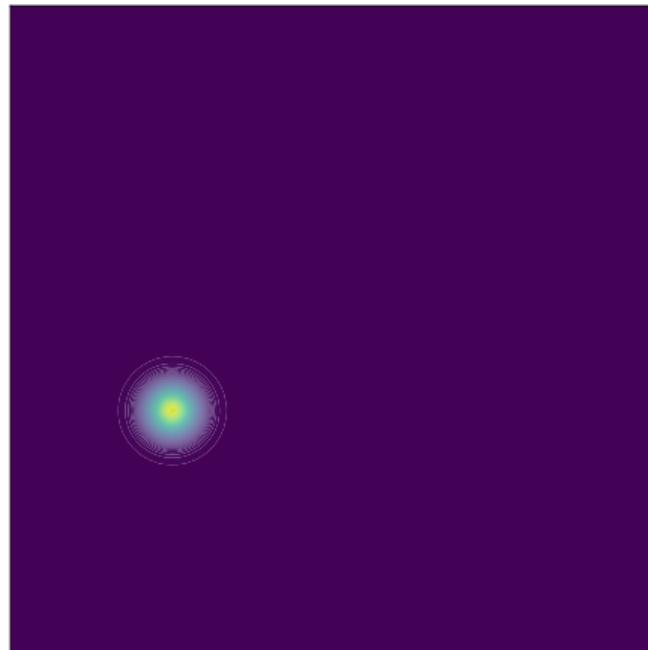
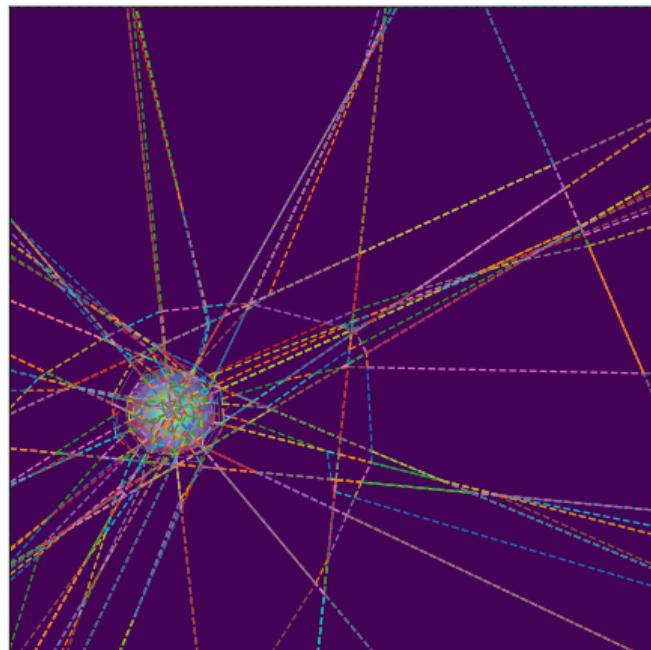
- More easily gets steep gradients
- Speed up the process of geometrical subdivision



## Rediscovering shallow NN: ReLU 2 hidden layers in 2D

### 2 hidden layers NN

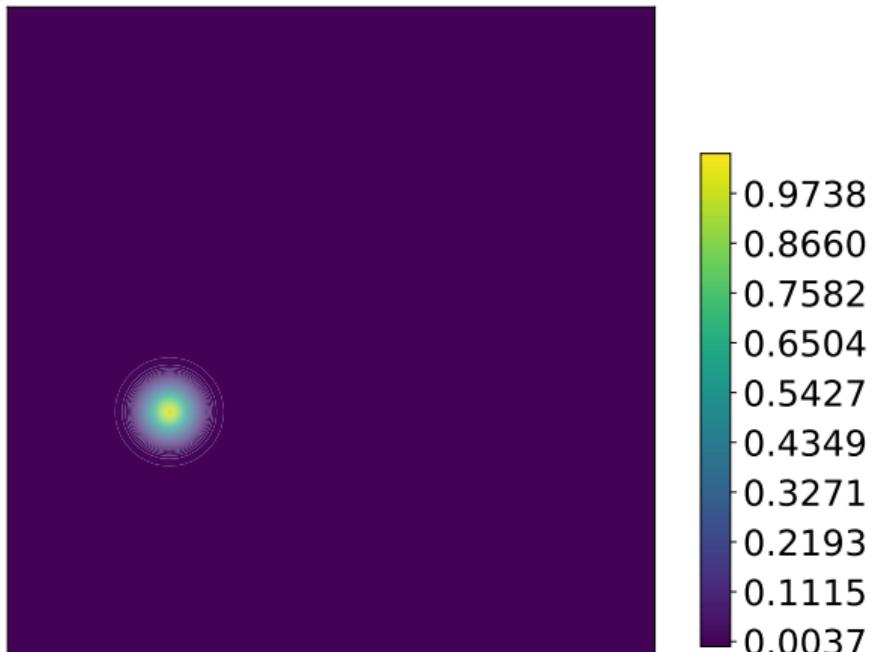
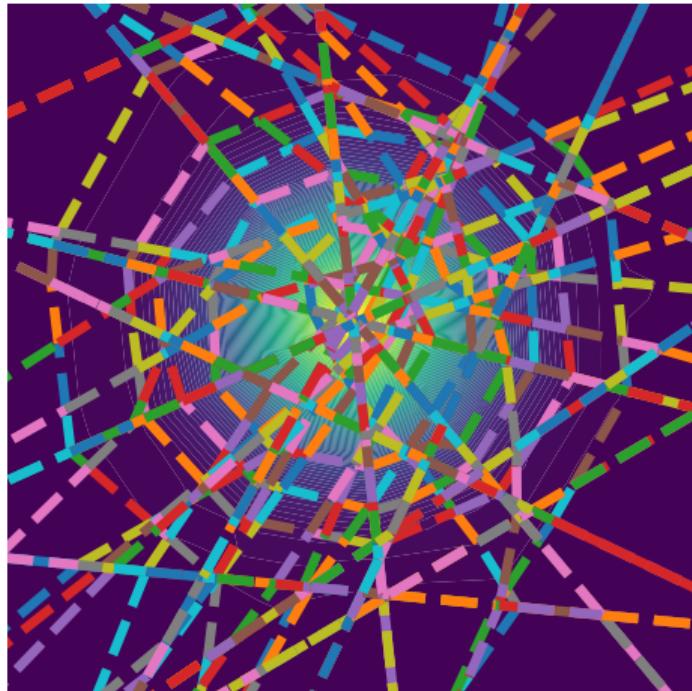
- More easily gets steep gradients
- Speed up the process of geometrical subdivision



## Rediscovering shallow NN: ReLU 2 hidden layers in 2D

### 2 hidden layers NN

- More easily gets steep gradients
- Speed up the process of geometrical subdivision



## What we have learned?

- 1-hidden-layer  $NN(x) = A^1(A^0x + b^0)^+ + b^1$   
**breaking lines**  $A_{i,:}^0x + b_i^0 = 0$  for all  $i$
- 2-hidden-layer  $NN(x) = A^2(A^1(A^0x + b^0)^+ + b^1)^+ + b^2$   
**possible** breaking lines  $A_{i,:}^0x + b_i^0 = 0 \forall i$  and

$$A_{i,:}^1(A^0x + b^0) + b_i^1 = 0 \quad \forall i$$

## Goals: Adaptive NN

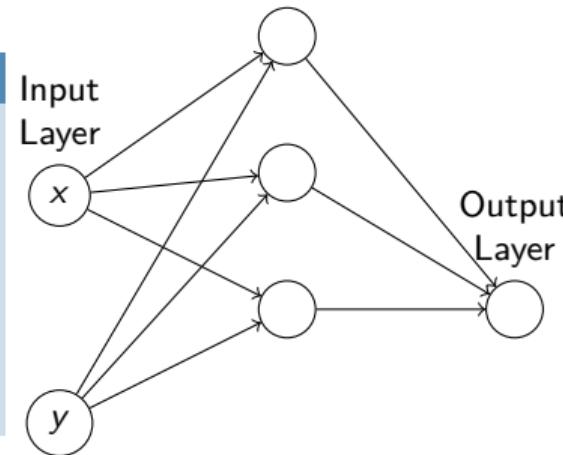
- Exploit **simple architectures** to save computational time in training!!
- Carefully selecting how **many** **breaking lines** and **where** we want to put them
- Copy ideas of **hp-adaptive** methods (h is now more nodes, p is now more layers (not really but more capability))

# Adaptive NN<sup>1</sup>

---

## Incremental architecture

- No *a priori* knowledge of how many layers/neurons are needed
- Increment nodes and layers until a tolerance
- Add neurons so that the new breaking line falls in the worst represented part
- Proceed with optimization process (Adam)



---

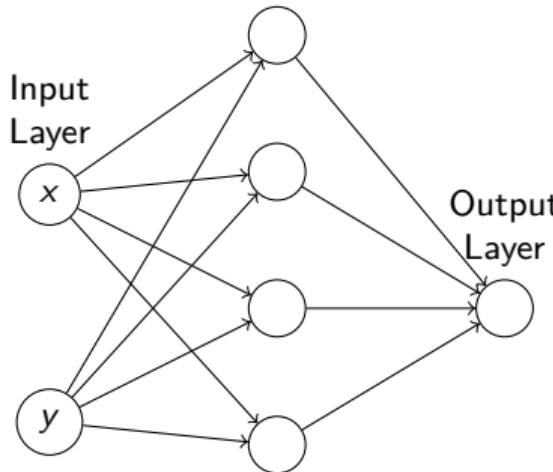
<sup>1</sup>Cai, Zhiqiang, Jingshuang Chen, and Min Liu. "Self-adaptive deep neural network: Numerical approximation to functions and PDEs." Journal of Computational Physics 455 (2022): 111021.

# Adaptive NN<sup>1</sup>

---

## Incremental architecture

- No *a priori* knowledge of how many layers/neurons are needed
- Increment nodes and layers until a tolerance
- Add neurons so that the new breaking line falls in the worst represented part
- Proceed with optimization process (Adam)



---

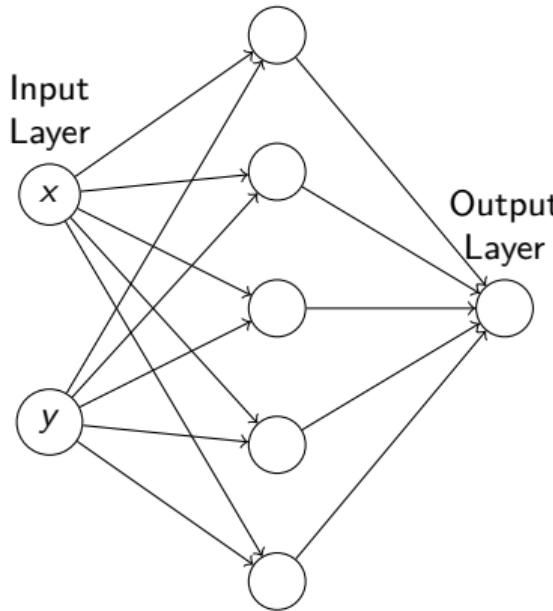
<sup>1</sup>Cai, Zhiqiang, Jingshuang Chen, and Min Liu. "Self-adaptive deep neural network: Numerical approximation to functions and PDEs." Journal of Computational Physics 455 (2022): 111021.

# Adaptive NN<sup>1</sup>

---

## Incremental architecture

- No *a priori* knowledge of how many layers/neurons are needed
- Increment nodes and layers until a tolerance
- Add neurons so that the new breaking line falls in the worst represented part
- Proceed with optimization process (Adam)



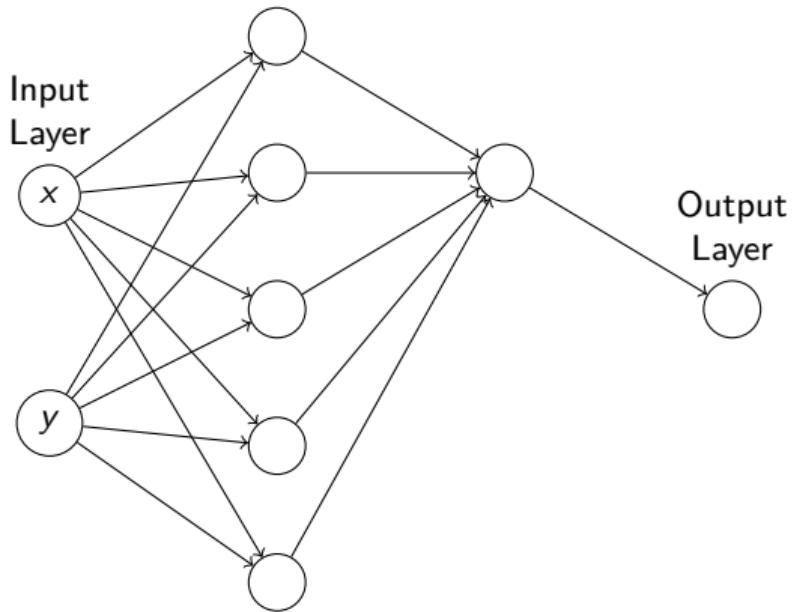
---

<sup>1</sup>Cai, Zhiqiang, Jingshuang Chen, and Min Liu. "Self-adaptive deep neural network: Numerical approximation to functions and PDEs." Journal of Computational Physics 455 (2022): 111021.

# Adaptive NN<sup>1</sup>

## Incremental architecture

- No *a priori* knowledge of how many layers/neurons are needed
- Increment nodes and layers until a tolerance
- Add neurons so that the new breaking line falls in the worst represented part
- Proceed with optimization process (Adam)

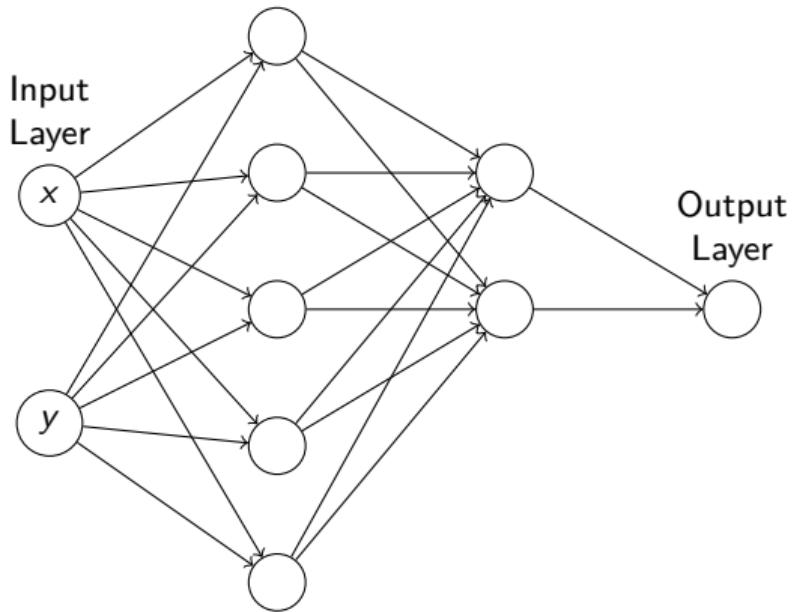


<sup>1</sup>Cai, Zhiqiang, Jingshuang Chen, and Min Liu. "Self-adaptive deep neural network: Numerical approximation to functions and PDEs." Journal of Computational Physics 455 (2022): 111021.

# Adaptive NN<sup>1</sup>

## Incremental architecture

- No *a priori* knowledge of how many layers/neurons are needed
- Increment nodes and layers until a tolerance
- Add neurons so that the new breaking line falls in the worst represented part
- Proceed with optimization process (Adam)

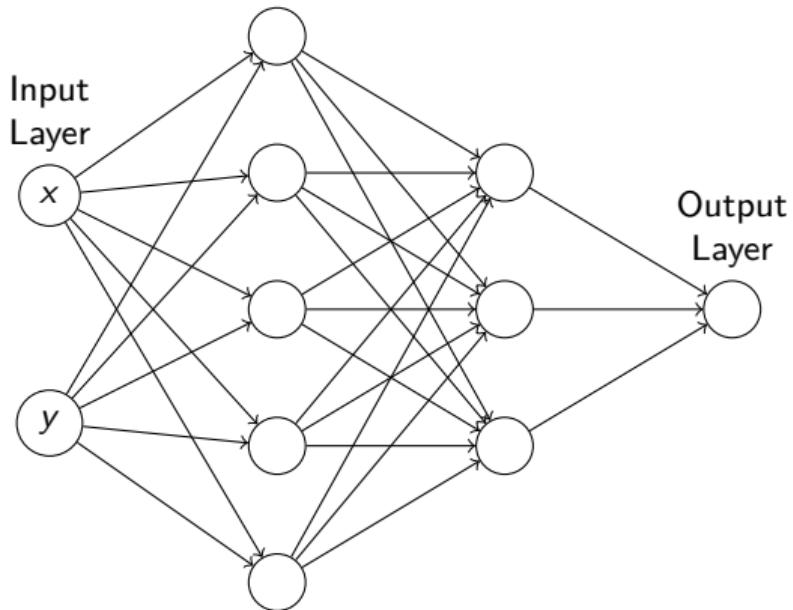


<sup>1</sup>Cai, Zhiqiang, Jingshuang Chen, and Min Liu. "Self-adaptive deep neural network: Numerical approximation to functions and PDEs." Journal of Computational Physics 455 (2022): 111021.

# Adaptive NN<sup>1</sup>

## Incremental architecture

- No *a priori* knowledge of how many layers/neurons are needed
- Increment nodes and layers until a tolerance
- Add neurons so that the new breaking line falls in the worst represented part
- Proceed with optimization process (Adam)

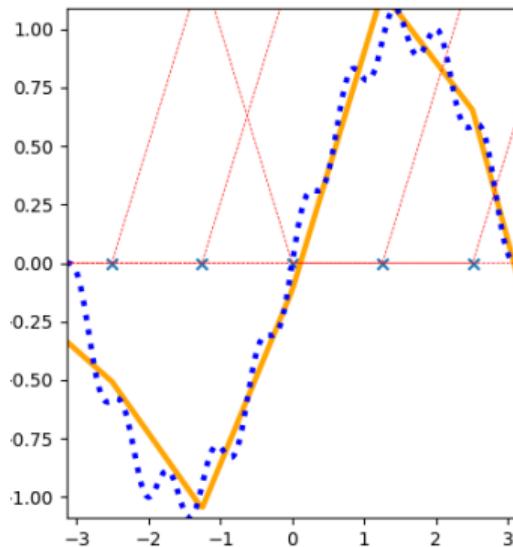


<sup>1</sup>Cai, Zhiqiang, Jingshuang Chen, and Min Liu. "Self-adaptive deep neural network: Numerical approximation to functions and PDEs." Journal of Computational Physics 455 (2022): 111021.

