Adaptive Neural Networks for Reduced Order Modeling

Davide Torlo, Federico Pichi

Dipartimento di Matematica "Guido Castelnuovo", Università di Roma La Sapienza, Italy
davidetorlo.it

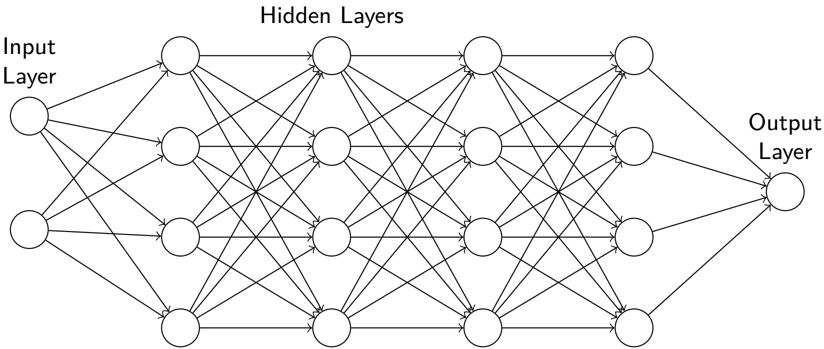
Trieste - 2nd September 2025

SAPIENZA
UNIVERSITÀ DI ROMA

Table of contents

Table of contents

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

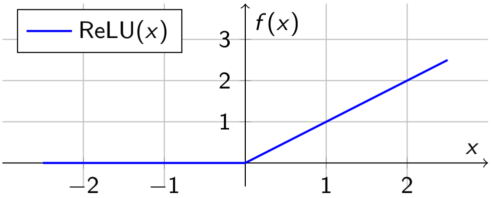
- 1 hidden layer
- few neurons
- ReLU activation function $\max(x,0)=x^+$
- Hat function: 1 hidden layer, 3 neurons

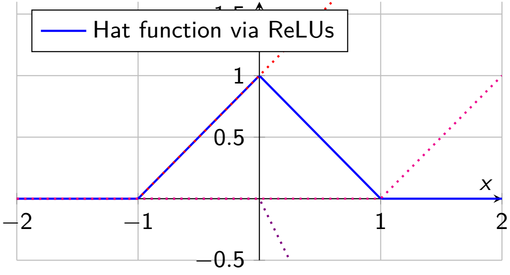
$$h(x) = (x+1)^+ - 2(x)^+ + (x-1)^+$$

- 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

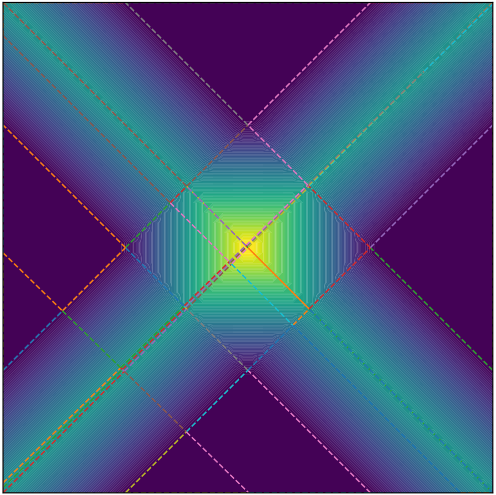
- 1 hidden layer
- few neurons
- ReLU activation function $\max(x, 0) = x^+$
- Hat function: 1 hidden layer, 3 neurons

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

- 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)



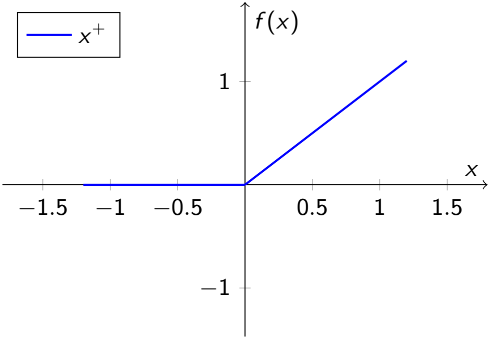
Rediscovering shallow NN: ReLU 2 hidden layers in 1D

2 hidden layers NN

- Speed up the process of geometrical subdivision
- In 1D, for example, easy to discretize discontinuities up to ε 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**



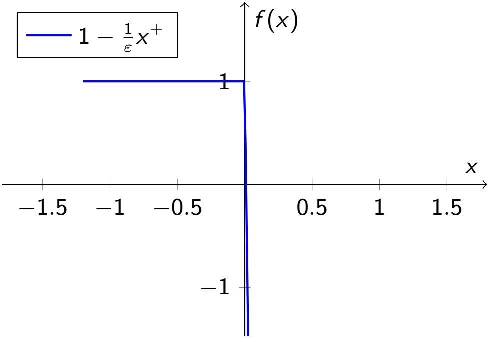
Rediscovering shallow NN: ReLU 2 hidden layers in 1D

2 hidden layers NN

- Speed up the process of geometrical subdivision
- In 1D, for example, easy to discretize discontinuities up to ε 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**



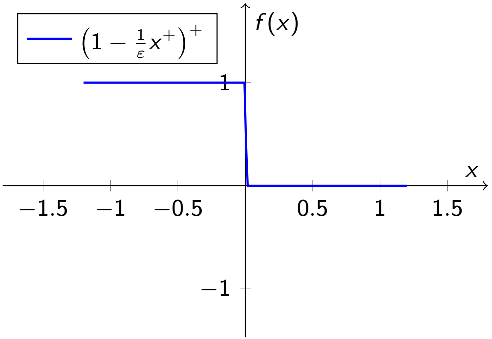
Rediscovering shallow NN: ReLU 2 hidden layers in 1D

2 hidden layers NN

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

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

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



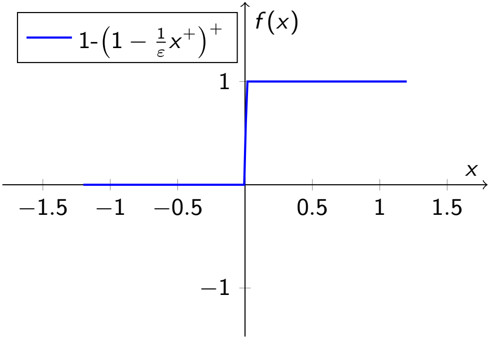
Rediscovering shallow NN: ReLU 2 hidden layers in 1D

2 hidden layers NN

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

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

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



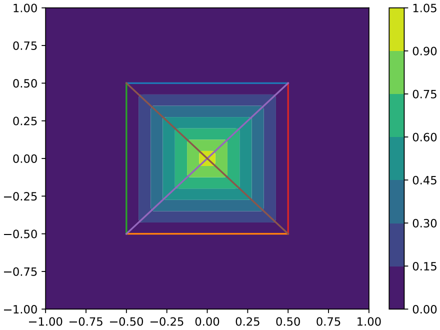
Rediscovering shallow NN: ReLU 2 hidden layers in 1D