## Example of adaptive NN

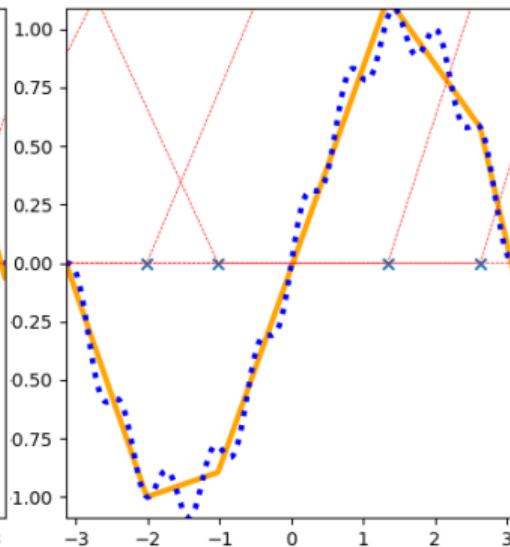
Before Training

Ref0

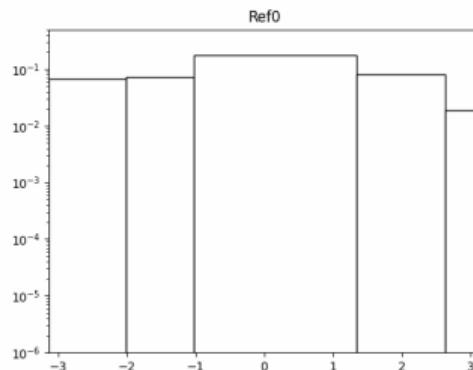


After Training

Ref0



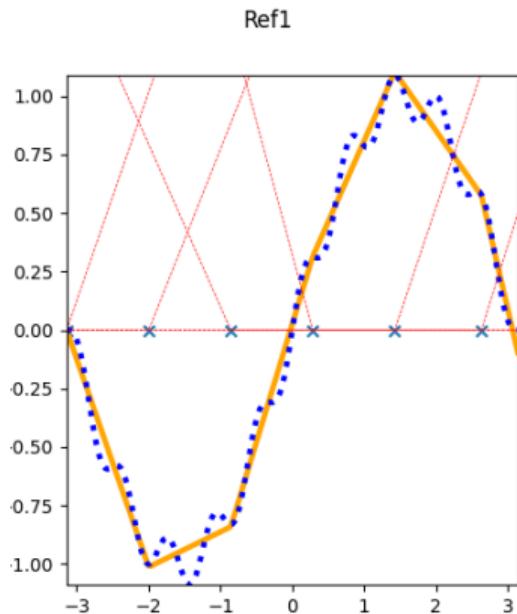
Error In Cells



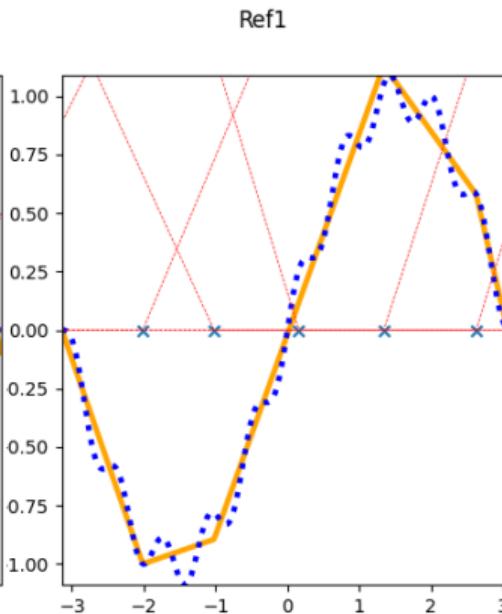
## Example of adaptive NN

---

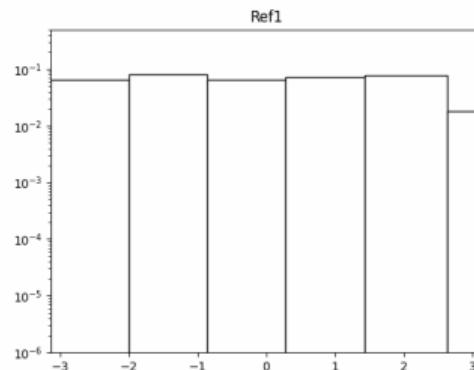
After Training



Before Training



Error In Cells

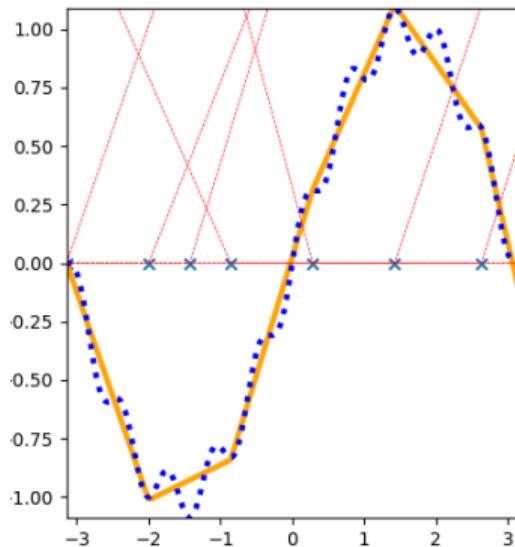


## Example of adaptive NN

---

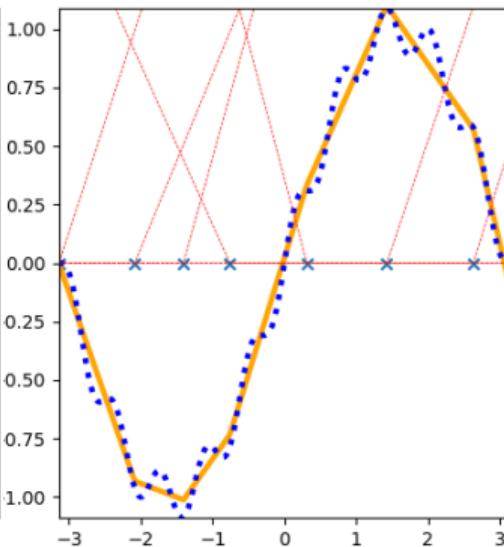
Before Training

Ref2

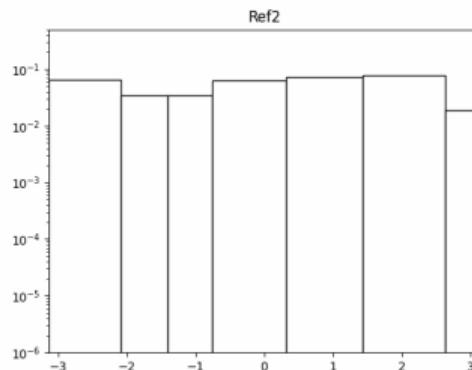


After Training

Ref2



Error In Cells

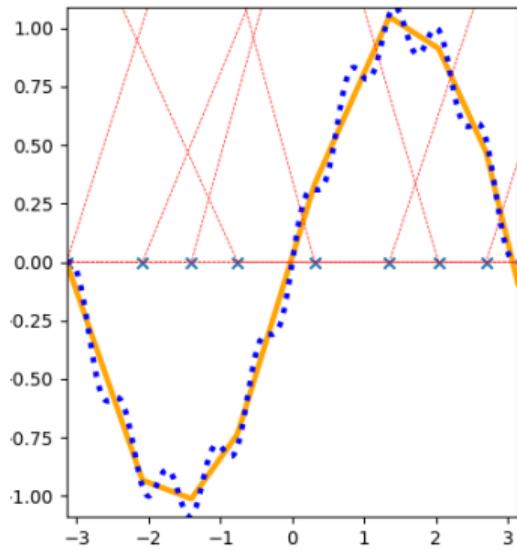


## Example of adaptive NN

---

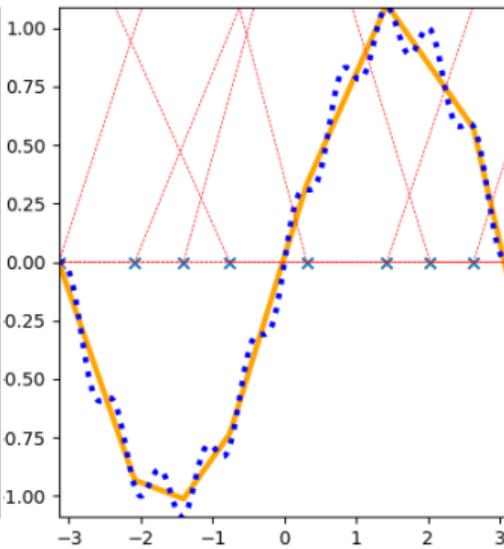
After Training

Ref3

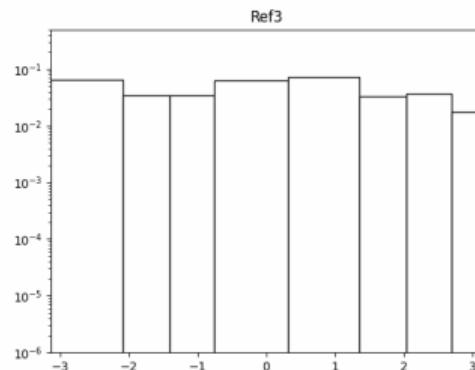


Before Training

Ref3



Error In Cells

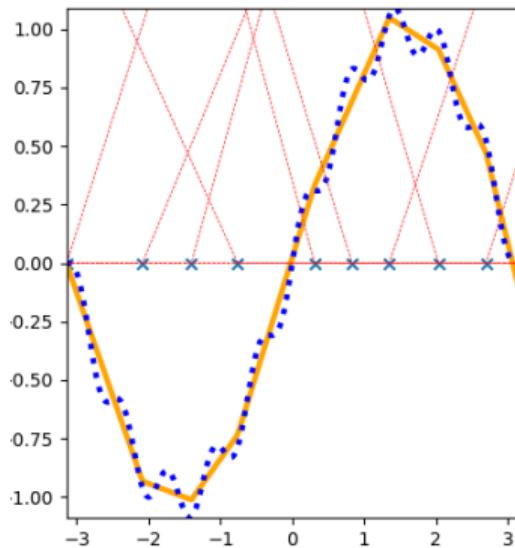


## Example of adaptive NN

---

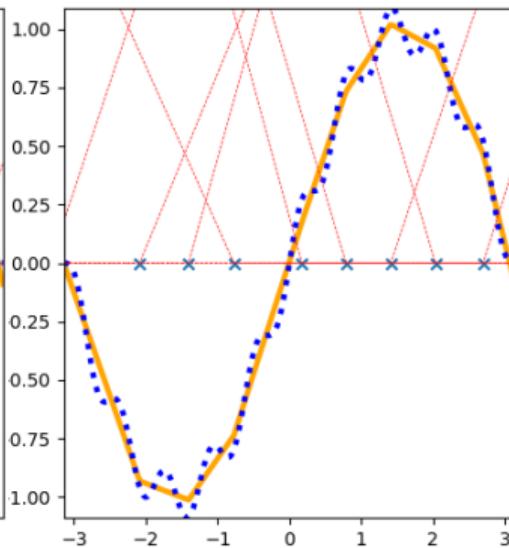
Before Training

Ref4

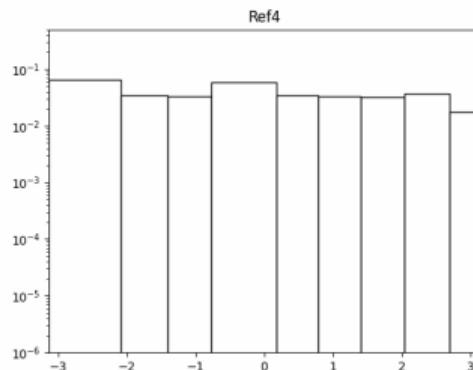


After Training

Ref4



Error In Cells

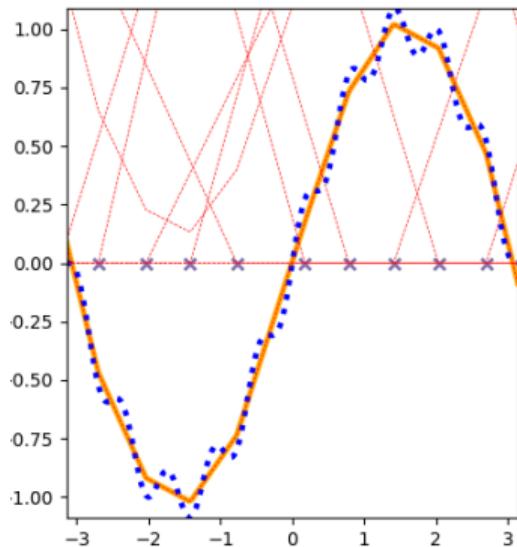


## Example of adaptive NN

---

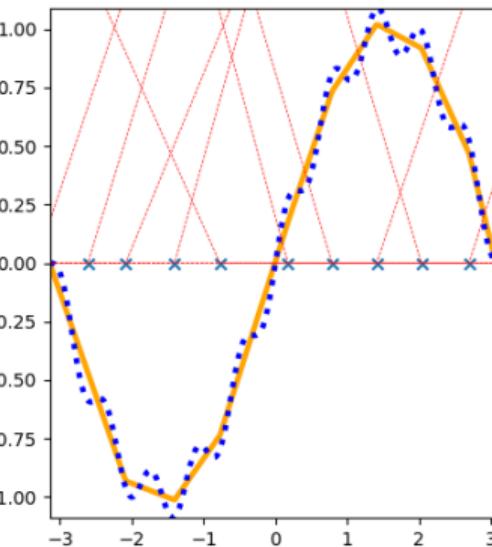
After Training

Ref5

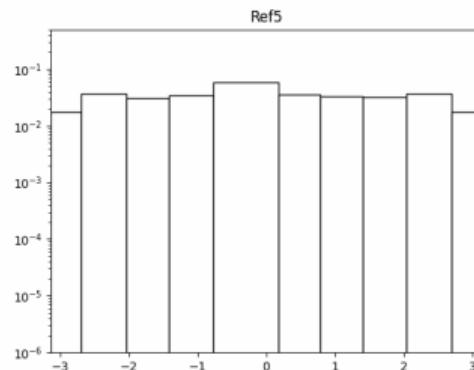


Before Training

Ref5



Error In Cells

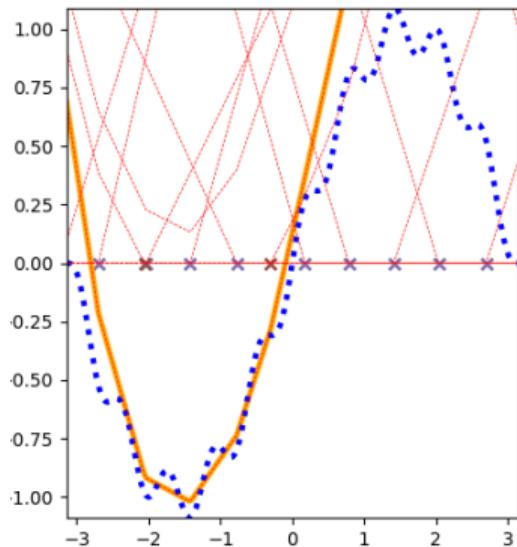


## Example of adaptive NN

---

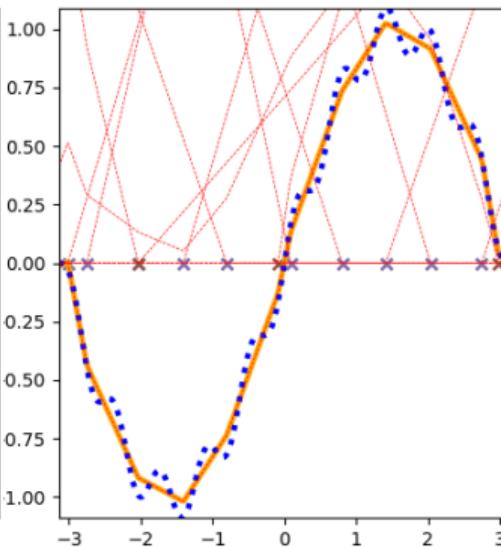
Before Training

Ref6

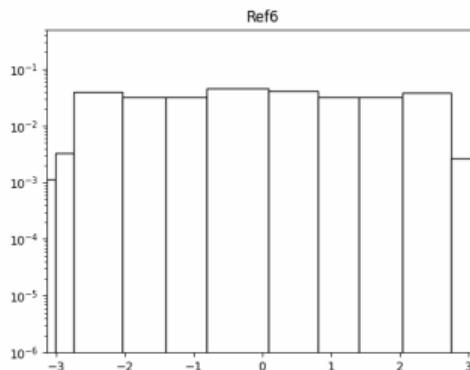


After Training

Ref6



Error In Cells

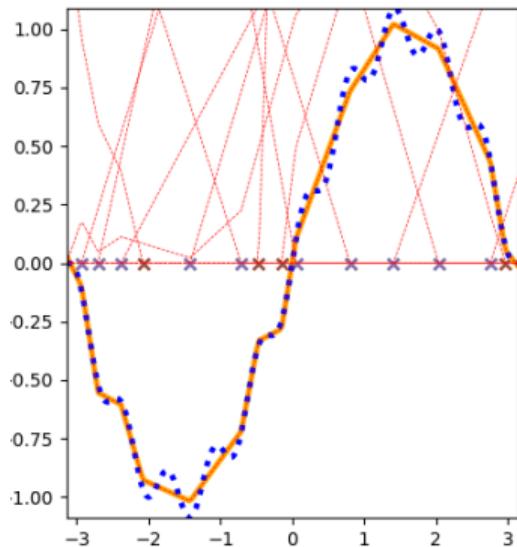


## Example of adaptive NN

---

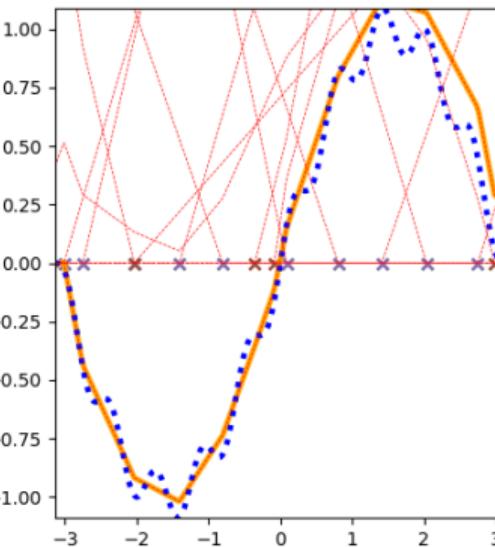
After Training

Ref7

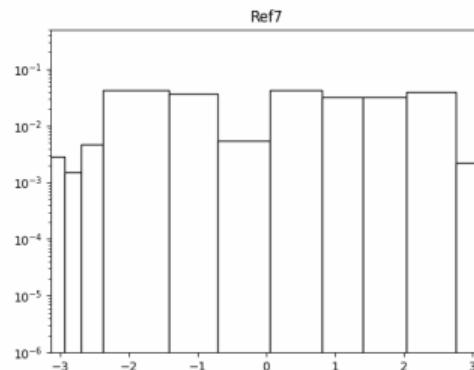


Before Training

Ref7



Error In Cells

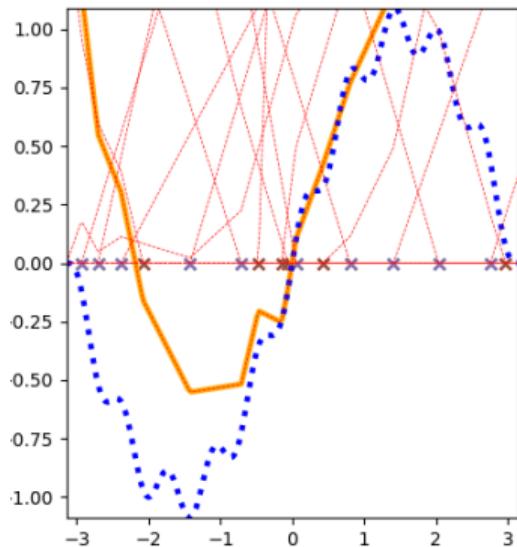


## Example of adaptive NN

---

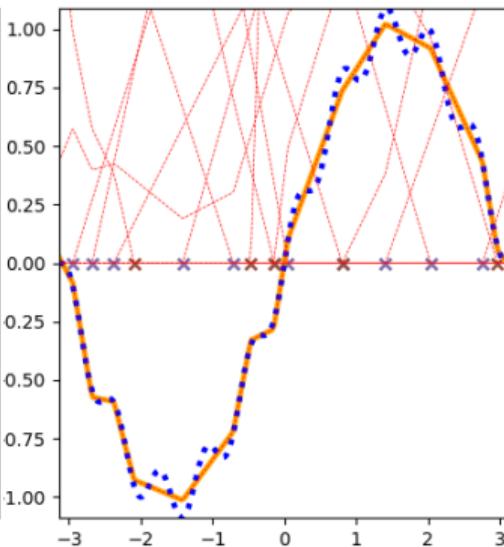
Before Training

Ref8

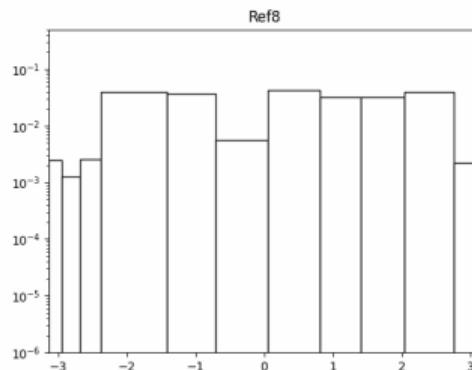


After Training

Ref8



Error In Cells

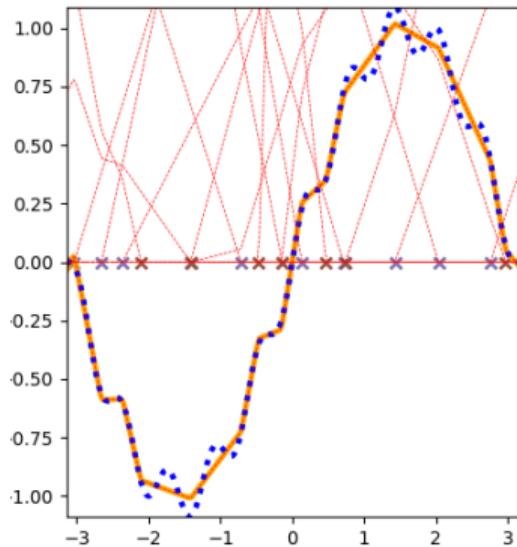


## Example of adaptive NN

---

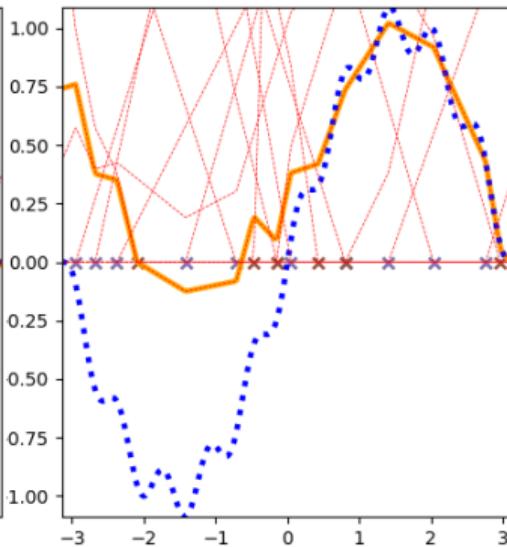
After Training

Ref9

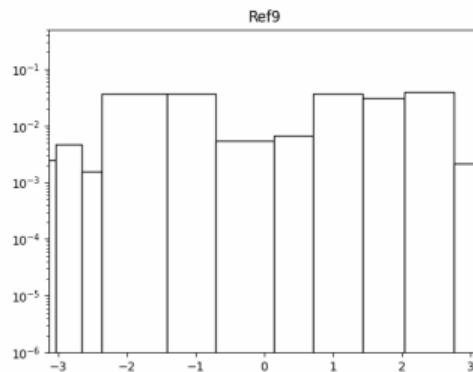


Before Training

Ref9



Error In Cells

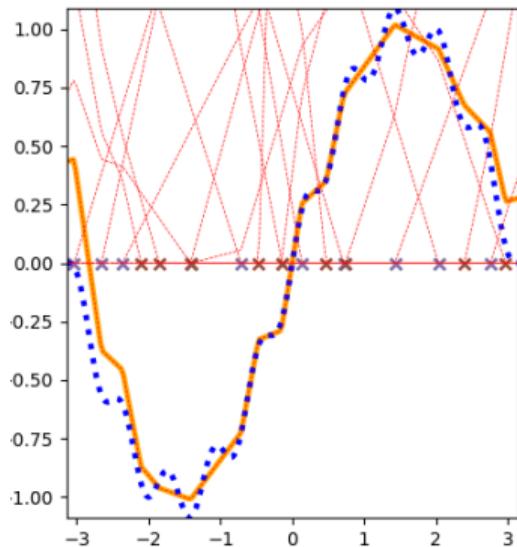


## Example of adaptive NN

---

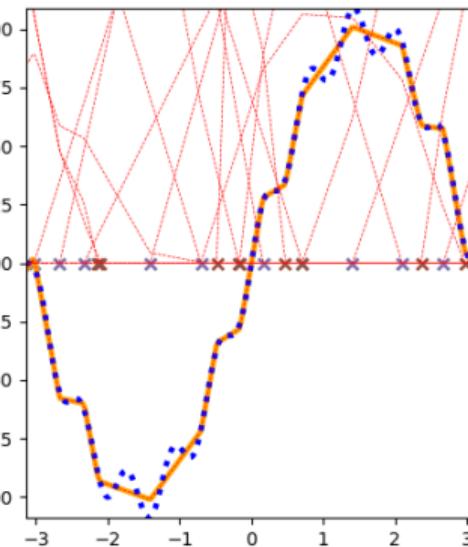
Before Training

Ref10

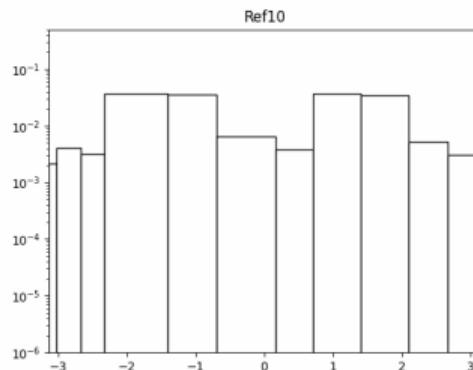


After Training

Ref10



Error In Cells

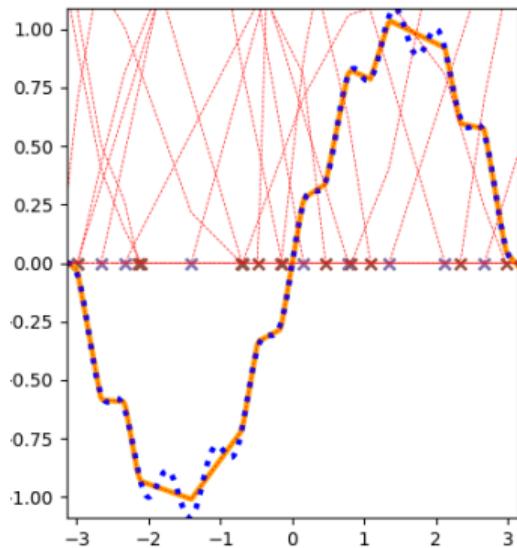


## Example of adaptive NN

---

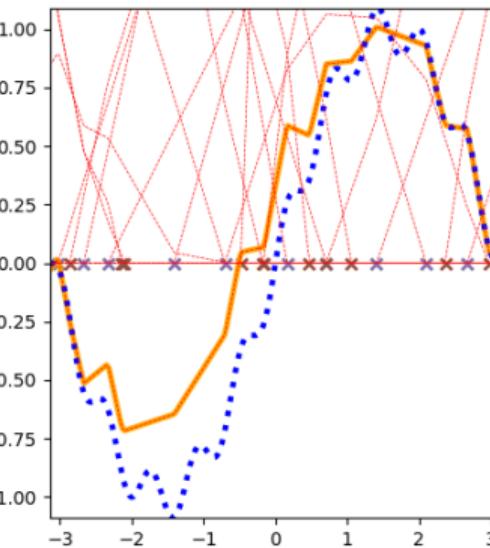
After Training

Ref11

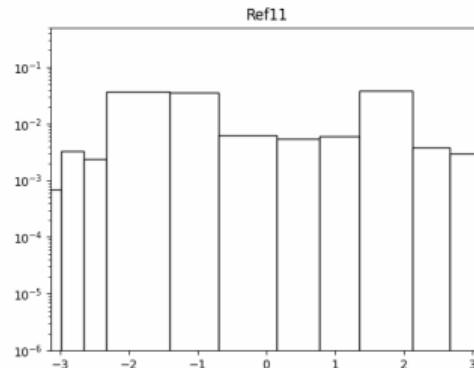


Before Training

Ref11



Error In Cells

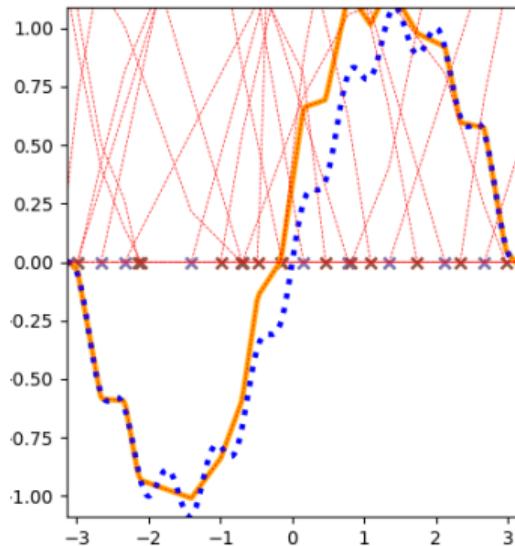


## Example of adaptive NN

---

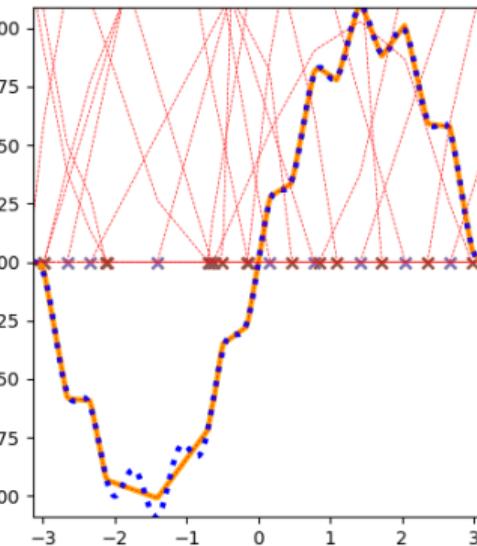
Before Training

Ref12

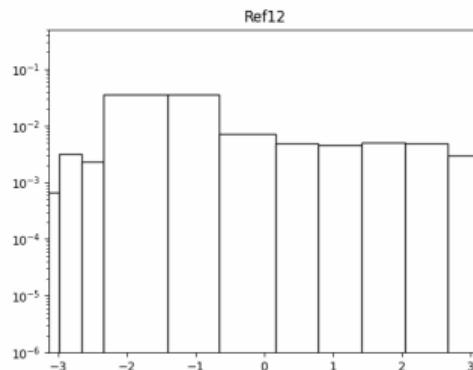


After Training

Ref12



Error In Cells

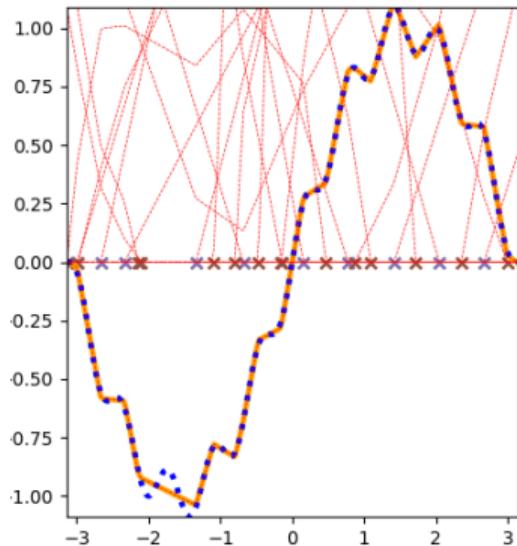


## Example of adaptive NN

---

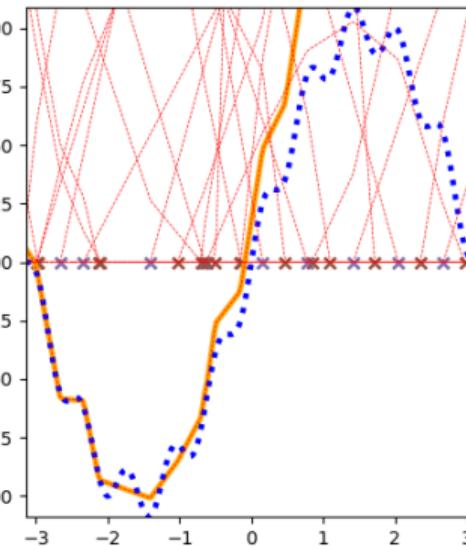
After Training

Ref13

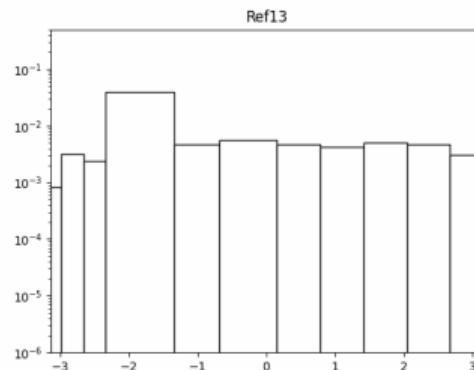


Before Training

Ref13



Error In Cells

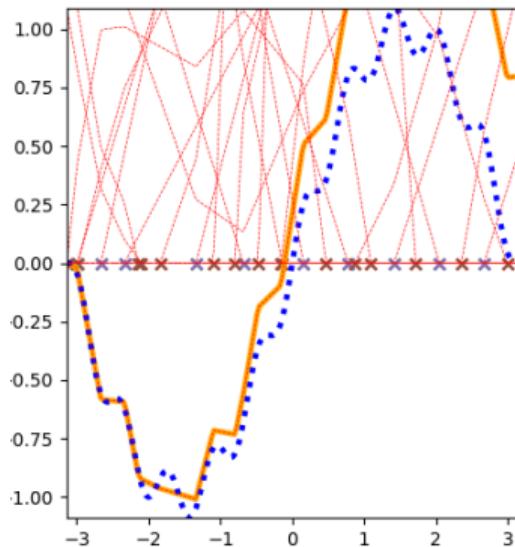


## Example of adaptive NN

---

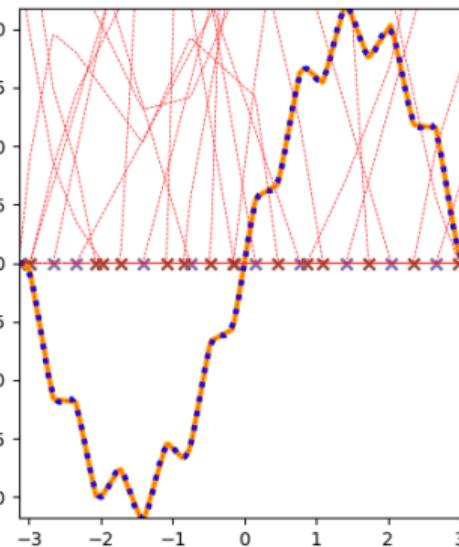
Before Training

Ref14

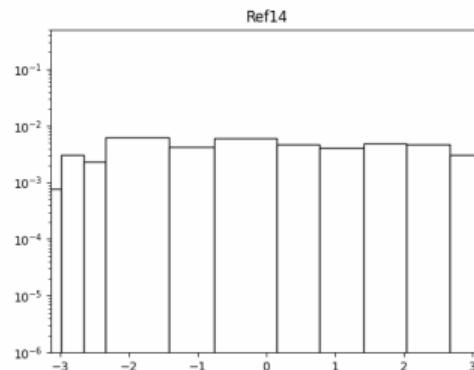


After Training

Ref14



Error In Cells



## Adaptive NN details

### Questions

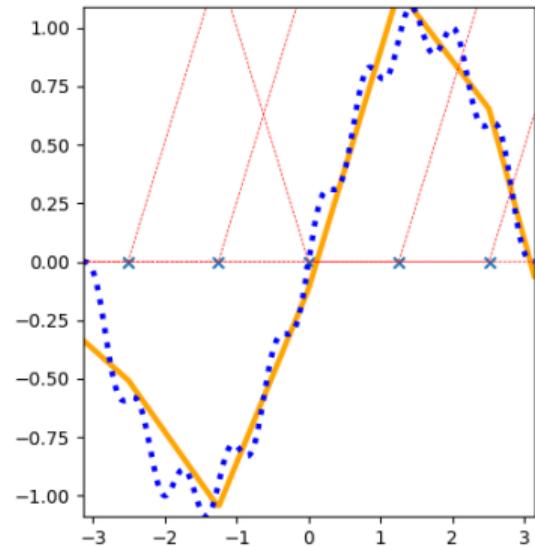
- How do we initialize all weights and biases at the beginning?

Answers: initialization at ref 0

$$NN(x) = A^1(A^0x + b^0)^+ + b^1$$

- $A^0, b^0$  such that  $A_i^0 x_i + b_i^0 = 0$  with  $x_i$  equispaced
- $|A_i^0| = 1$  for all  $i$  random sign
- Outer layer: Least Square

$$\min_{A^1, b^1} \sum_j |NN(x_j) - y_j|^2$$



## Adaptive NN details

### Questions

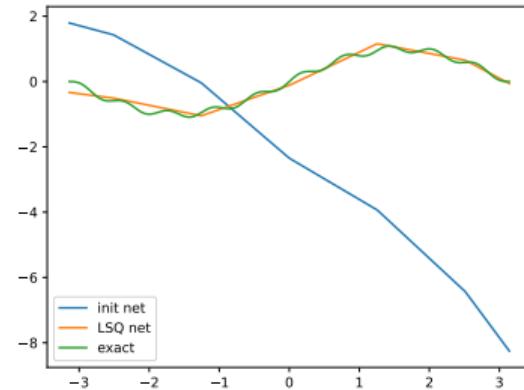
- How do we initialize all weights and biases at the beginning?

### Answers: initialization at ref 0

$$NN(x) = A^1(A^0x + b^0)^+ + b^1$$

- $A^0, b^0$  such that  $A_i^0 x_i + b_i^0 = 0$  with  $x_i$  equispaced
- $|A_i^0| = 1$  for all  $i$  random sign
- Outer layer: Least Square

$$\min_{A^1, b^1} \sum_j |NN(x_j) - y_j|^2$$



## Adaptive NN details: first hidden layer

### Questions

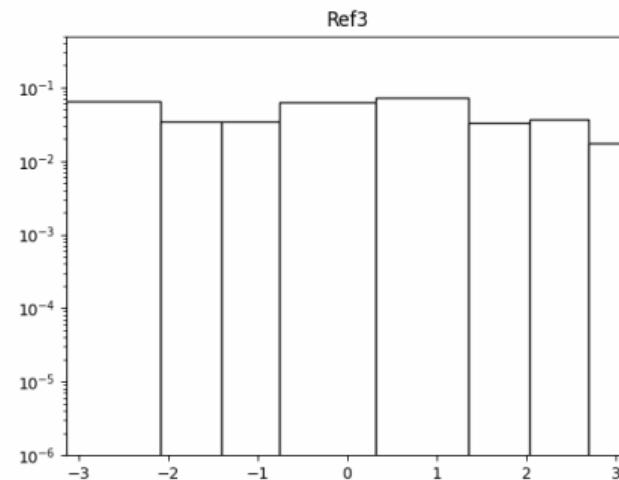
- How do we initialize the new weights and biases?
- When do we decide to add a layer or a node and how?

### Answers: Additional **node** in first layer

- If Loss is not decreasing more than 4% in the last 5000 epochs: add new node
- Cell-wise error: divide the domain with breaking lines and search for the highest error
- Add new node so that the breaking hyperplane cuts the barycenter of this cell

$$A_{\ell^1+1}^0 x_{bary} + b_{\ell^1+1}^0 = 0$$

- Outer layer:  $A_{\ell^1+1}^1 = 0$



## Adaptive NN details: first hidden layer

### Questions

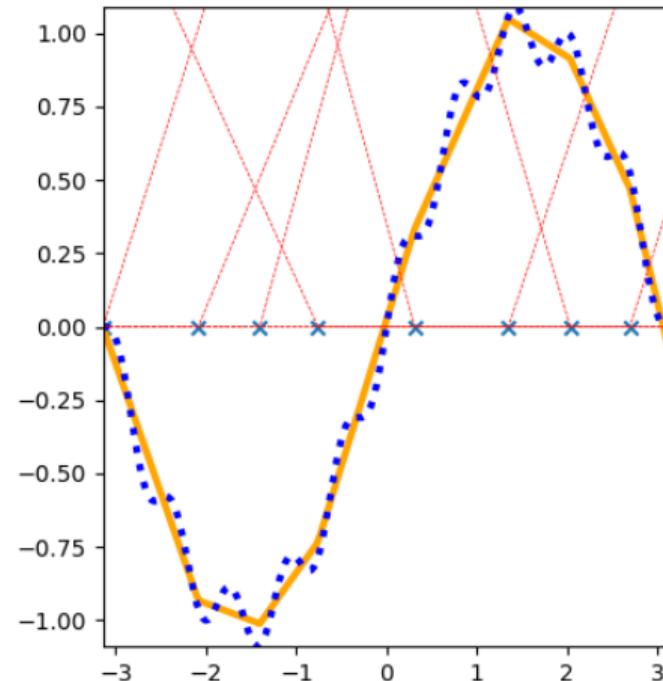
- How do we initialize the new weights and biases?
- When do we decide to add a layer or a node and how?

### Answers: Additional node in first layer

- If Loss is not decreasing more than 4% in the last 5000 epochs: add new node
- Cell-wise error: divide the domain with breaking lines and search for the highest error
- Add new node so that the breaking hyperplane cuts the barycenter of this cell

$$A_{\ell^1+1}^0 x_{bary} + b_{\ell^1+1}^0 = 0$$

- Outer layer:  $A_{\ell^1+1}^1 = 0$



## Adaptive NN details: first hidden layer

### Questions

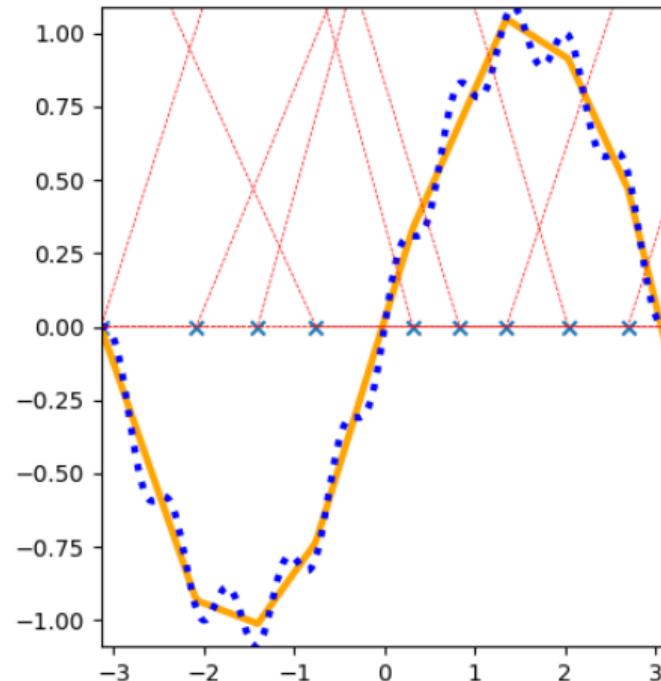
- How do we initialize the new weights and biases?
- When do we decide to add a layer or a node and how?