2 hidden layers NN

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

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

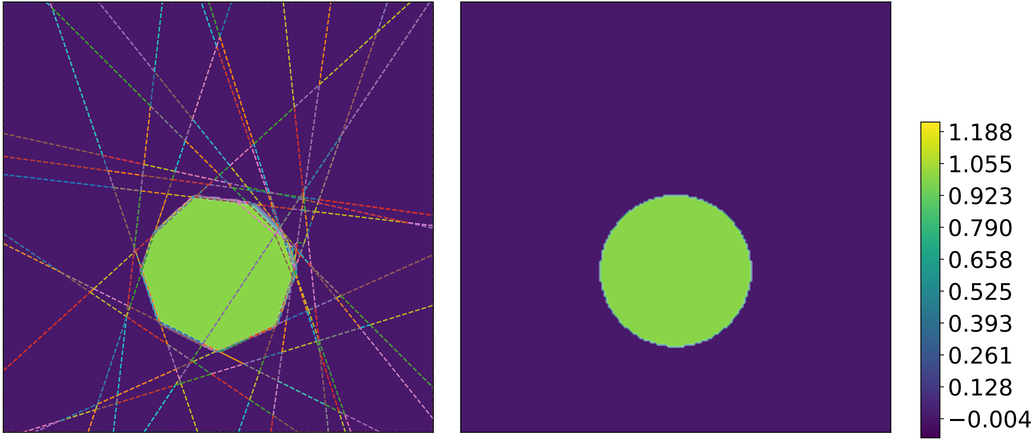
- Less sensitive to hyperparameters (to do the same with 1 hidden layer the different weights have to match exactly)
- **Fully interpretable**
- Easy 2D Hat function: $(1-(x-y)^+-(y-x)^+-(x+y)^+-(-x-y)^+)^+$



Rediscovering shallow NN: ReLU 2 hidden layers in 2D

2 hidden layers NN

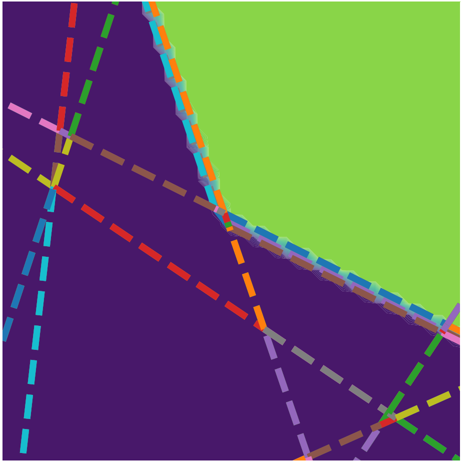
- More easily gets steep gradients
- Speed up the process of geometrical subdivision

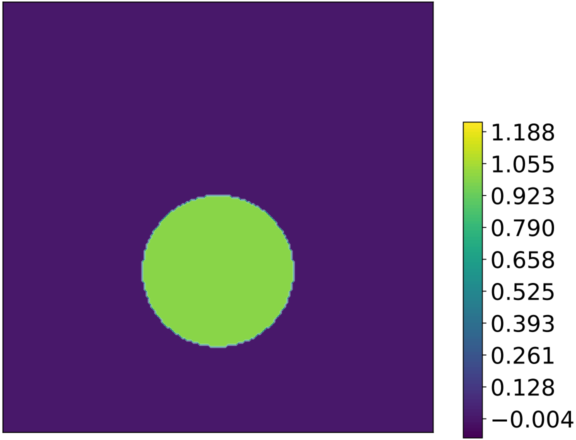


Rediscovering shallow NN: ReLU 2 hidden layers in 2D

2 hidden layers NN

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

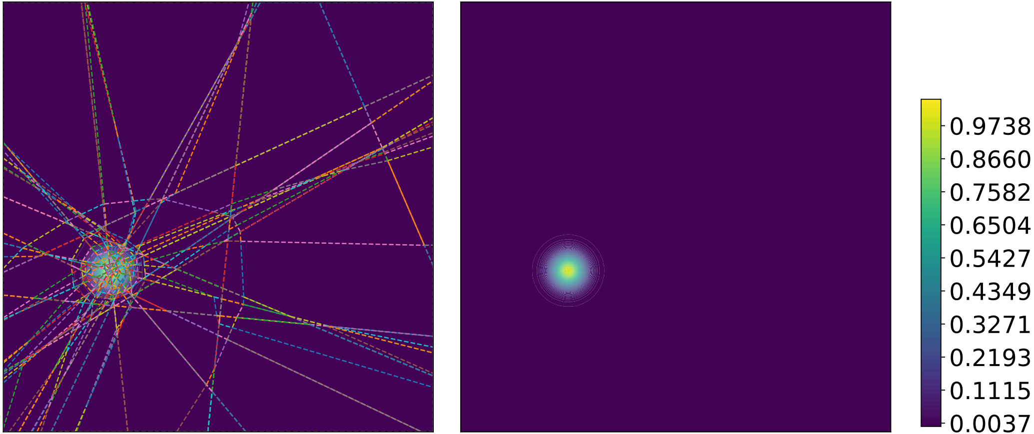




Rediscovering shallow NN: ReLU 2 hidden layers in 2D

2 hidden layers NN

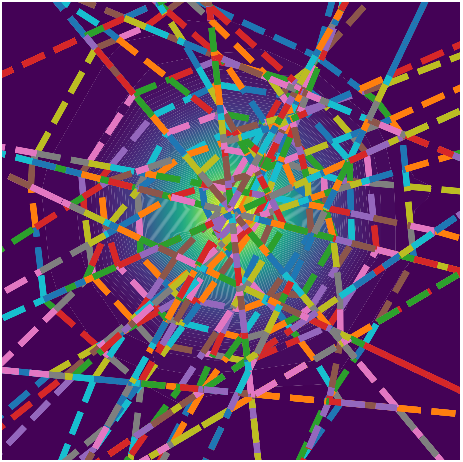
- More easily gets steep gradients
- Speed up the process of geometrical subdivision

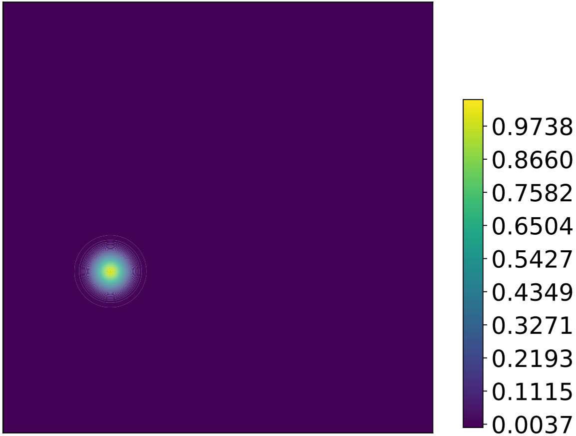


Rediscovering shallow NN: ReLU 2 hidden layers in 2D

2 hidden layers NN

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





Towards simpler NN

What we have learned?

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

$$A^1_{i,:}(A^0x + b^0) + b^1_i = 0 \qquad \forall i$$

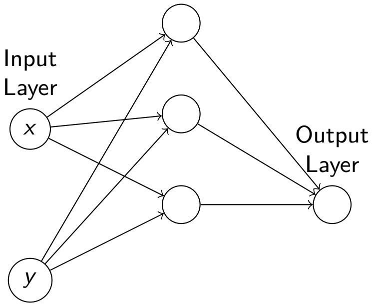
Goals: Adaptive NN

- 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[1]

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)

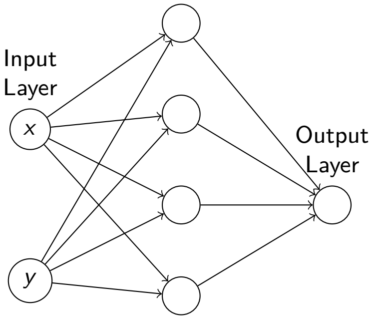


[1] Cai, Zhiqiang, Jingshuang Chen, and Min Liu. "Self-adaptive deep neural network: Numerical approximation to functions and PDEs." Journal of Computational Physics 455 (2022): 111021.