### Answers: Additional node in first layer

- If Loss is not decreasing more than 4% in the last 5000 epochs: add new node
- Cell-wise error: divide the domain with breaking lines and search for the highest error
- Add new node so that the breaking hyperplane cuts the barycenter of this cell

$$A_{\ell^1+1}^0 x_{bary} + b_{\ell^1+1}^0 = 0$$

- Outer layer:  $A_{\ell^1+1}^1 = 0$



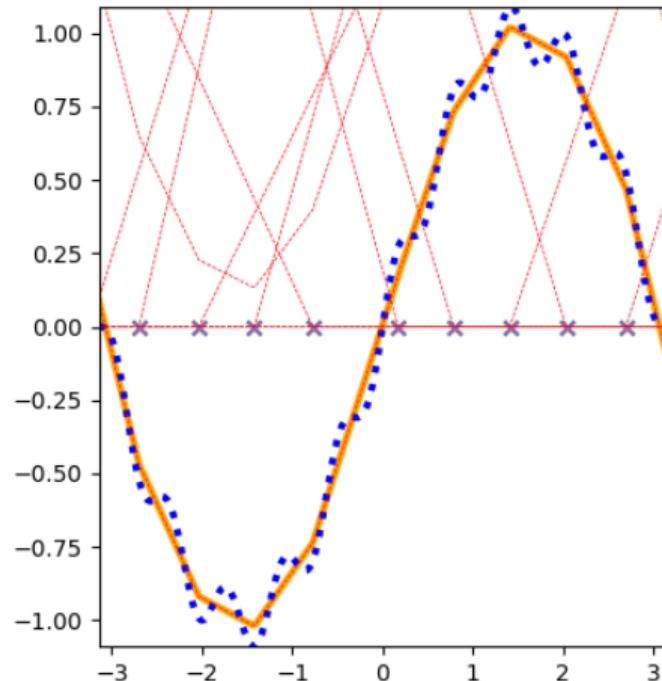
## Adaptive NN details: second hidden layer

### Questions

- How do we initialize the new weights and biases?
- When do we decide to add a layer or a node and how?

### Answers: Additional **layer** or **node** in second layer

- If Loss is not decreasing more than 20% in the last 7 new nodes: add second layer
- Cell-wise error: divide the domain with breaking lines and search for the highest error
- Add new layer:  $A_1^2 = 1$  and  $b^2 = -\min_x((A^1(A^0x + b^0)^+ + b^1)^+)$  (no influence)
- Add new node in second layer: more complicated, add one breaking line in one place, but more unpredictable effects



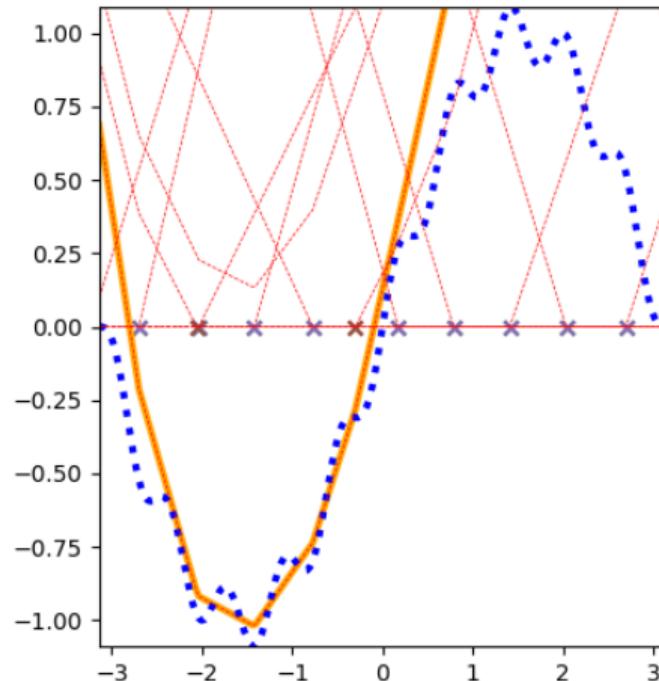
## Adaptive NN details: second hidden layer

### Questions

- How do we initialize the new weights and biases?
- When do we decide to add a layer or a node and how?

### Answers: Additional **layer** or **node** in second layer

- If Loss is not decreasing more than 20% in the last 7 new nodes: add second layer
- Cell-wise error: divide the domain with breaking lines and search for the highest error
- Add new layer:  $A_1^2 = 1$  and  $b^2 = -\min_x((A^1(A^0x + b^0)^+ + b^1)^+)$  (no influence)
- Add new node in second layer: more complicated, add one breaking line in one place, but more unpredictable effects



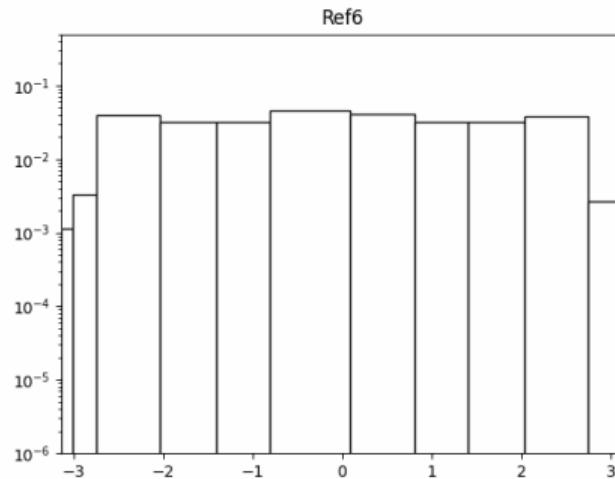
## Adaptive NN details: second hidden layer

### Questions

- How do we initialize the new weights and biases?
- When do we decide to add a layer or a node and how?

### Answers: Additional layer or node in second layer

- If Loss is not decreasing more than 20% in the last 7 new nodes: add second layer
- Cell-wise error: divide the domain with breaking lines and search for the highest error
- Add new layer:  $A_1^2 = 1$  and  $b^2 = -\min_x((A^1(A^0x + b^0)^+ + b^1)^+)$  (no influence)
- Add new node in second layer: more complicated, add one breaking line in one place, but more unpredictable effects



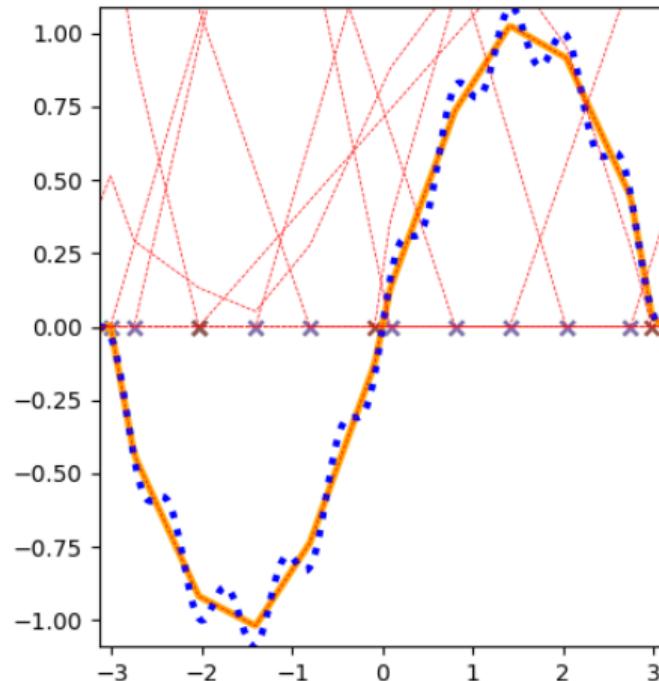
## Adaptive NN details: second hidden layer

### Questions

- How do we initialize the new weights and biases?
- When do we decide to add a layer or a node and how?

### Answers: Additional layer or node in second layer

- If Loss is not decreasing more than 20% in the last 7 new nodes: add second layer
- Cell-wise error: divide the domain with breaking lines and search for the highest error
- Add new layer:  $A_1^2 = 1$  and  $b^2 = -\min_x((A^1(A^0x + b^0)^+ + b^1)^+)$  (no influence)
- Add new node in second layer: more complicated, add one breaking line in one place, but more unpredictable effects



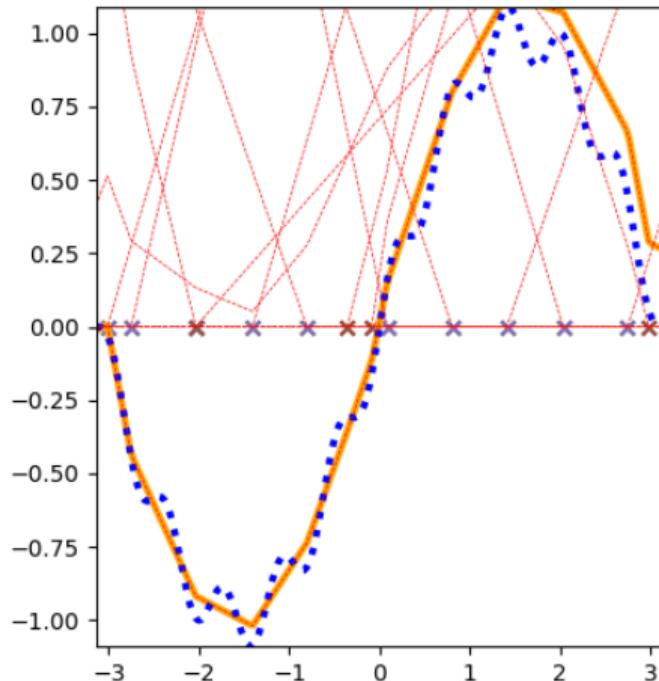
## Adaptive NN details: second hidden layer

### Questions

- How do we initialize the new weights and biases?
- When do we decide to add a layer or a node and how?

### Answers: Additional **layer** or **node** in second layer

- If Loss is not decreasing more than 20% in the last 7 new nodes: add second layer
- Cell-wise error: divide the domain with breaking lines and search for the highest error
- Add new layer:  $A_1^2 = 1$  and  $b^2 = -\min_x((A^1(A^0x + b^0)^+ + b^1)^+)$  (no influence)
- Add new node in second layer: more complicated, add one breaking line in one place, but more unpredictable effects



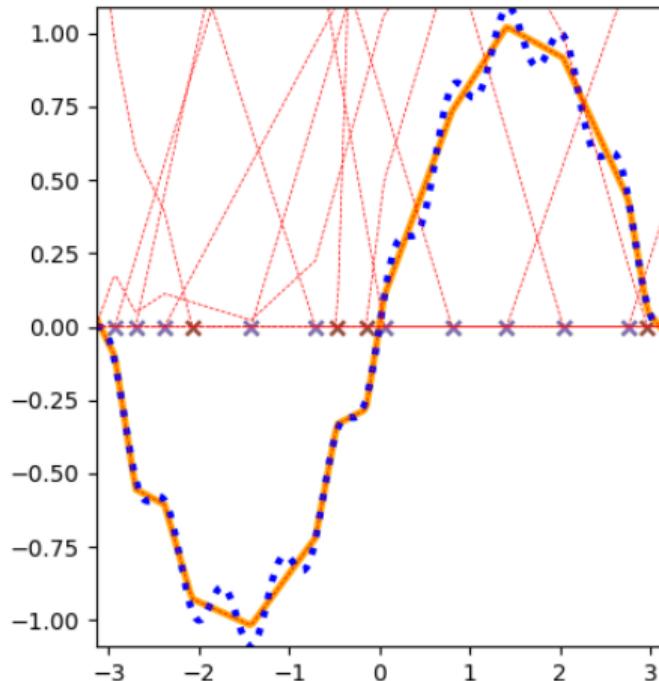
## Adaptive NN details: second hidden layer

### Questions

- How do we initialize the new weights and biases?
- When do we decide to add a layer or a node and how?

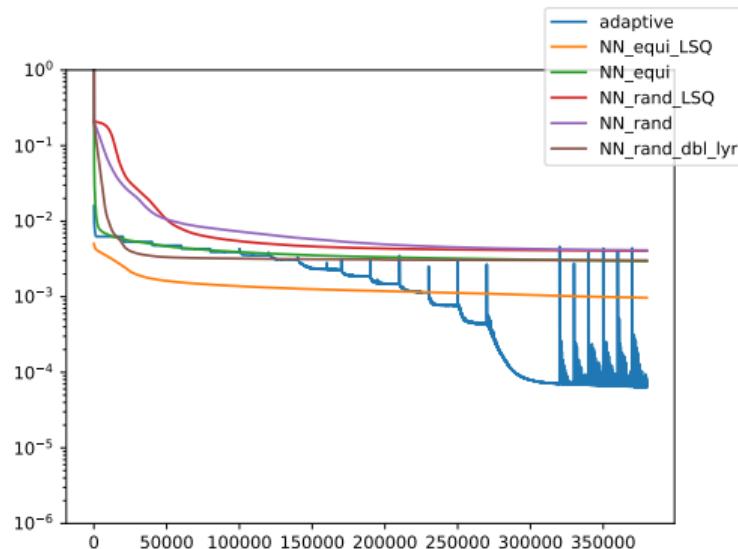
### Answers: Additional **layer** or **node** in second layer

- If Loss is not decreasing more than 20% in the last 7 new nodes: add second layer
- Cell-wise error: divide the domain with breaking lines and search for the highest error
- Add new layer:  $A_1^2 = 1$  and  $b^2 = -\min_x((A^1(A^0x + b^0)^+ + b^1)^+)$  (no influence)
- Add new node in second layer: more complicated, add one breaking line in one place, but more unpredictable effects

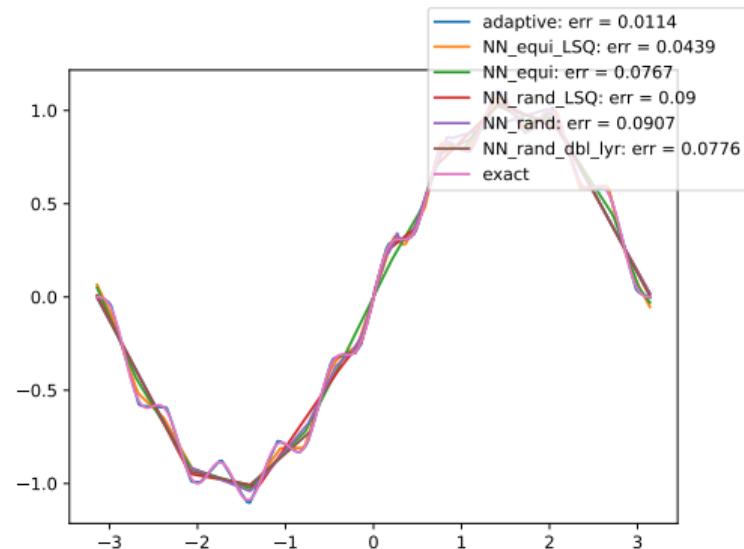


# Results for ANN: multisin 1D [1 9 16 1]

Loss decay

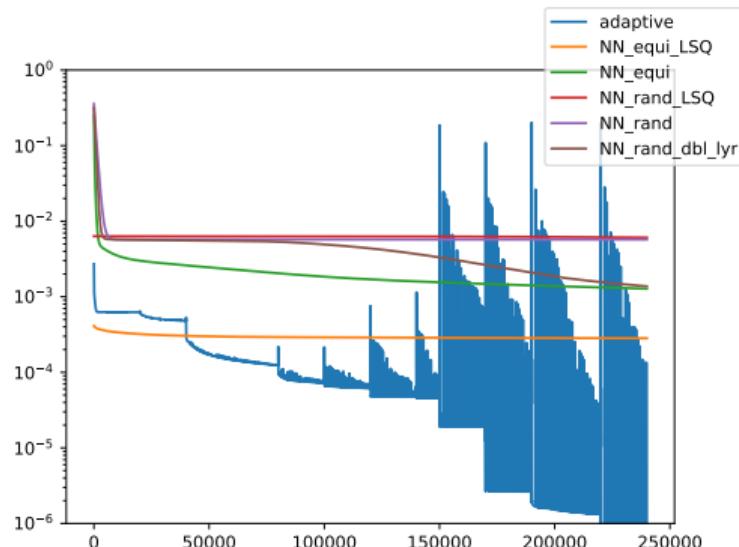


Approximation

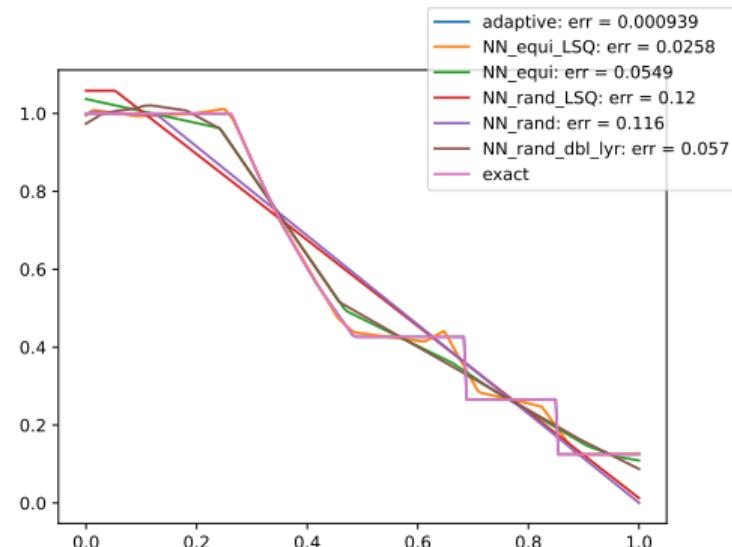


## Results for ANN: Sod 1D [1 15 1]

Loss decay



Approximation



# Table of contents

---

① Adaptive NN

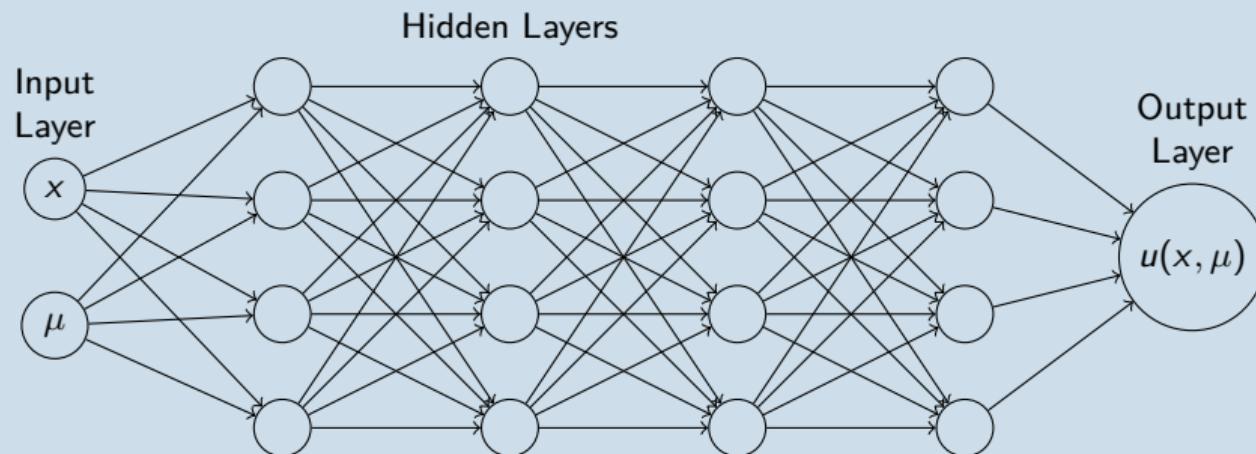
② Adaptive NN for Reduced Order Modeling  
Simulations

③ Conclusions

## Parametrized problem

- We are looking for the solution  $u$  of  $\mathcal{P}(u(\mu); \mu) = 0$  for some  $\mu \in M$
- We can afford computational costs for few  $u(\mu_i)$  with  $\{\mu_i\}_{i=1}^{n_{train}}$
- We want to forecast  $u(\mu)$  for  $\mu \notin \{\mu_i\}_{i=1}^{n_{train}}$

## NN to learn functions example



# Adaptive NN for Reduced Order Modeling architectures

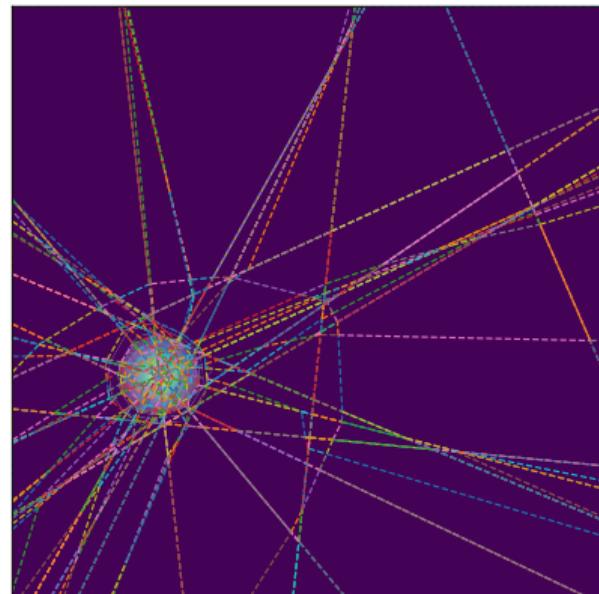
## Adaptive NN and domain dimension

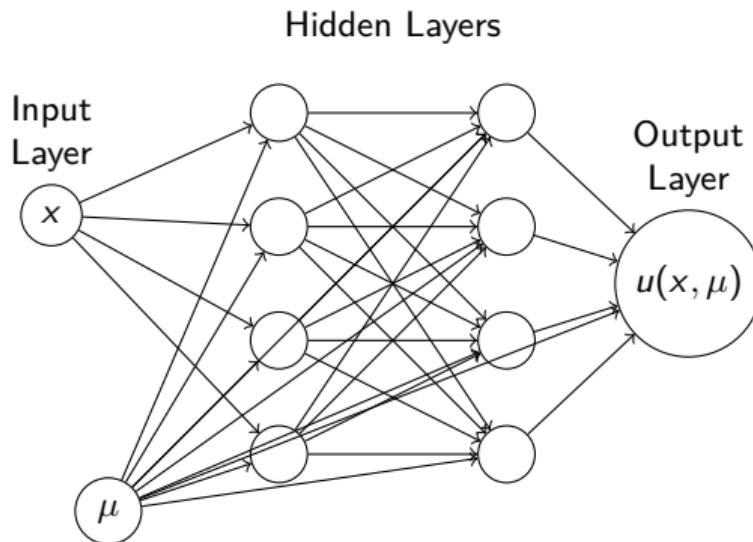
- Error estimator based on domain subdivision from breaking lines
- Defining the breaking hyperplanes is increasingly complex with dimensions (and layers)
- Bad idea considering parameter as extra dimension (curse of dimensionality)
- Code: only 2D

## Solution

- Consider biases that are parameter dependent  
 $b^i = b_0^i \mu + b_1^i$
- Fix a reference parameter  $\mu^*$  where to apply the error estimator

Network [2 14 22 1]





## Training Strategy

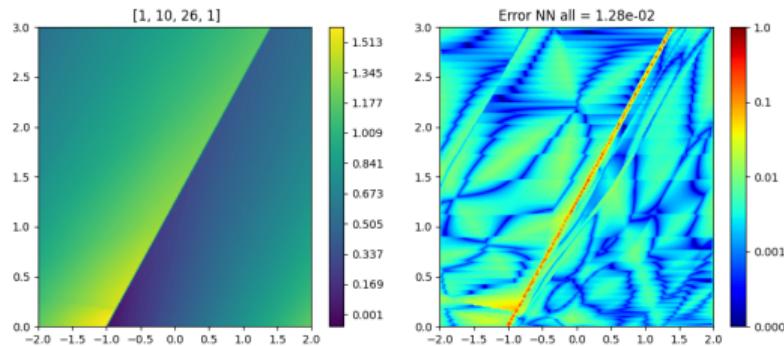
- Option **ALL**: train **from zero** with this architecture
- Option **FNP**: train first a non parametric for a  $\mu^*$ , start from that architecture and train again with  $b^i = b_0^i(\mu - \mu^*) + b_1^i$  initializing  $b_0^i = 0$  and  $b_1^i = b_{NP}^i$
- Option **1D-as-2D**: If in 1D, and 1 param, we can pretend it is 2D and train as in 2D

## Training Elements

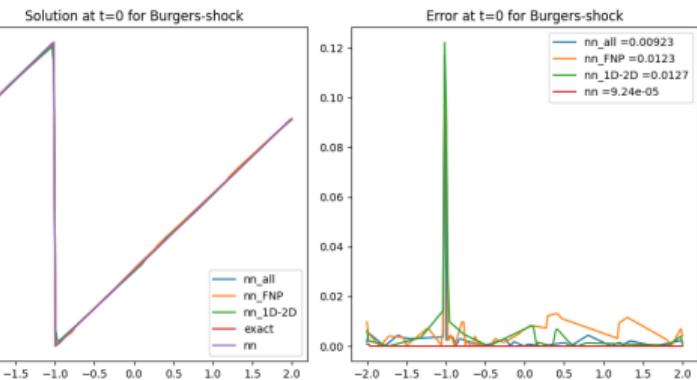
- Training set: some snapshots  $\{u(\mu_i)\}_{i=1}^{n_{train}}$
- Adaptive procedure on error set on geometry at  $\mu^*$