Adaptive NN[1]

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)

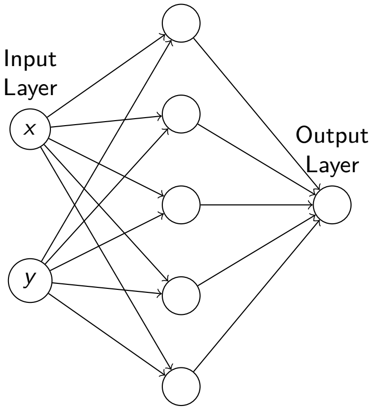


[1]Cai, Zhiqiang, Jingshuang Chen, and Min Liu. "Self-adaptive deep neural network: Numerical approximation to functions and PDEs." Journal of Computational Physics 455 (2022): 111021.

Adaptive NN[1]

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)

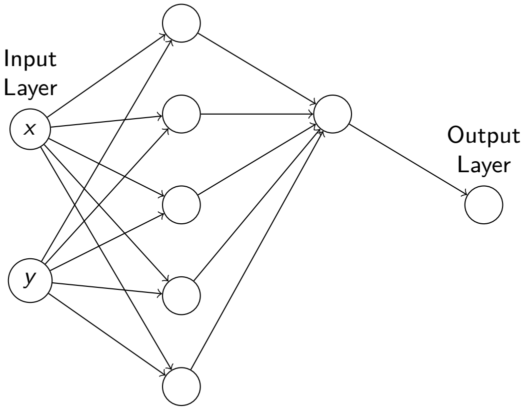


[1] Cai, Zhiqiang, Jingshuang Chen, and Min Liu. "Self-adaptive deep neural network: Numerical approximation to functions and PDEs." Journal of Computational Physics 455 (2022): 111021.

Adaptive NN[1]

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)

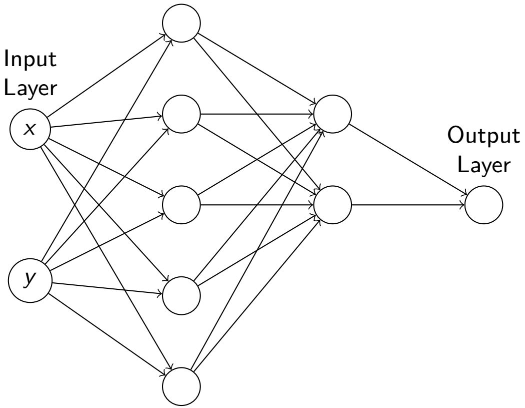


[1] Cai, Zhiqiang, Jingshuang Chen, and Min Liu. "Self-adaptive deep neural network: Numerical approximation to functions and PDEs." Journal of Computational Physics 455 (2022): 111021.

Adaptive NN[1]

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)

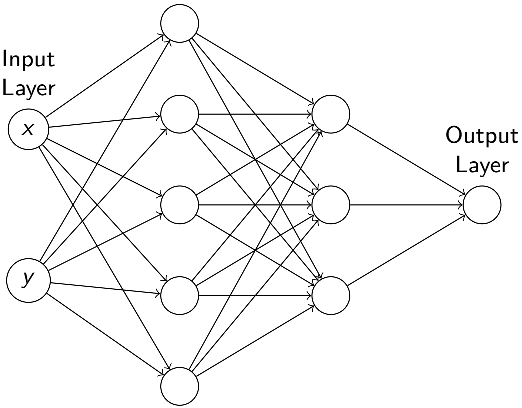


[1]Cai, Zhiqiang, Jingshuang Chen, and Min Liu. "Self-adaptive deep neural network: Numerical approximation to functions and PDEs." Journal of Computational Physics 455 (2022): 111021.

Adaptive NN[1]

Incremental architecture

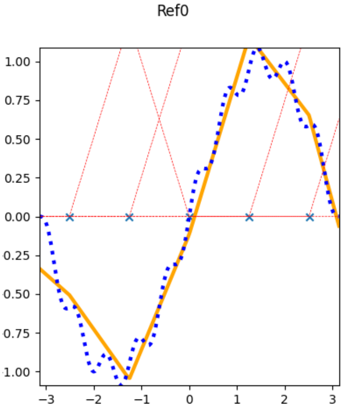
- No *a priori* knowledge of how many layers/neurons are needed
- Increment nodes and layers until a tolerance
- Add neurons so that the new breaking line falls in the worst represented part
- Proceed with optimization process (Adam)



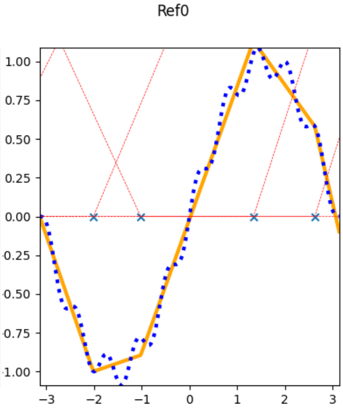
[1]Cai, Zhiqiang, Jingshuang Chen, and Min Liu. "Self-adaptive deep neural network: Numerical approximation to functions and PDEs." Journal of Computational Physics 455 (2022): 111021.

Example of adaptive NN

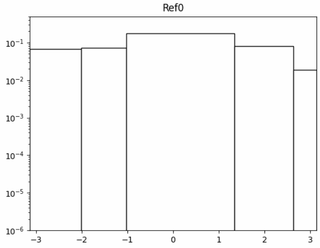
Before Training



After Training

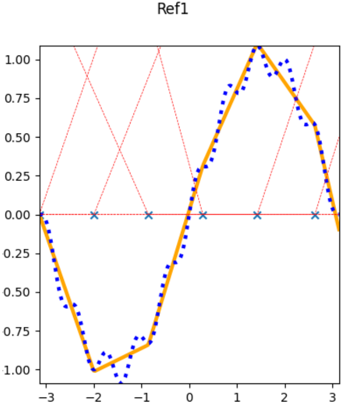


Error In Cells

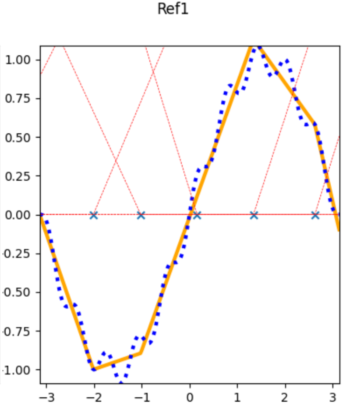


Example of adaptive NN

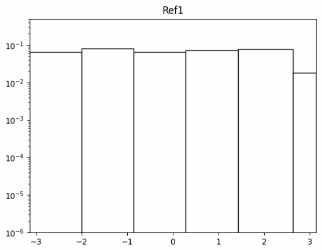
After Training



Before Training

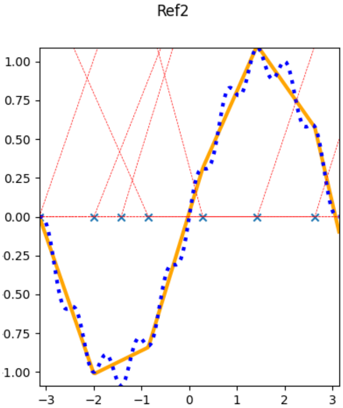


Error In Cells

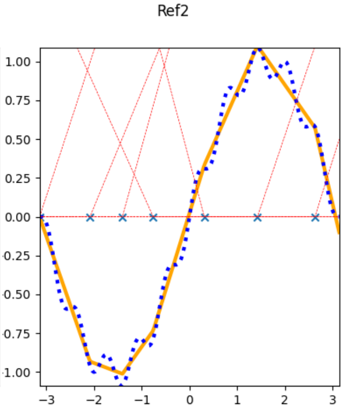


Example of adaptive NN

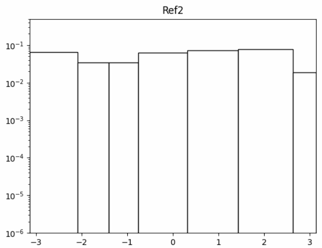
Before Training



After Training

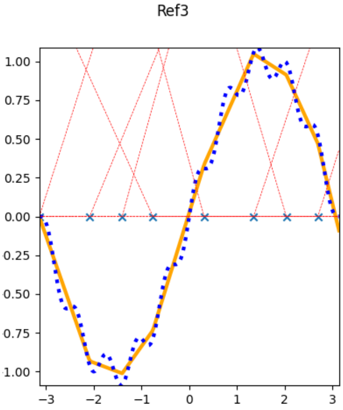


Error In Cells

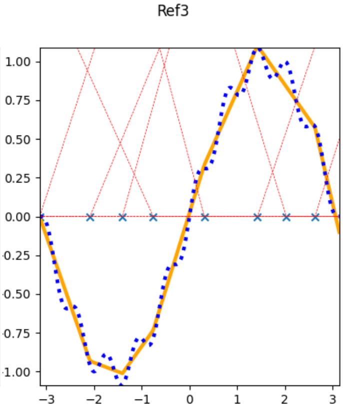


Example of adaptive NN

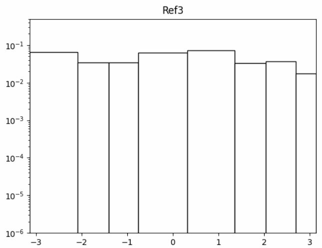
After Training



Before Training

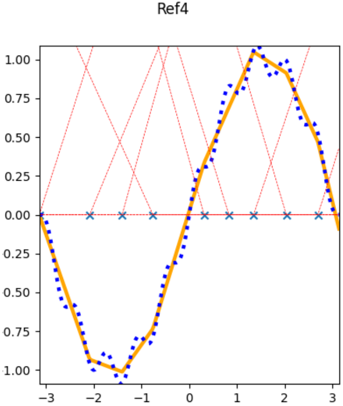


Error In Cells

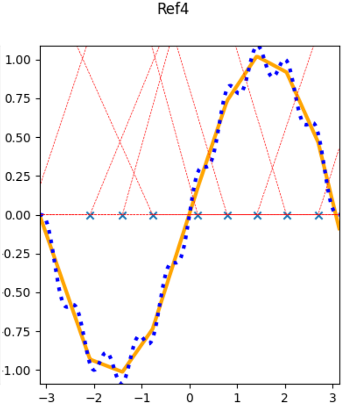


Example of adaptive NN

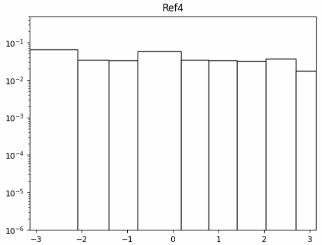
Before Training



After Training

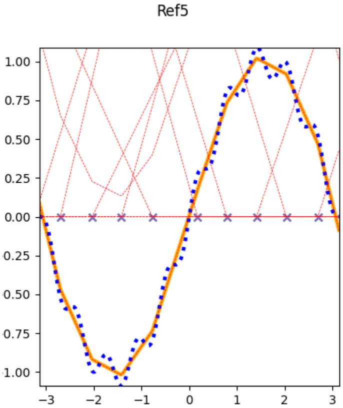


Error In Cells

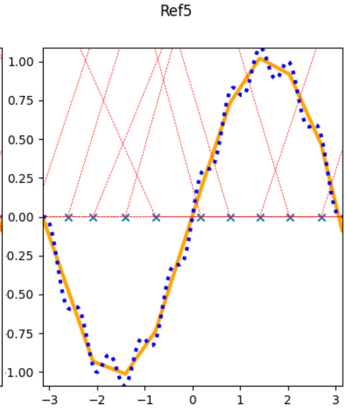


Example of adaptive NN

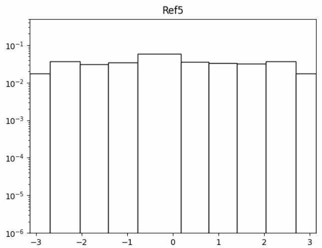
After Training



Before Training

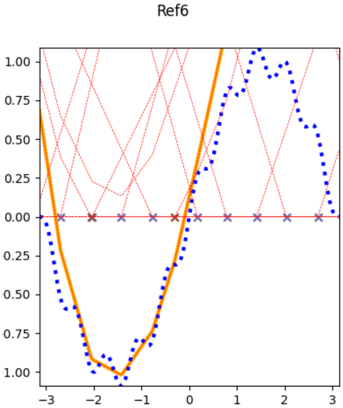


Error In Cells

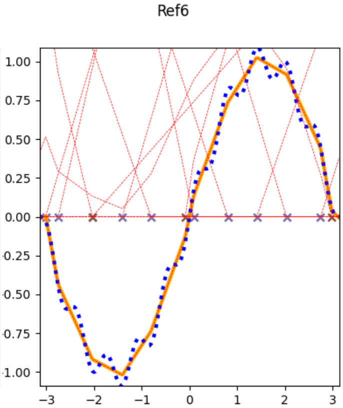


Example of adaptive NN

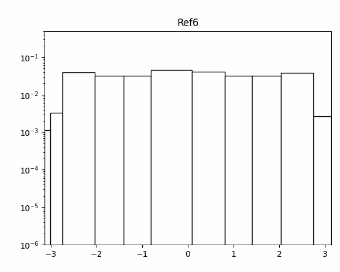
Before Training



After Training

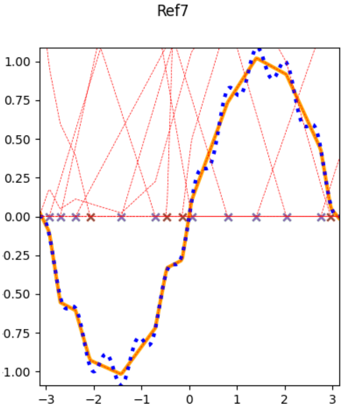


Error In Cells

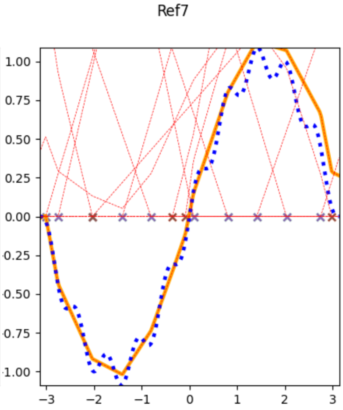


Example of adaptive NN

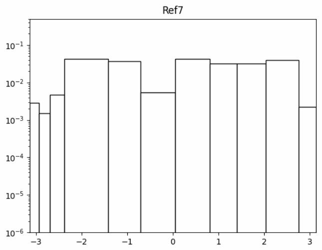