# Burgers shock

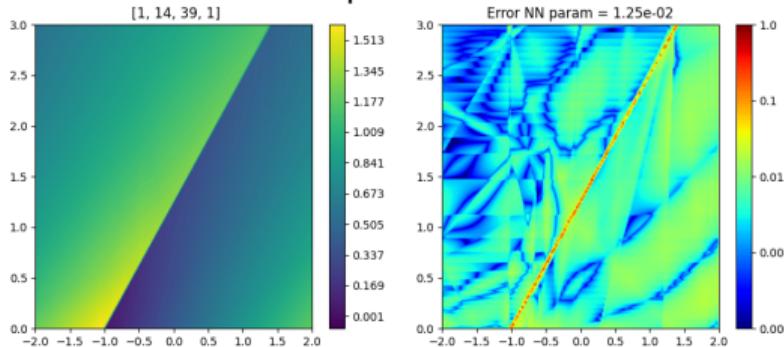
Adaptive ALL



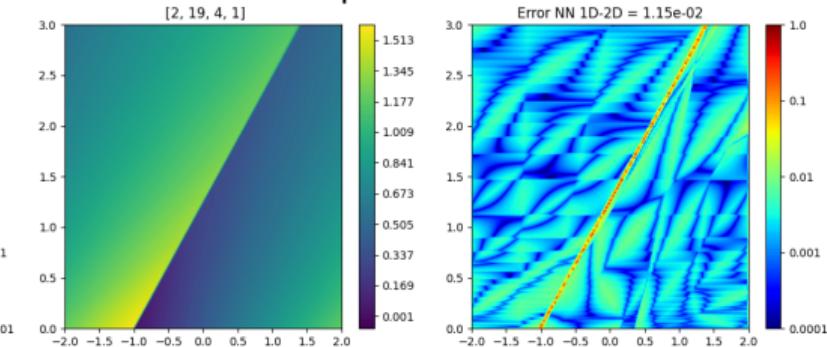
Comparison at initial time



Adaptive FNP

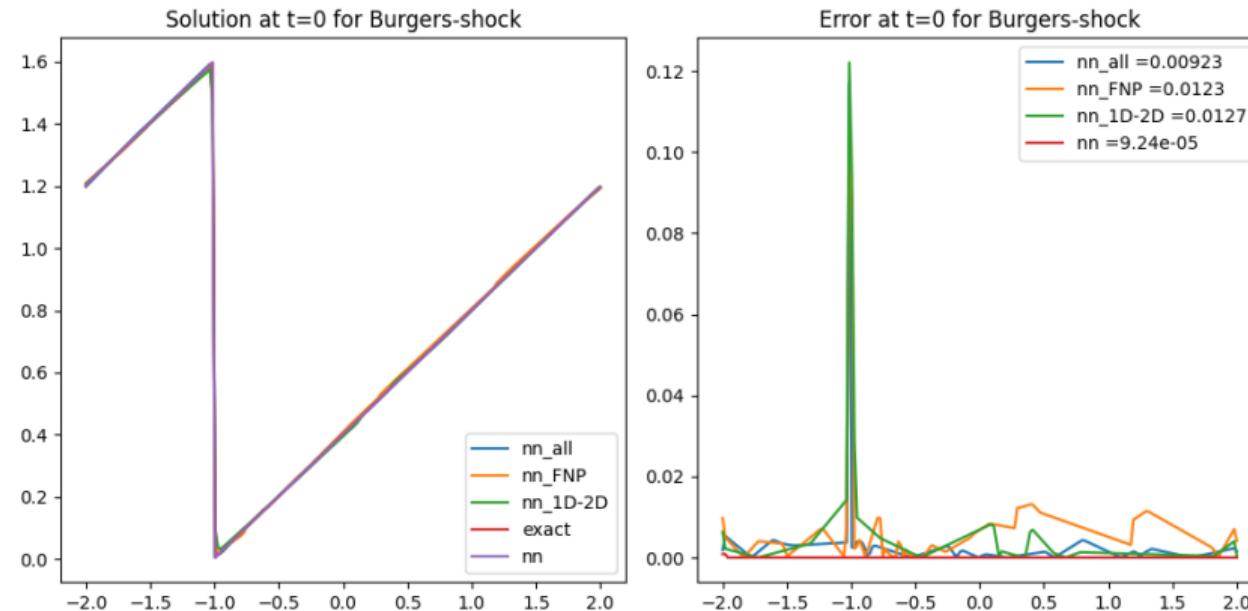


Adaptive 1D-as-2D



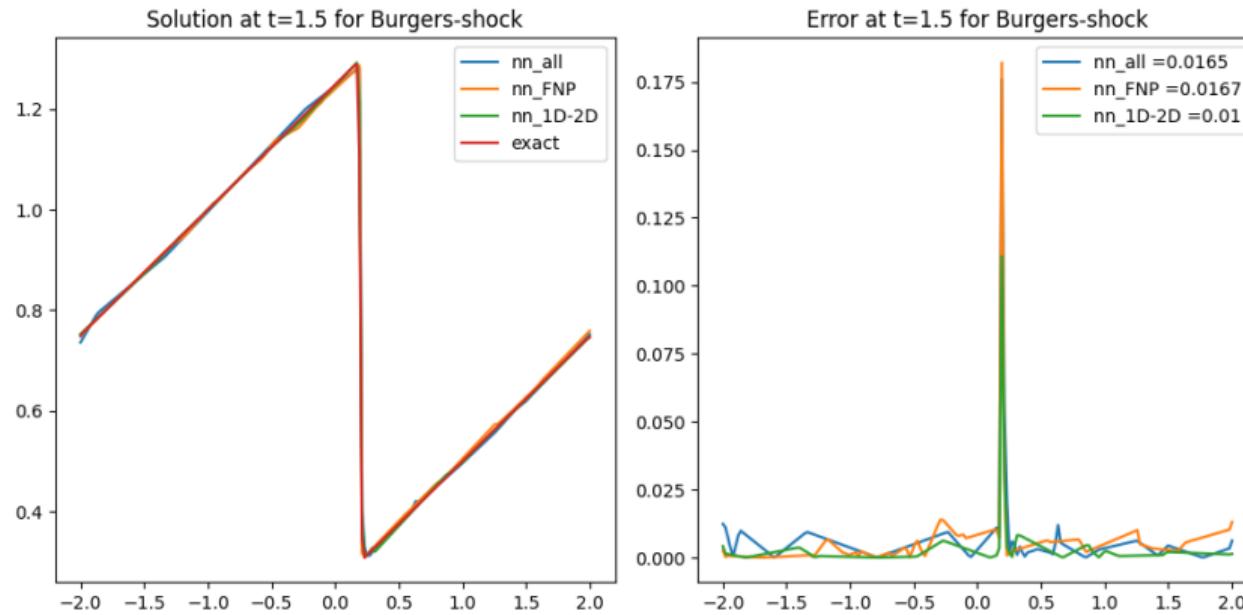
# Burgers shock

---



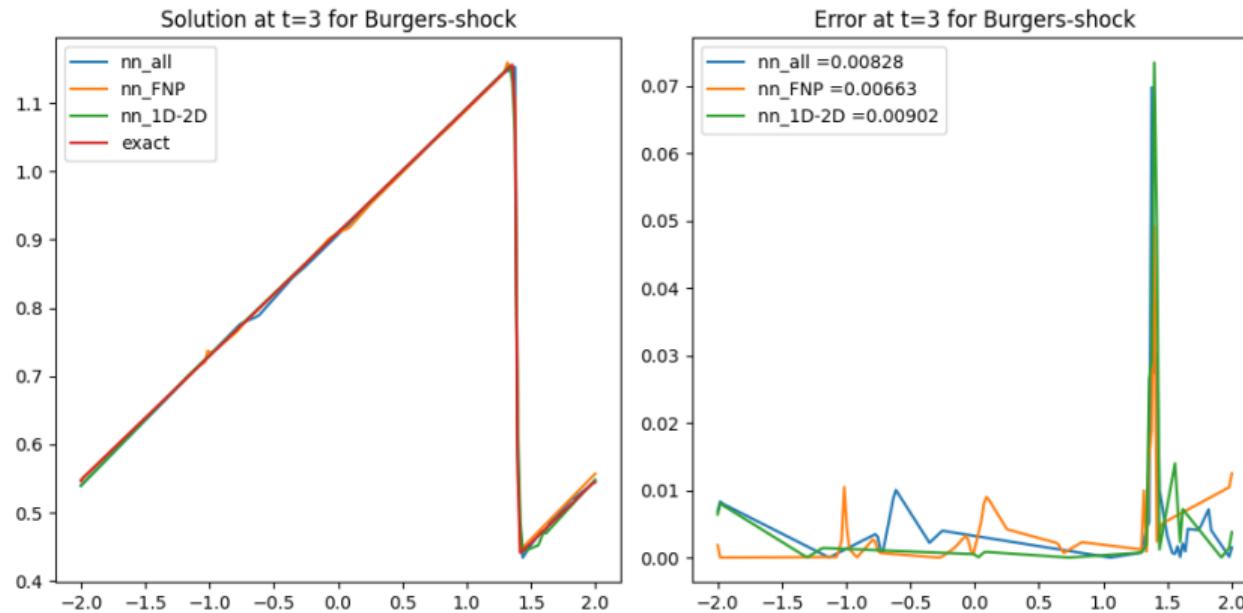
# Burgers shock

---



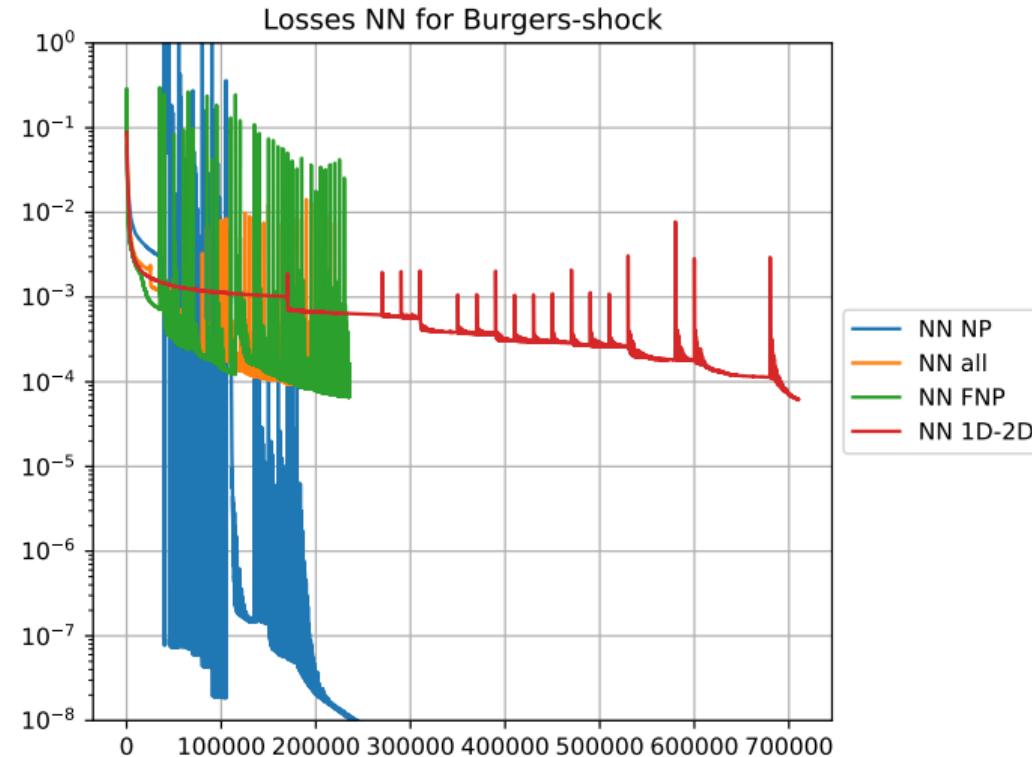
# Burgers shock

---



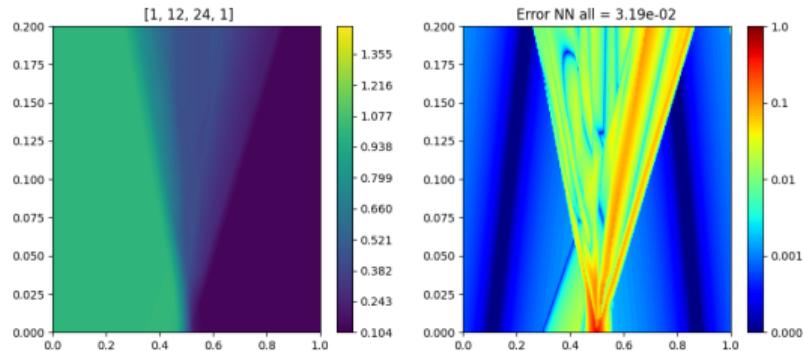
## Burgers shock

---



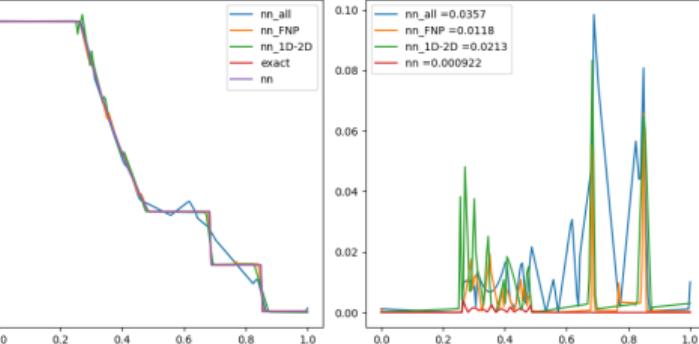
# Sod shock tube

Adaptive ALL

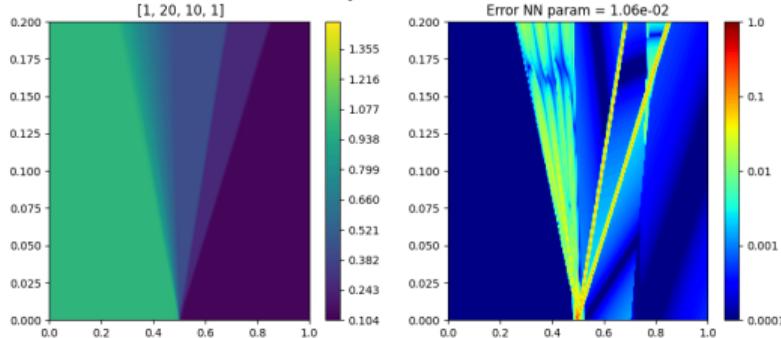


Comparison at final time

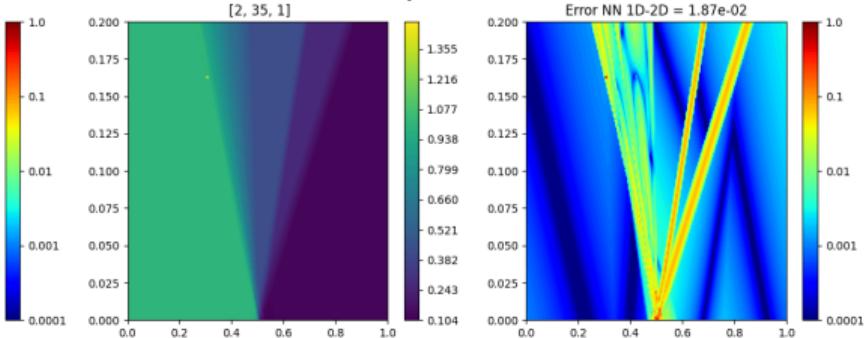
Solution at  $t=0.2$  for Sod



Adaptive FNP

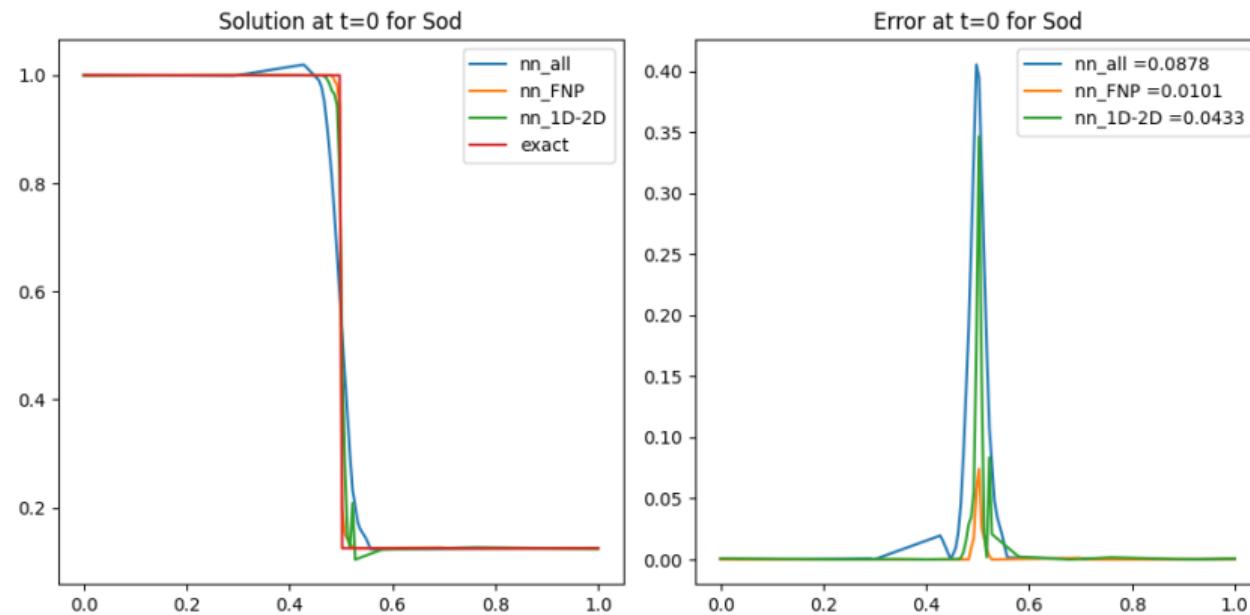


Adaptive 1D-as-2D



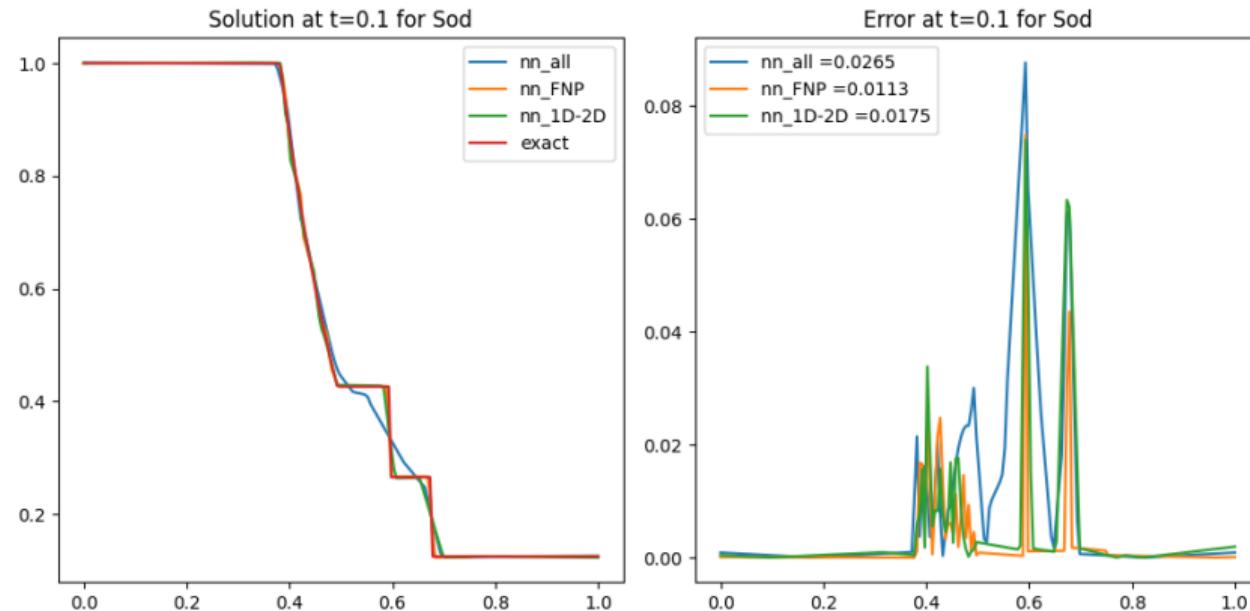
# Sod shock tube

---



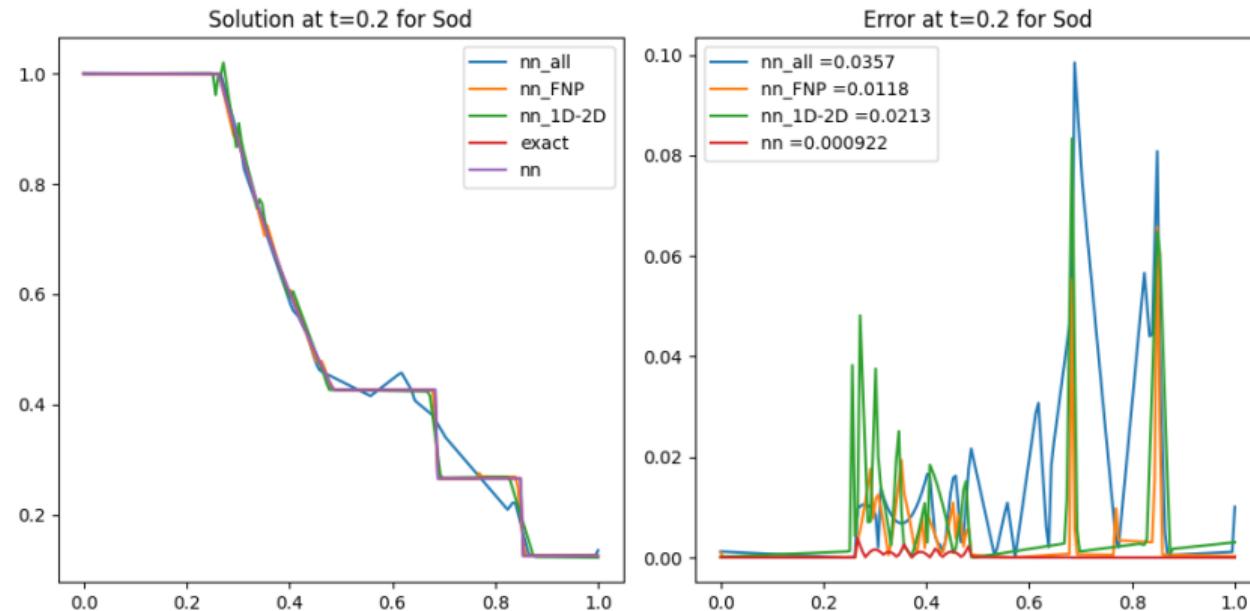
# Sod shock tube

---



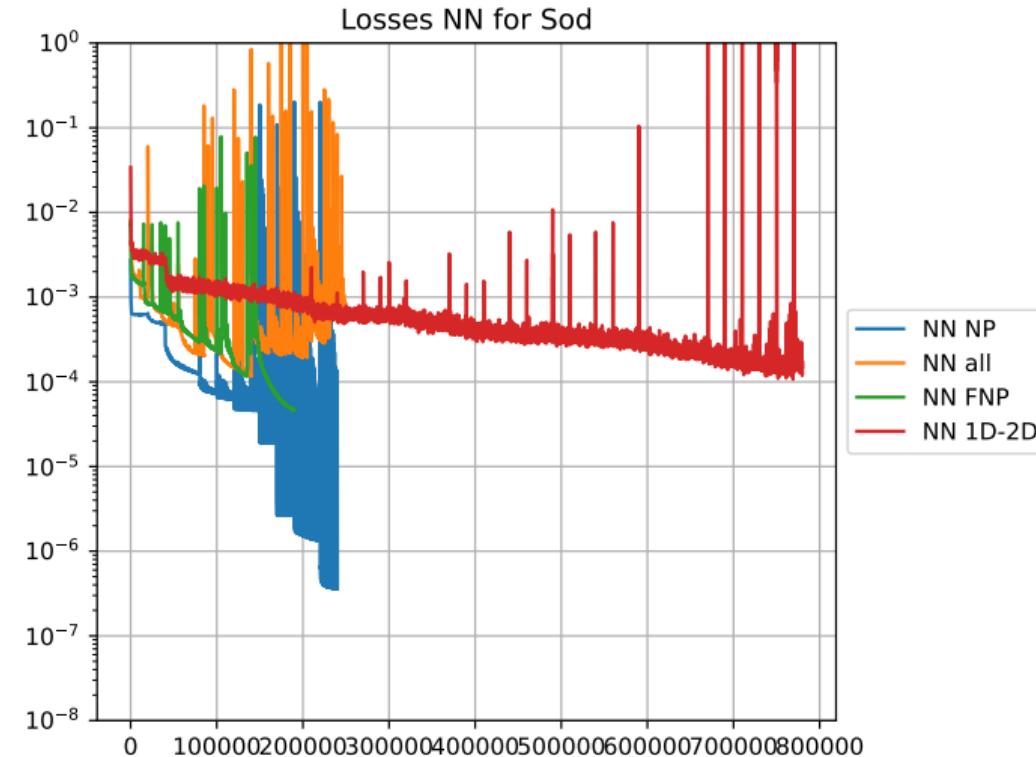
# Sod shock tube

---



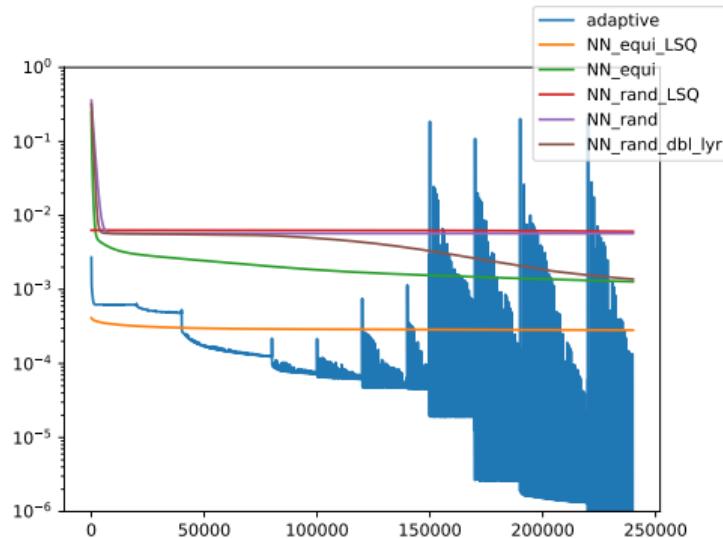
## Sod shock tube

---

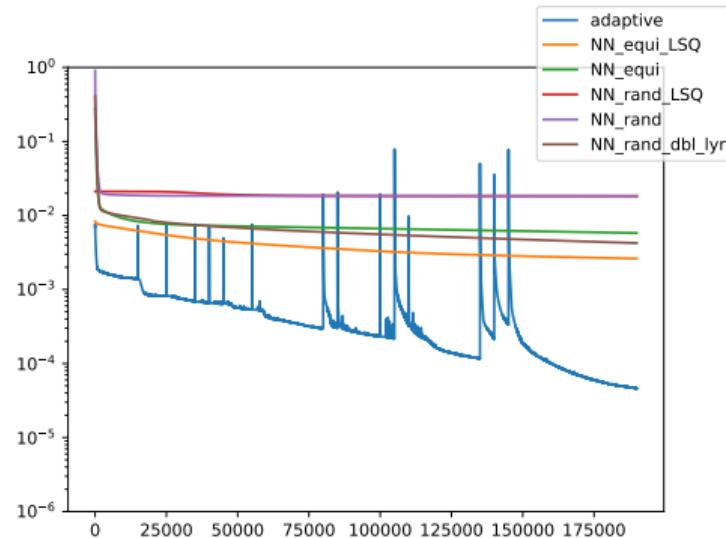


## Sod shock tube (comparison with non adaptive)

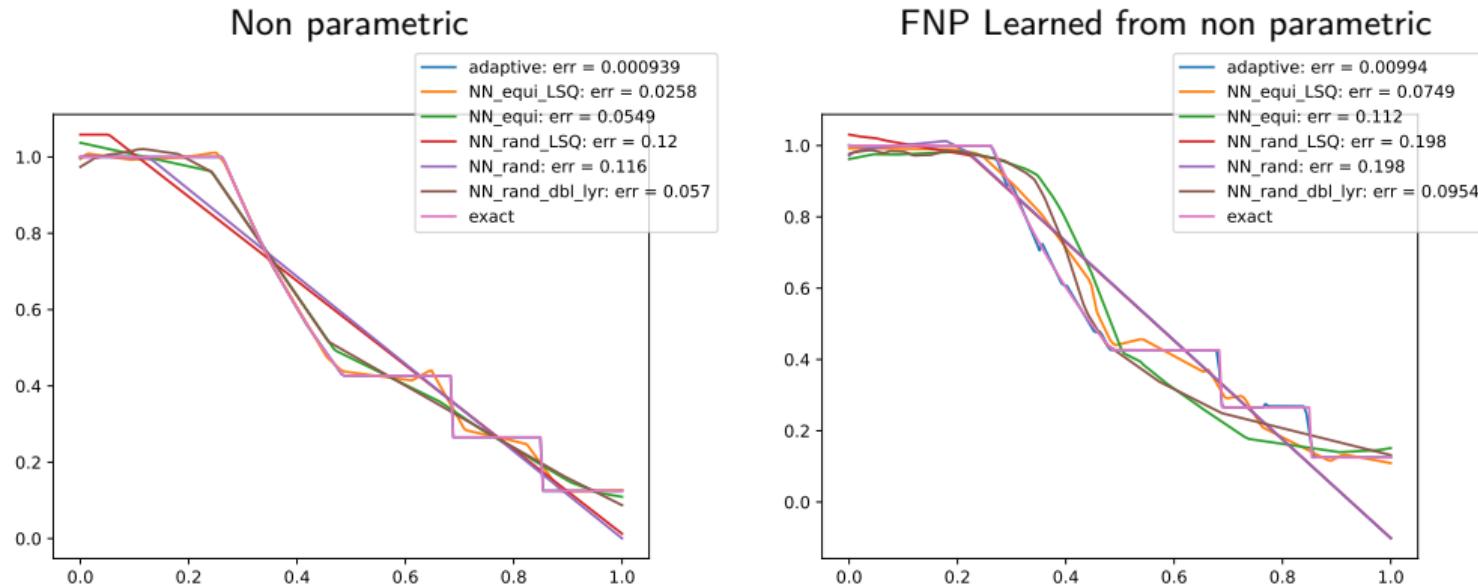
Non parametric



FNP Learned from non parametric

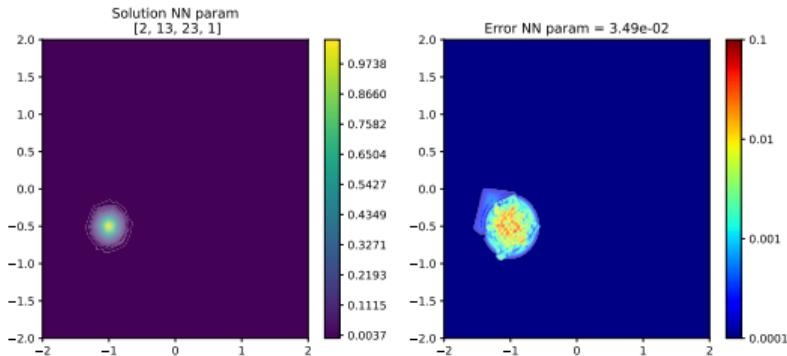


## Sod shock tube (comparison with non adaptive)

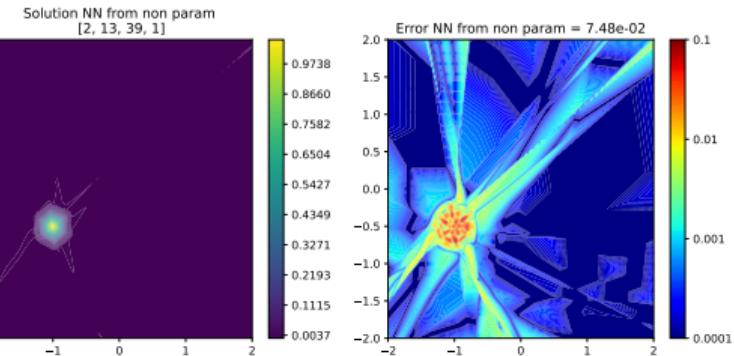


## 2D moving Gaussian

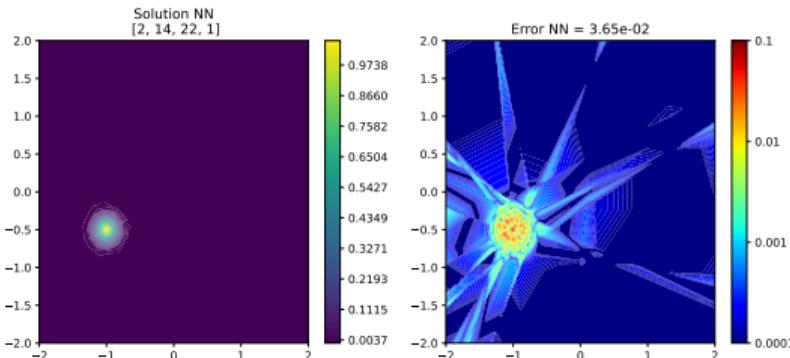
Adaptive ALL



Adaptive FNP



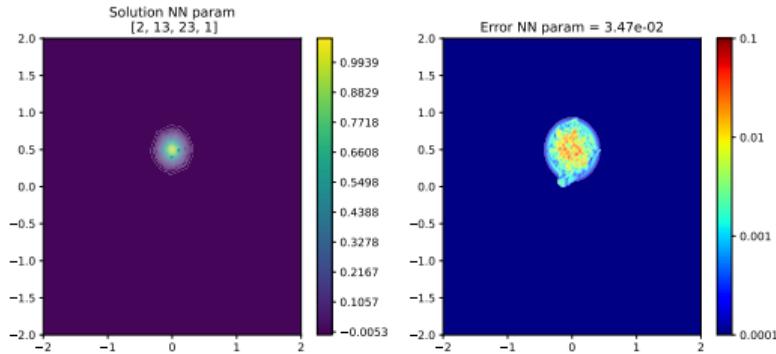
Adaptive non parametric



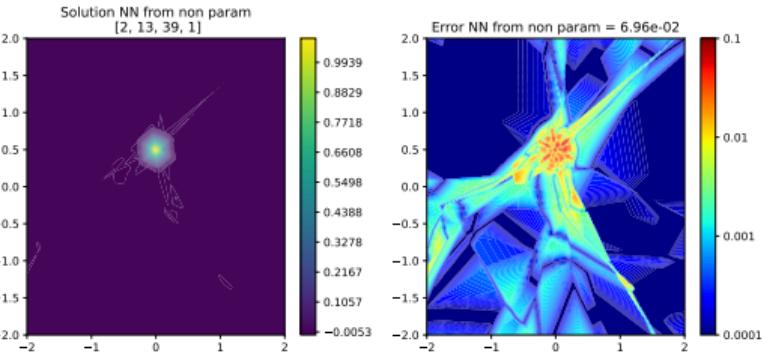
Initial time

## 2D moving Gaussian

Adaptive ALL

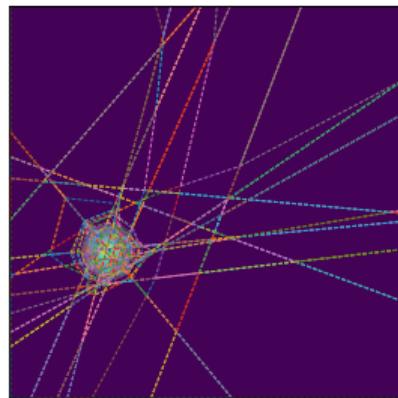


Adaptive FNP

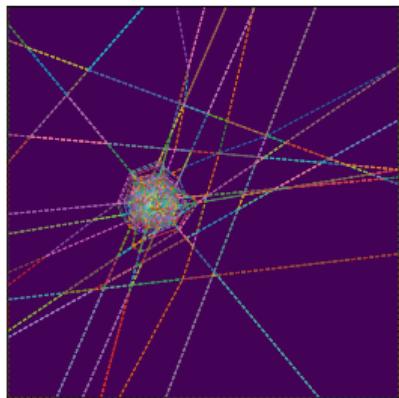


## 2D moving Gaussian

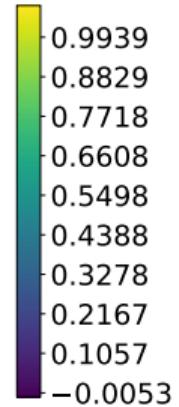
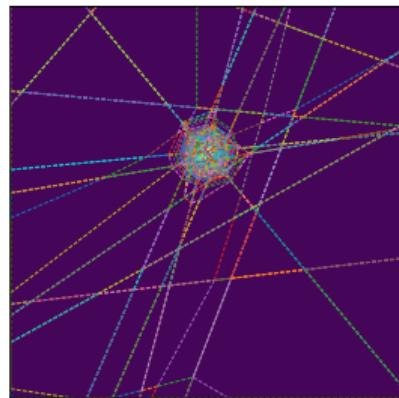
$t = 0$