After Training



Before Training

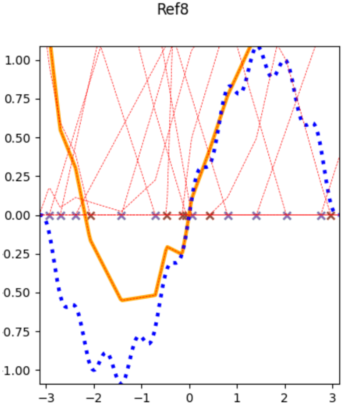


Error In Cells

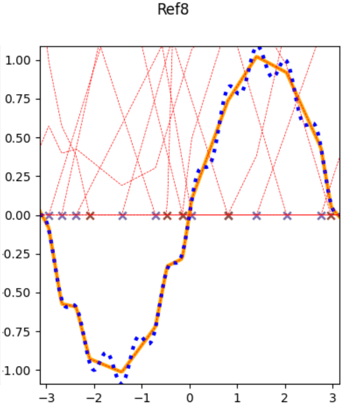


Example of adaptive NN

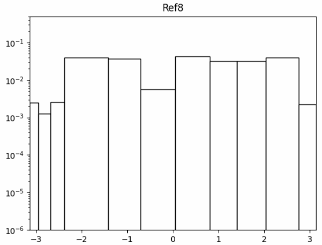
Before Training



After Training

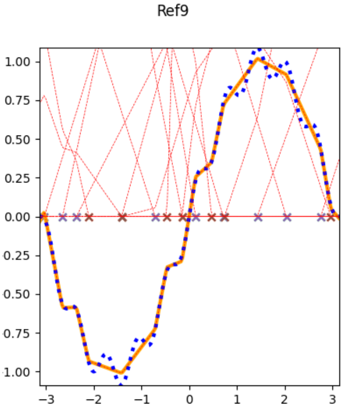


Error In Cells

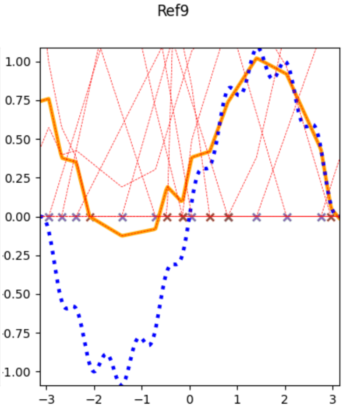


Example of adaptive NN

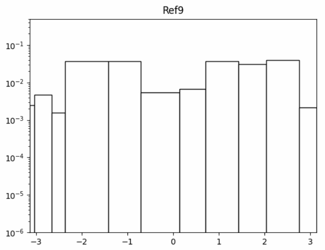
After Training



Before Training

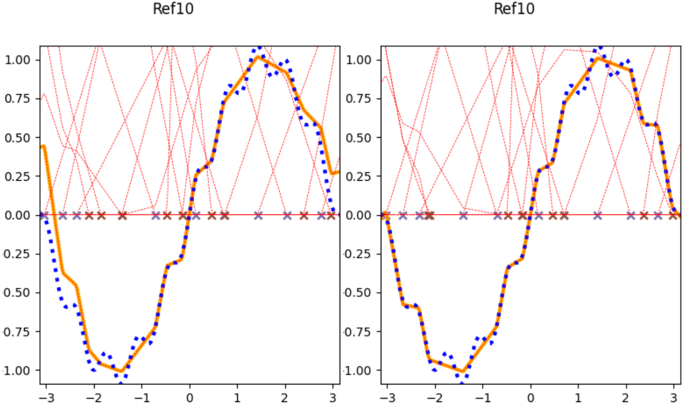


Error In Cells

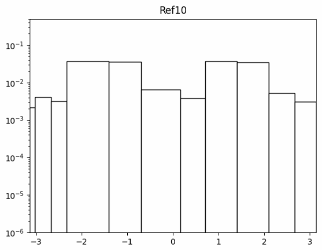


Example of adaptive NN

Before Training | After Training

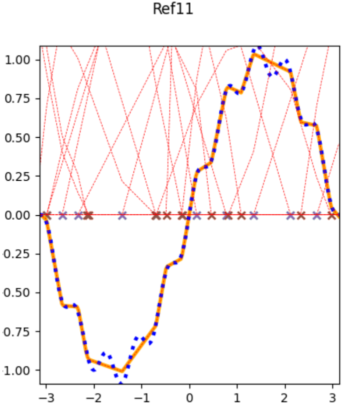


Error In Cells

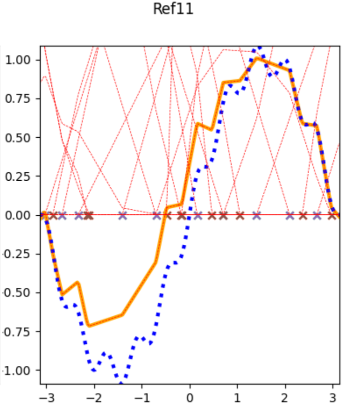


Example of adaptive NN

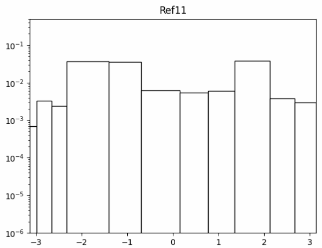
After Training



Before Training

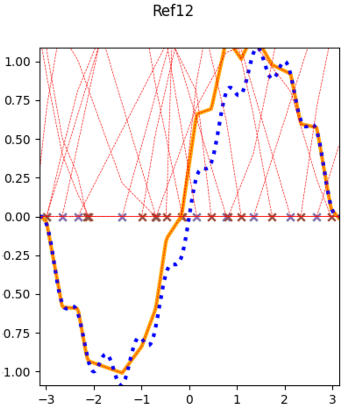


Error In Cells

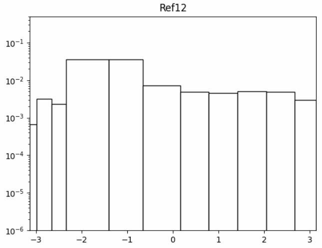


Example of adaptive NN

Before Training | After Training

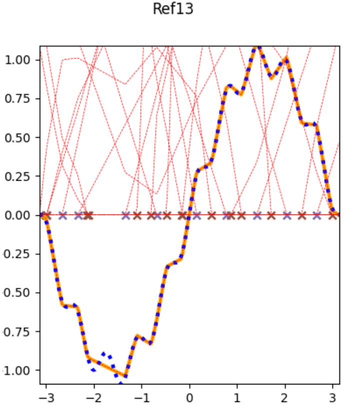


Error In Cells

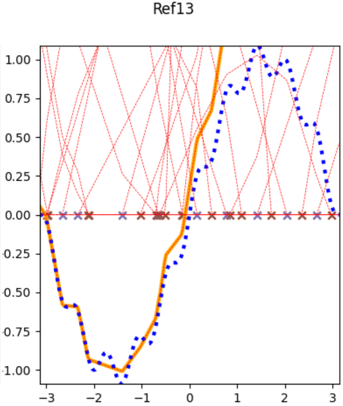


Example of adaptive NN

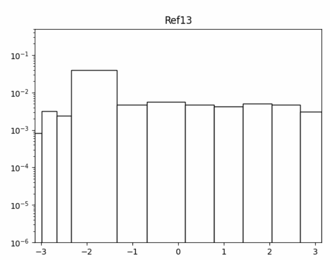
After Training



Before Training

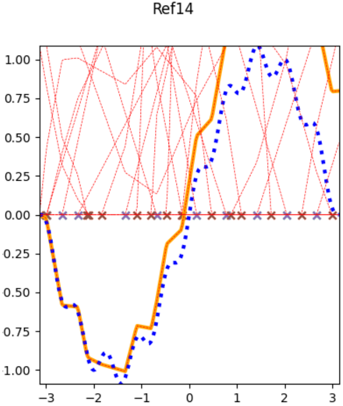


Error In Cells

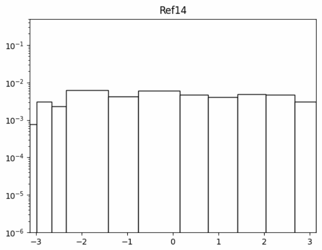


Example of adaptive NN

Before Training | After Training



Error In Cells



Adaptive NN details

Questions

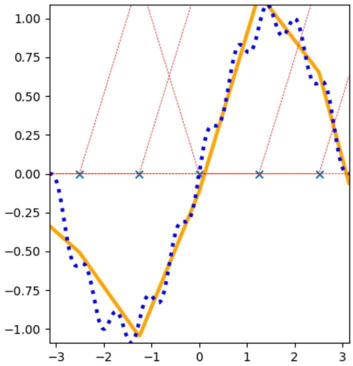
- How do we initialize all weights and biases at the beginning?

Answers: initialization at ref 0

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

- A^0, b^0 such that $A^0_i x_i + b^0_i = 0$ with x_i equispaced
- $|A^0_i| = 1$ 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

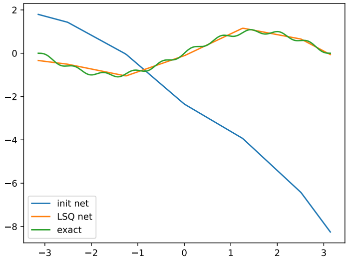
- How do we initialize all weights and biases at the beginning?

Answers: initialization at ref 0

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

- A^0, b^0 such that $A^0_i x_i + b^0_i = 0$ with x_i equispaced
- $|A^0_i| = 1$ 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

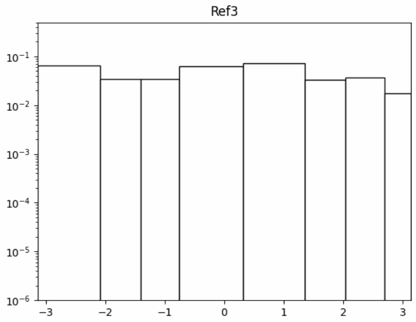
- How do we initialize the new weights and biases?
- When do we decide to add a layer or a node and how?

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

$$A^0_{\ell^1+1} x_{bary} + b^0_{\ell^1+1} = 0$$

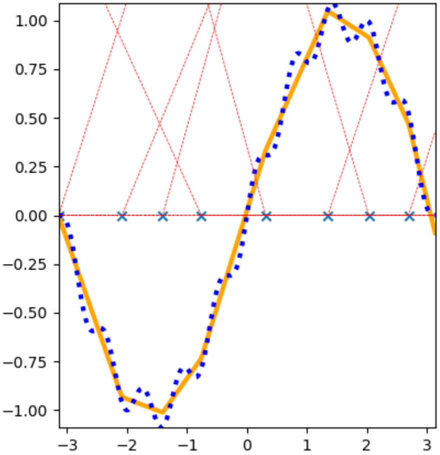
- Outer layer: $A^1_{\ell^1+1} = 0$



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

$$A^0_{\ell 1+1} x_{bary} + b^0_{\ell 1+1} = 0$$

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

Adaptive NN details: first hidden layer

Questions

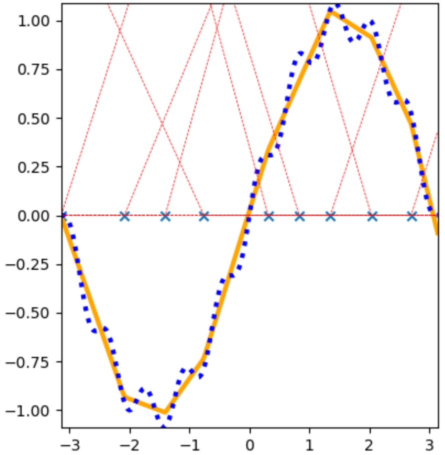
- How do we initialize the new weights and biases?
- When do we decide to add a layer or a node and how?

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

$$A^0_{\ell^1+1} x_{bary} + b^0_{\ell^1+1} = 0$$

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



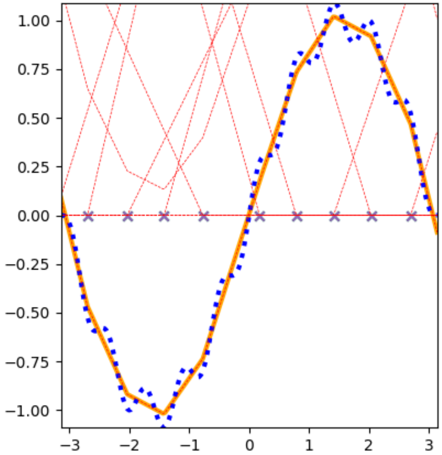
Adaptive NN details: second hidden layer

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: $A^2_1 = 1$ 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



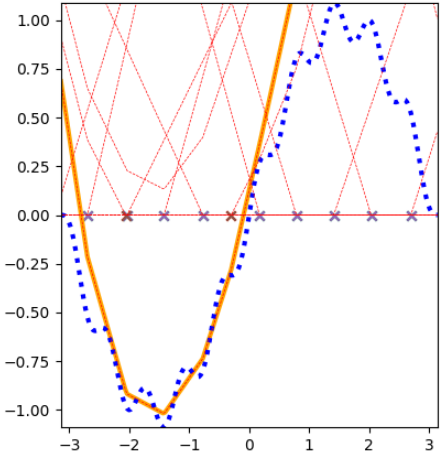
Adaptive NN details: second hidden layer

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: $A^2_1 = 1$ 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



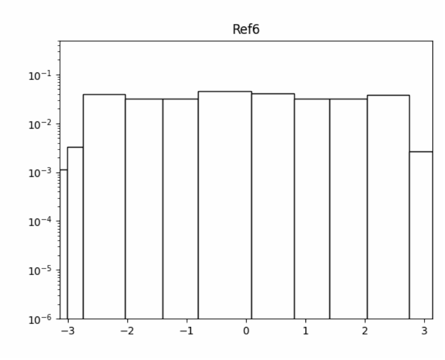
Adaptive NN details: second hidden layer

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: $A^2_1 = 1$ 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



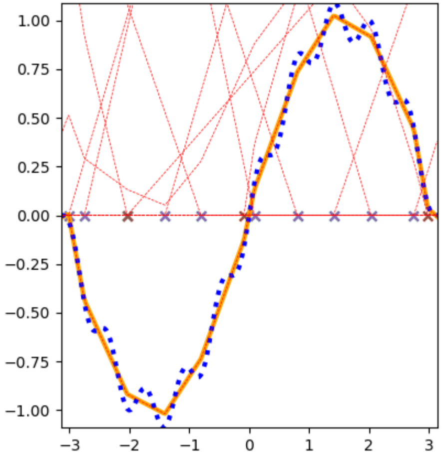
Adaptive NN details: second hidden layer

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: $A^2_1 = 1$ 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



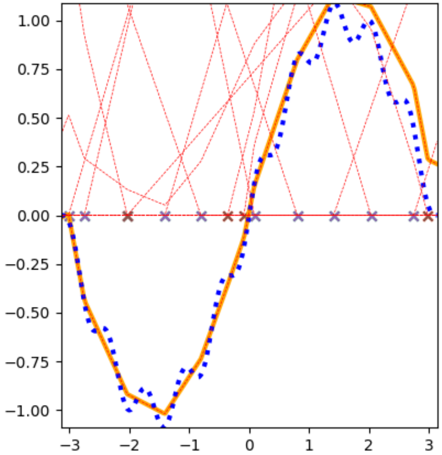
Adaptive NN details: second hidden layer