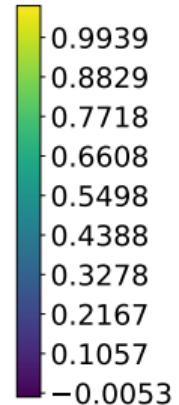
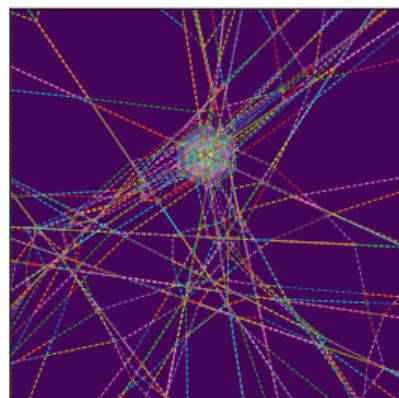
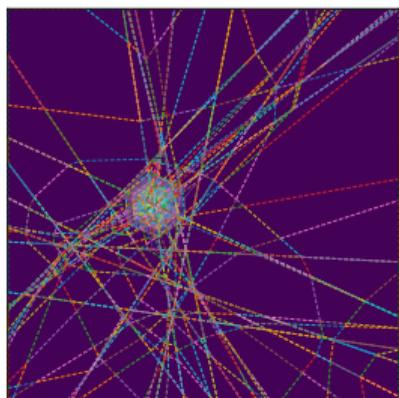
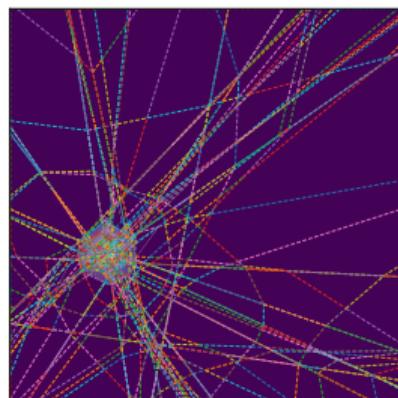
$t = 0.5$



$t = 1$



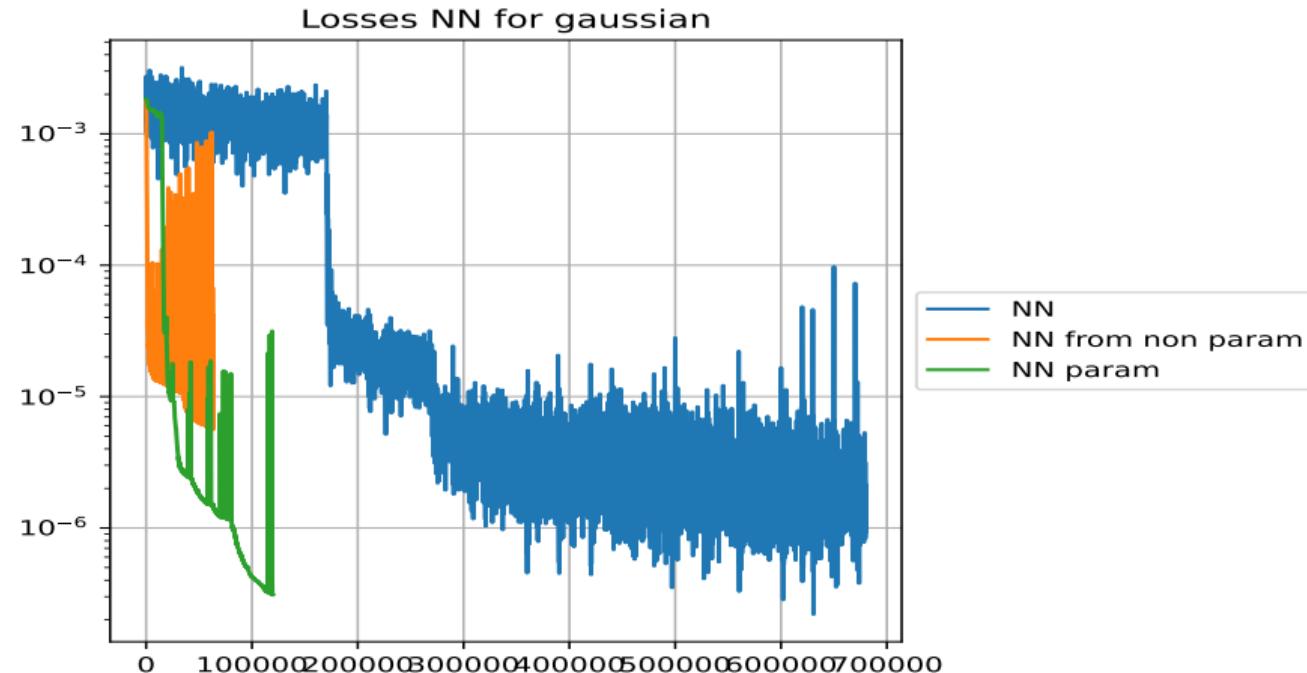
All



FNP

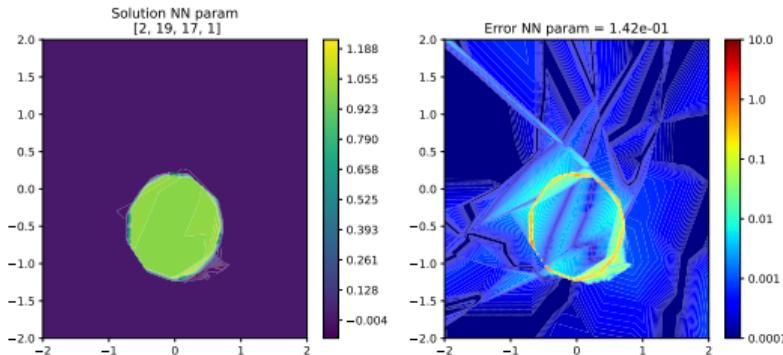
## 2D moving Gaussian

---

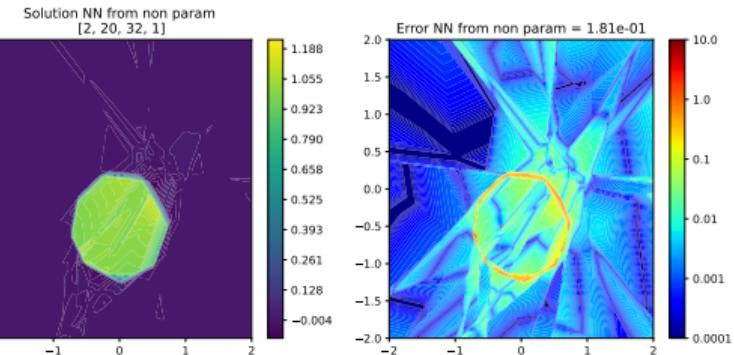


## 2D moving circle

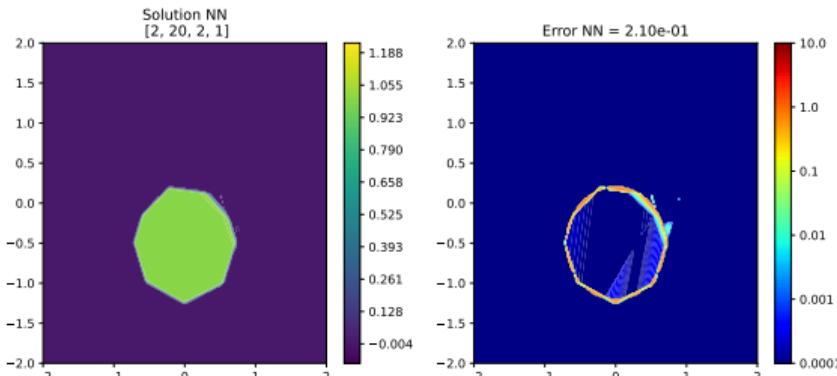
Adaptive ALL



Adaptive FNP



Adaptive non parametric

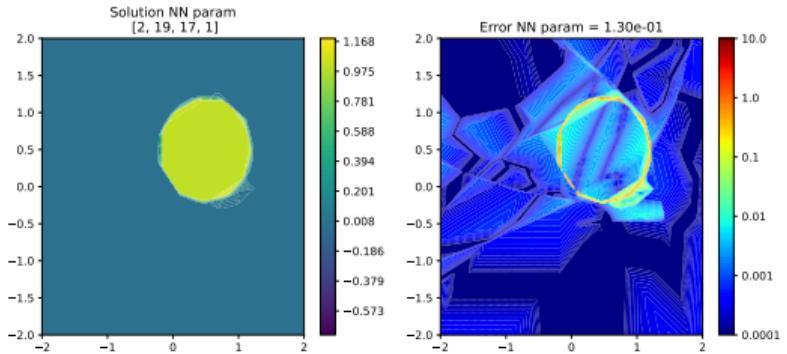


Initial time

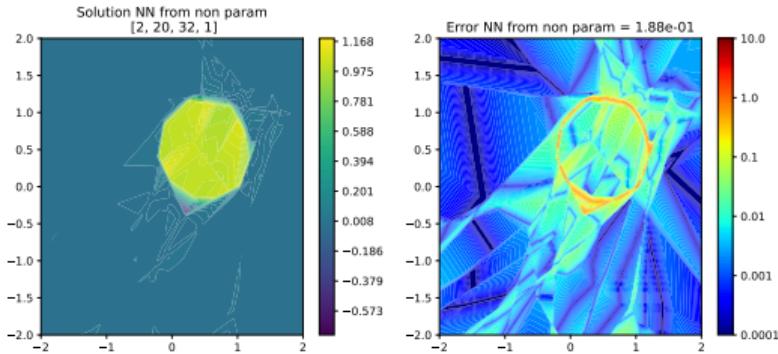
## 2D moving circle

---

Adaptive ALL



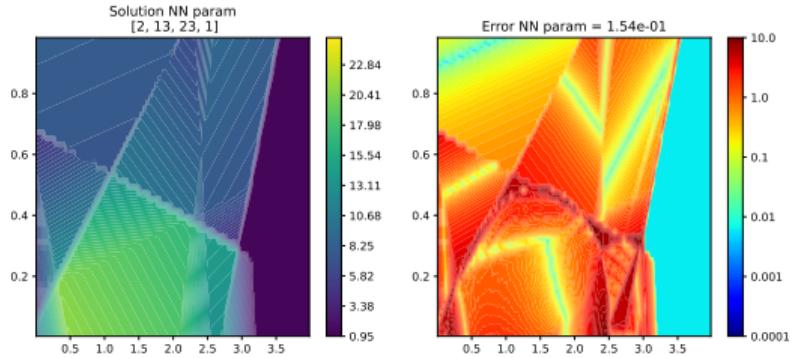
Adaptive FNP



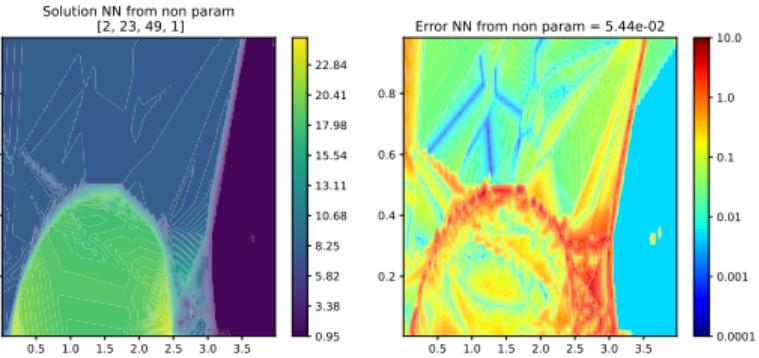
Final time

# 2D Double Mach Reflection for Euler's Equations

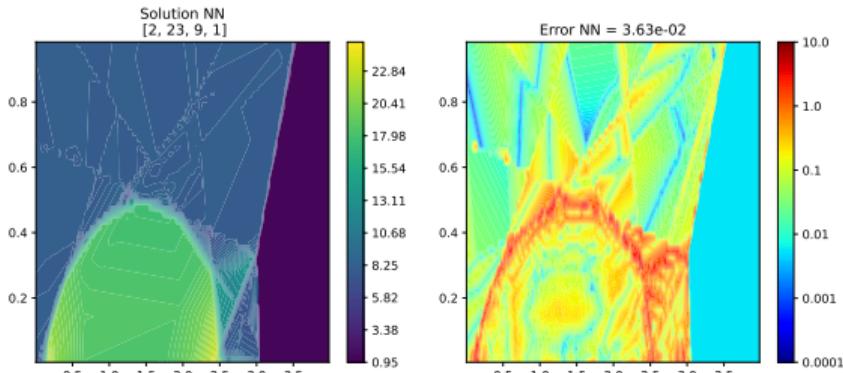
Adaptive ALL



Adaptive FNP

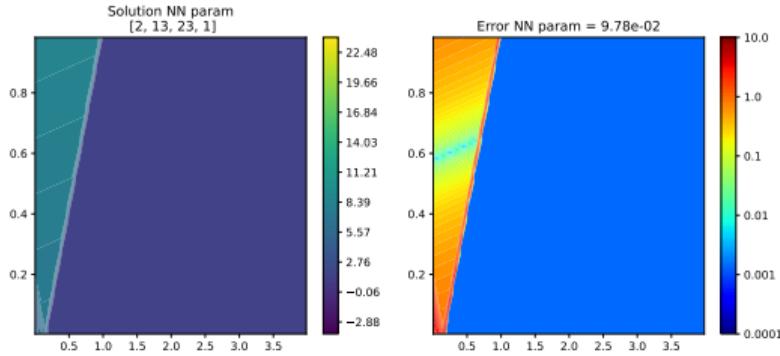


Adaptive non parametric

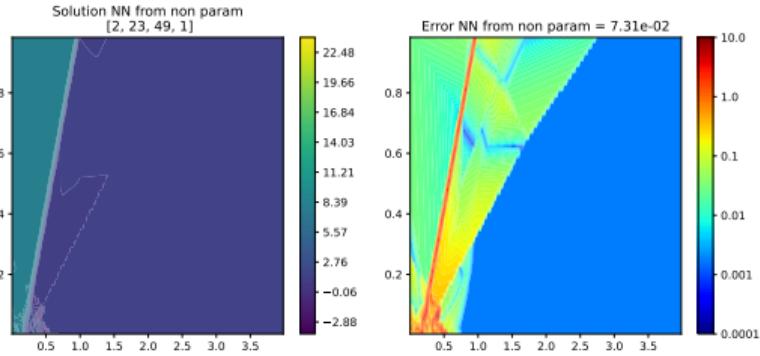


# 2D Double Mach Reflection for Euler's Equations

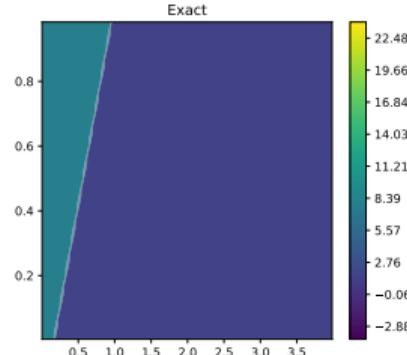
Adaptive ALL



Adaptive FNP



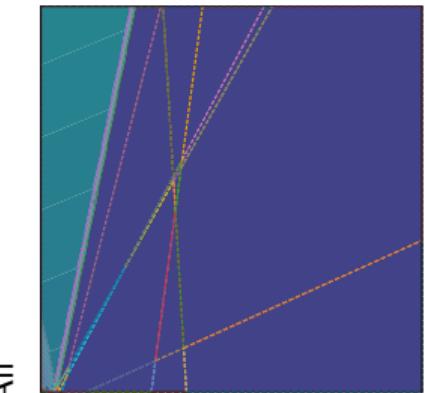
Exact



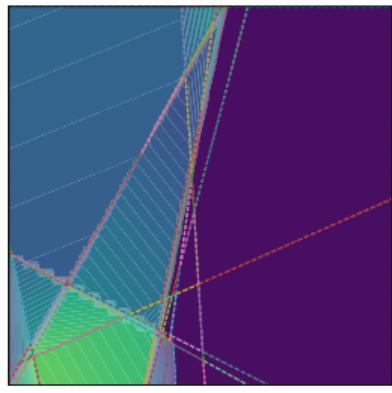
Initial time

## 2D Double Mach Reflection for Euler's Equations

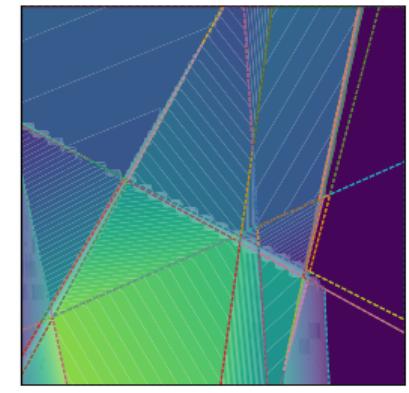
$t = 0$



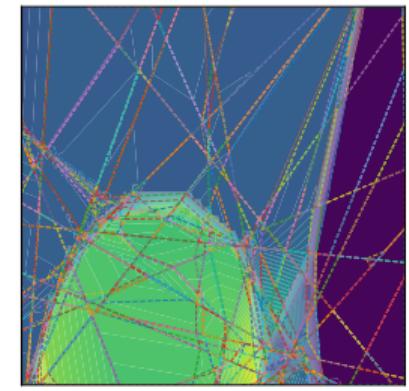
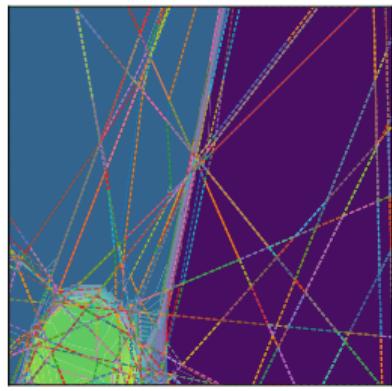
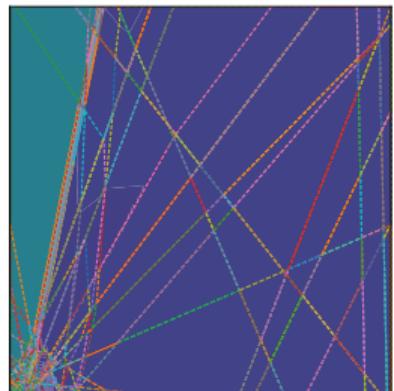
$t = 0.5T$



$t = T$



All



FNP

# Table of contents

---

① Adaptive NN

② Adaptive NN for Reduced Order Modeling  
Simulations

③ Conclusions

## Conclusions, limitations, perspectives

### Summary Adaptive NN

- Add nodes and layer iteratively
- Use ReLUs to define breaking lines to get mesh
- Error estimator for refining in the right position
- MOR with a given reference parameter

### Perspectives

- Try different configurations for parameter dependent weights and biases
- Hyperparameter tuning (how many new nodes all together, maximum number of nodes etc)
- 3D