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: $A^2_1 = 1$ 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



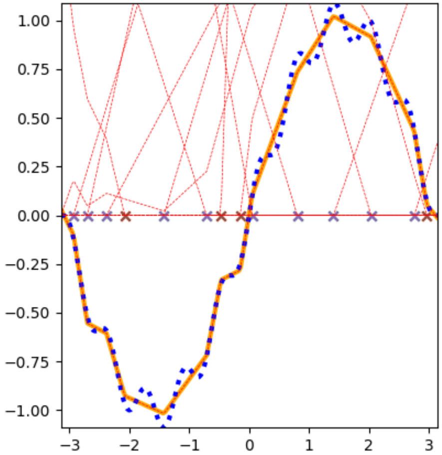
Adaptive NN details: second hidden layer

Questions

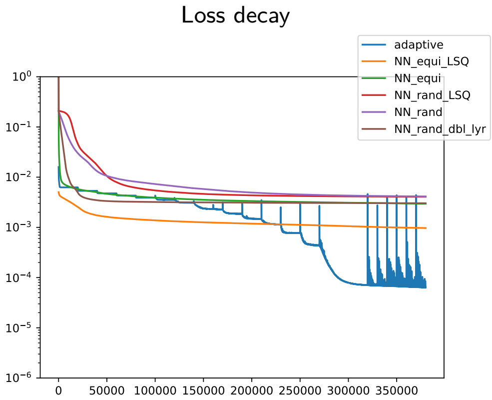
- How do we initialize the new weights and biases?
- When do we decide to add a layer or a node and how?

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: $A^2_1 = 1$ 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]



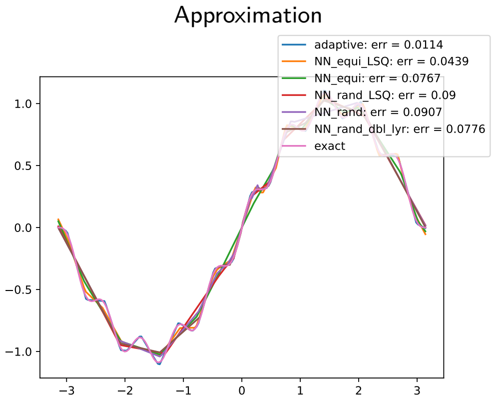


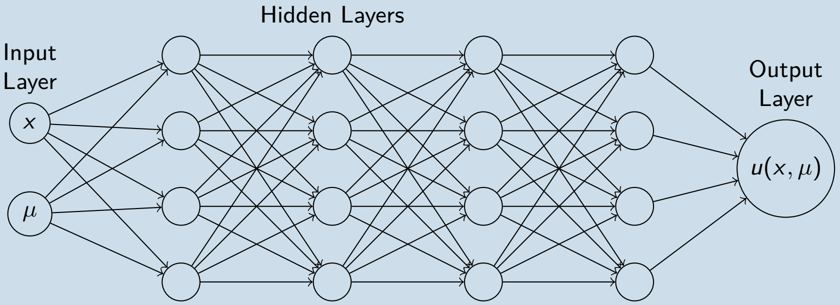
Table of contents

Parametric Problems and Reduced Order Modeling

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

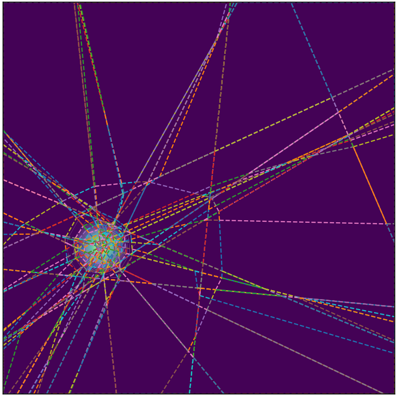
Adaptive NN and domain dimension

- 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

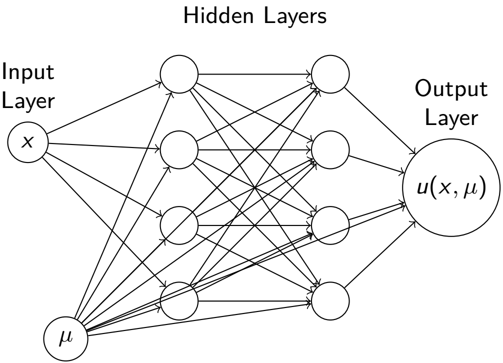
Solution

- Consider biases that are parameter dependent $b^i = b^i_0 \mu + b^i_1$
- Fix a reference parameter μ^* where to apply the error estimator

Network [2 14 22 1]



Adaptive NN ROM architecture

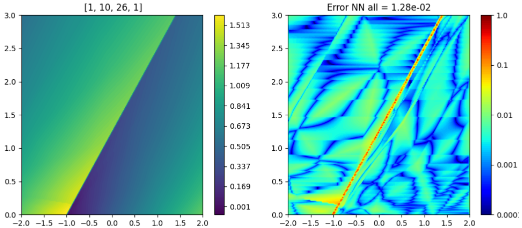


Training Strategy

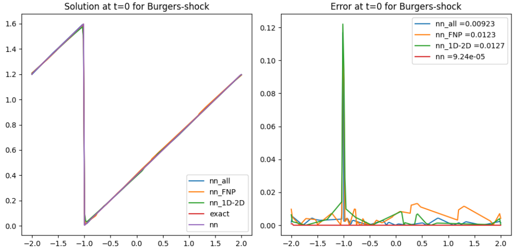
- Option **ALL**: train **from zero** with this architecture
- Option **FNP**: train first a non parametric for a μ^*, start from that architecture and train again with $b^i = b^i_0(\mu - \mu^*) + b^i_1$ initializing $b^i_0 = 0$ and $b^i_1 = b^i_{NP}$
- Option **1D-as-2D**: If in 1D, and 1 param, we can pretend it is 2D and train as in 2D