## Conclusions, limitations, perspectives

### Summary Adaptive NN

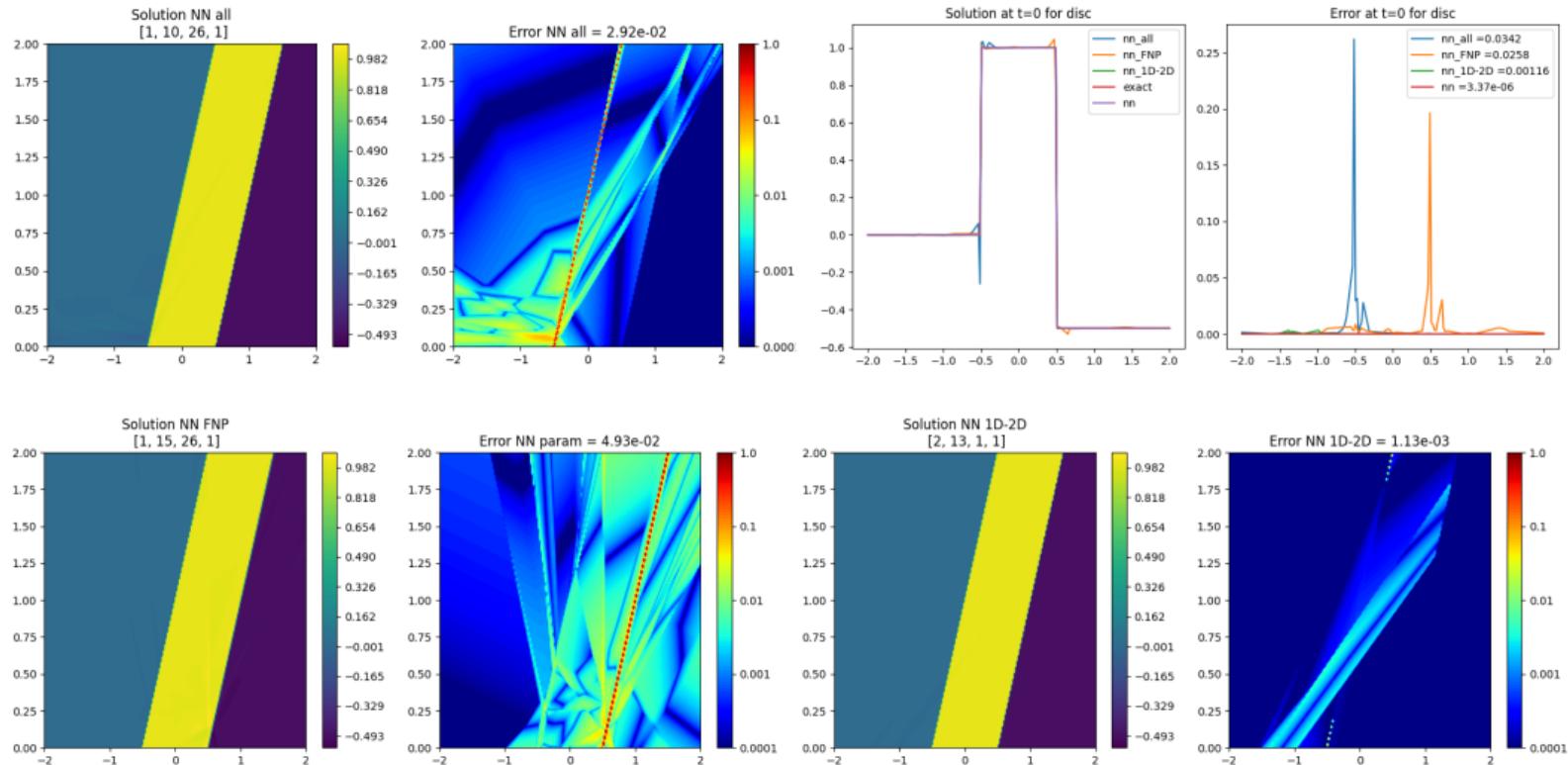
- Add nodes and layer iteratively
- Use ReLUs to define breaking lines to get mesh
- Error estimator for refining in the right position
- MOR with a given reference parameter

### Perspectives

- Try different configurations for parameter dependent weights and biases
- Hyperparameter tuning (how many new nodes all together, maximum number of nodes etc)
- 3D

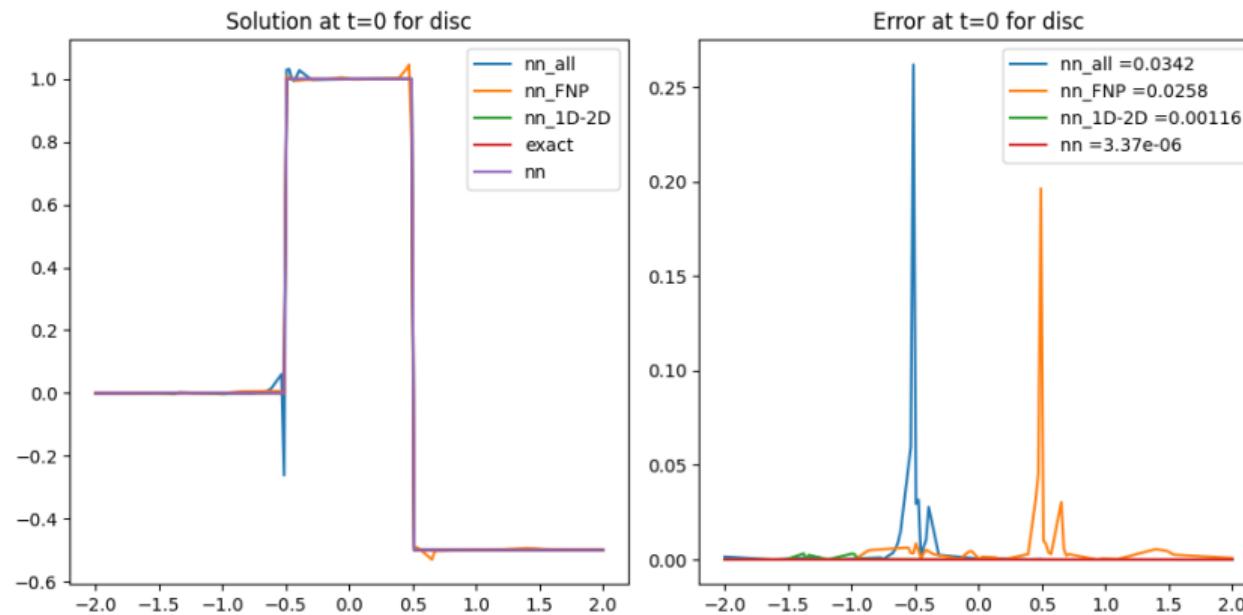
**THANKS!!**

# Moving discontinuity



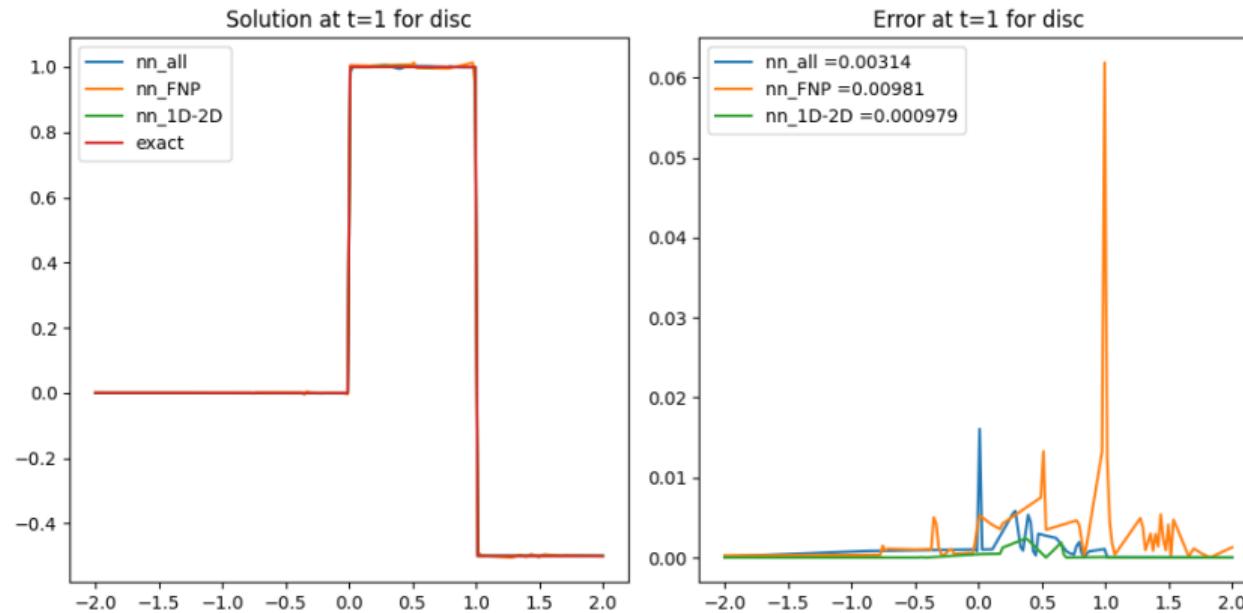
## Moving discontinuity

---



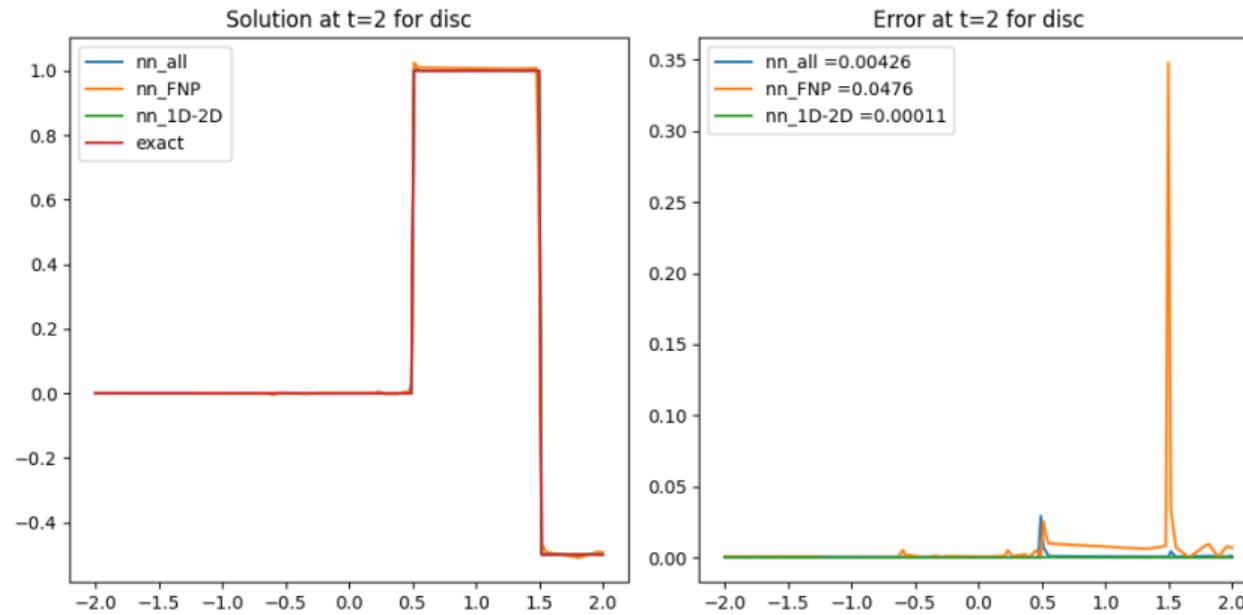
## Moving discontinuity

---



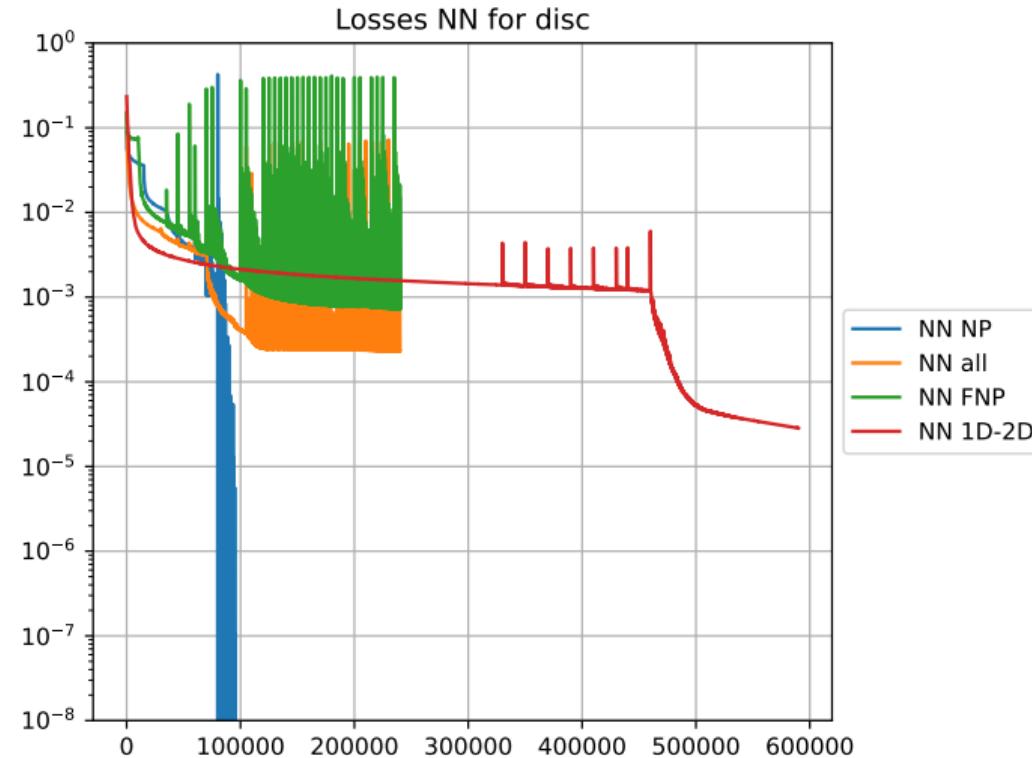
## Moving discontinuity

---

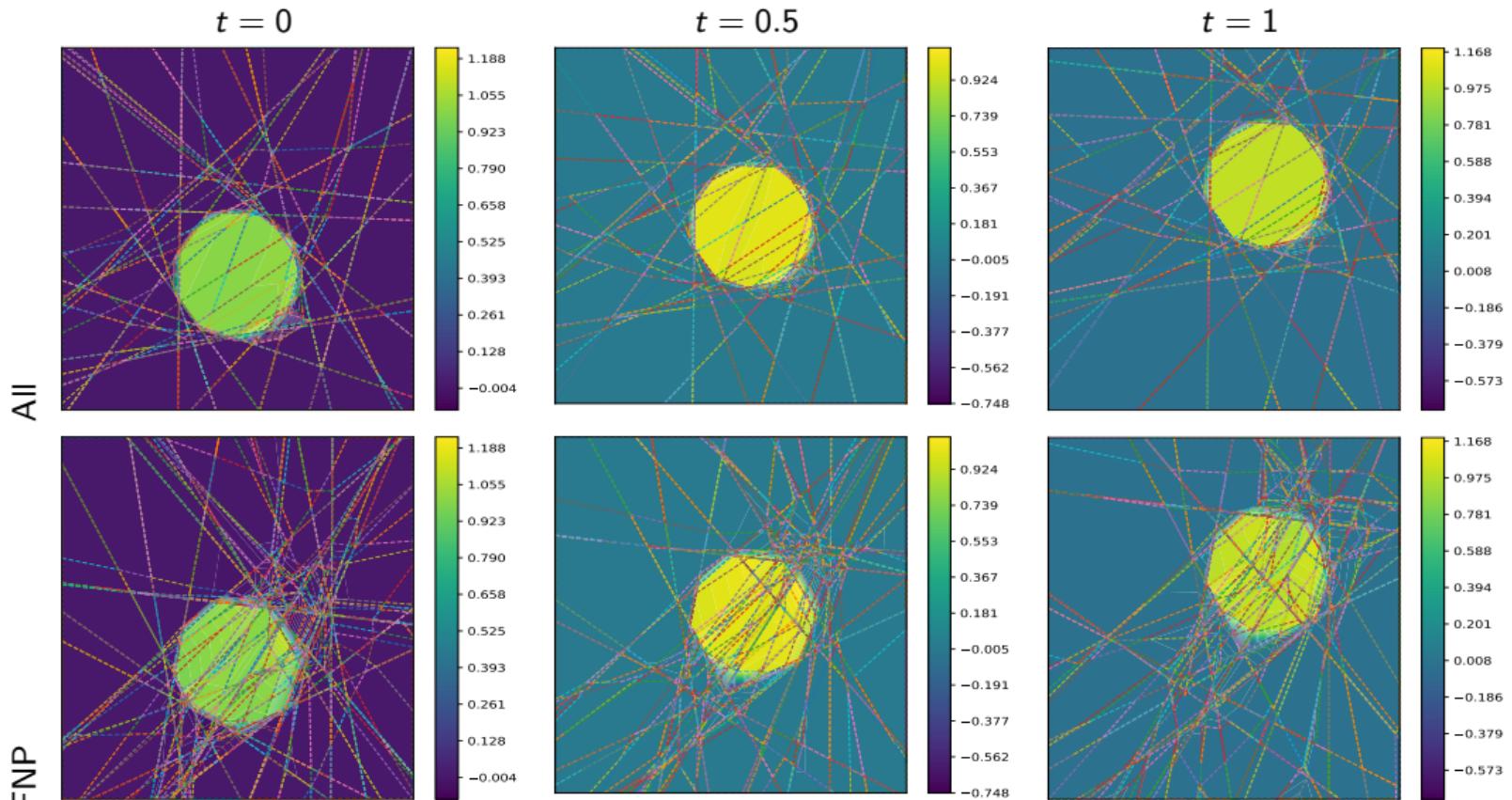


## Moving discontinuity

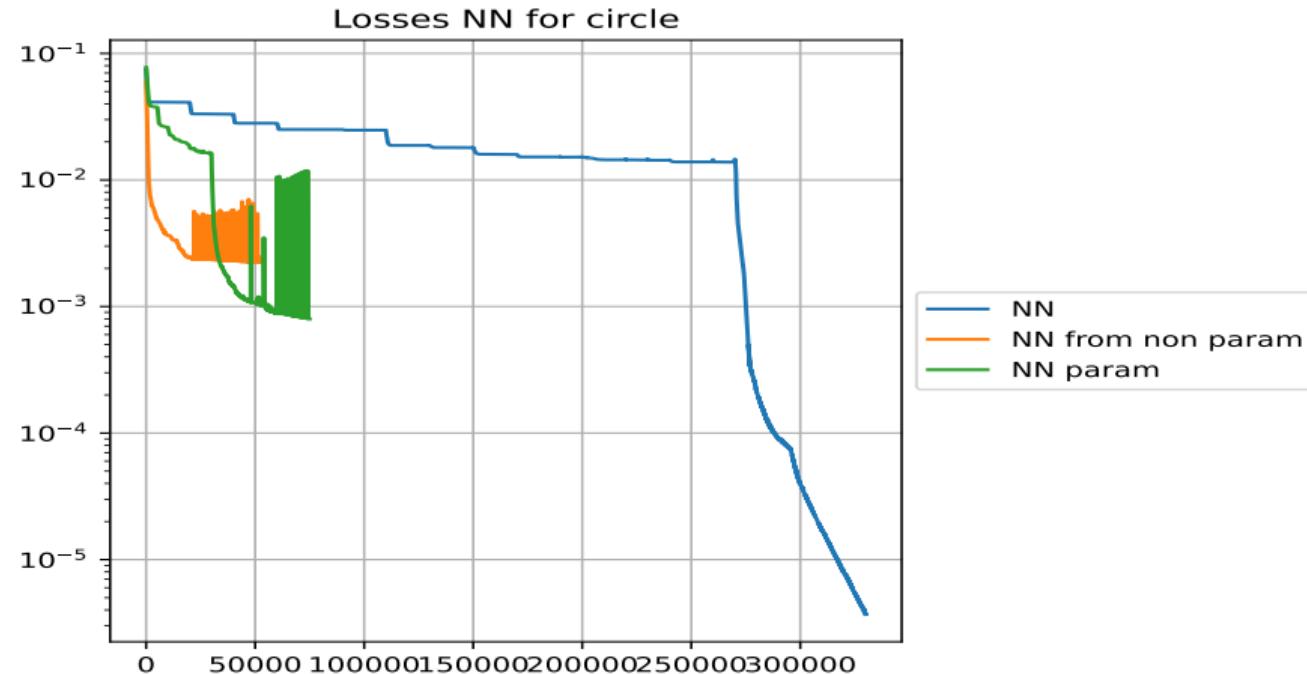
---



## 2D moving circle



## 2D moving circle



## 2D Double Mach Reflection for Euler's Equations