Burgers shock

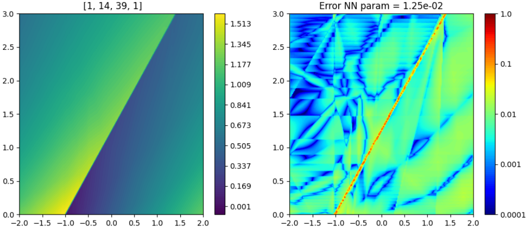
Adaptive ALL



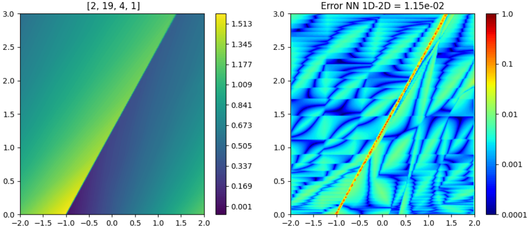
Comparison at initial time



Adaptive FNP



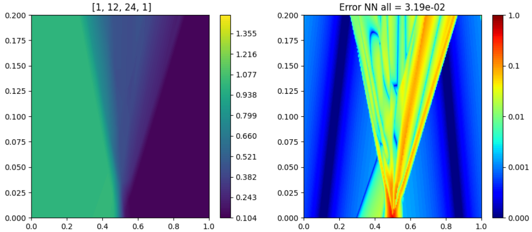
Adaptive 1D-as-2D



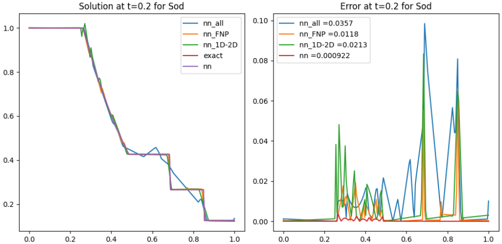
Burgers shock

Sod shock tube

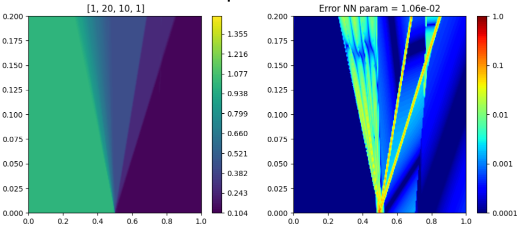
Adaptive ALL



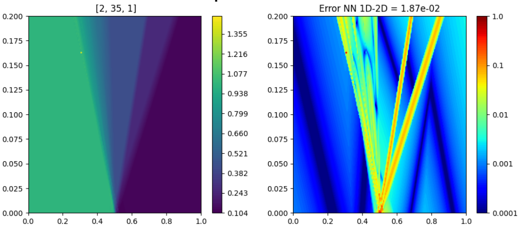
Comparison at final time



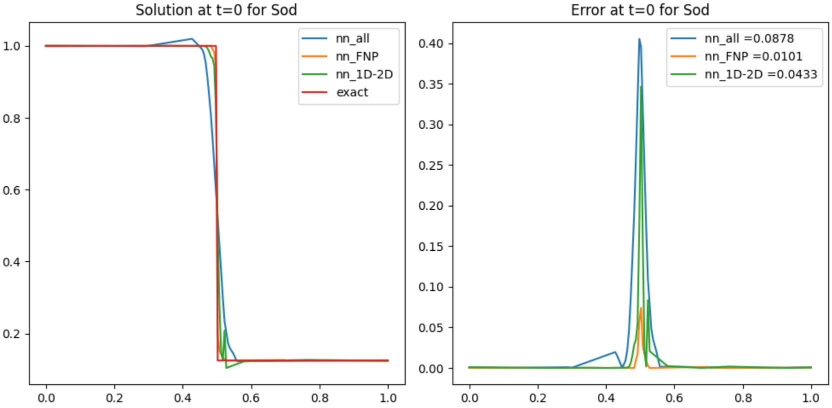
Adaptive FNP



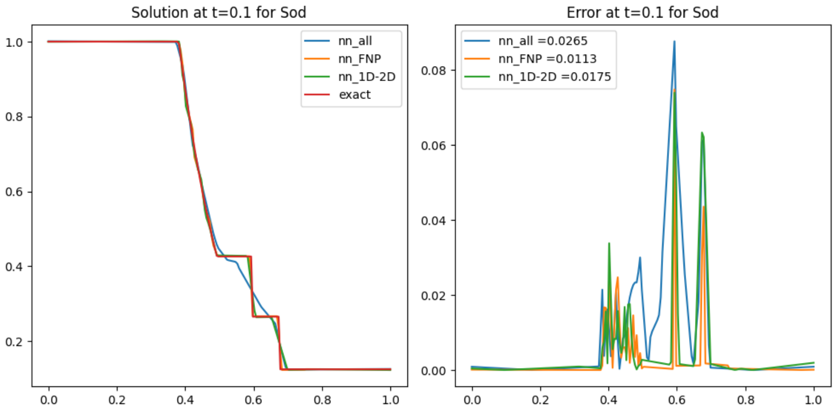
Adaptive 1D-as-2D



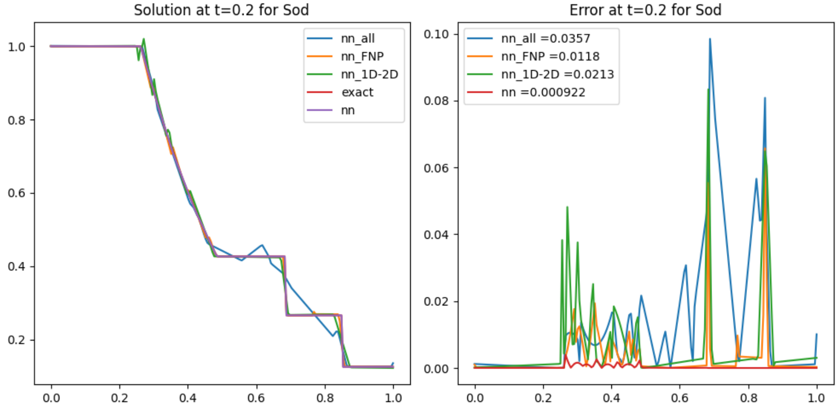
Sod shock tube



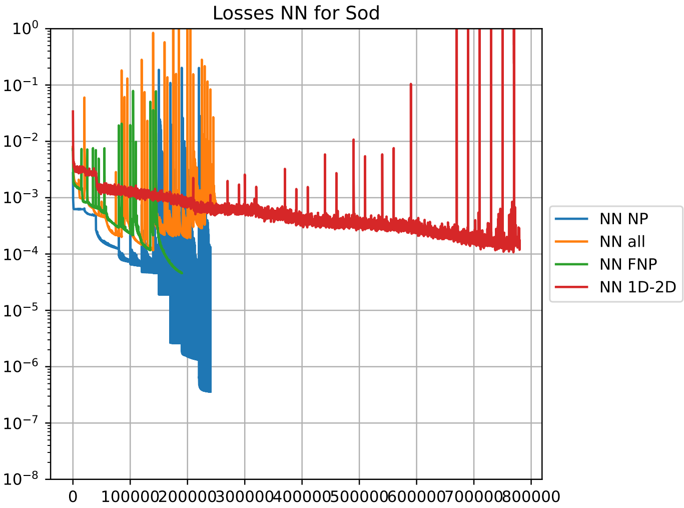
Sod shock tube



Sod shock tube

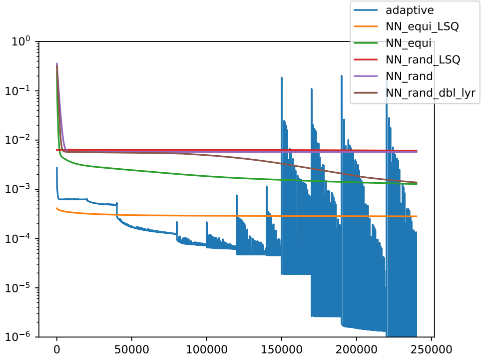


Sod shock tube

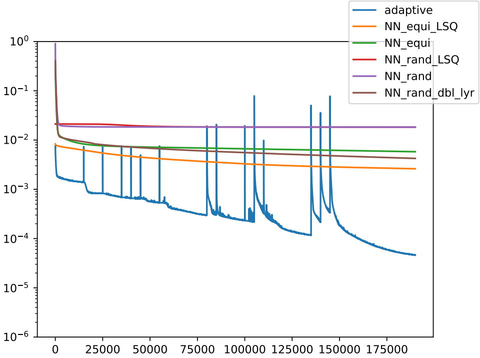


Sod shock tube (comparison with non adaptive)

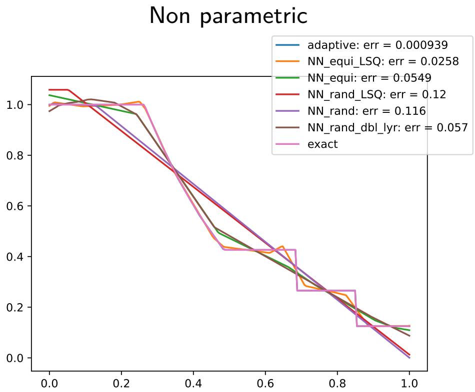
Non parametric

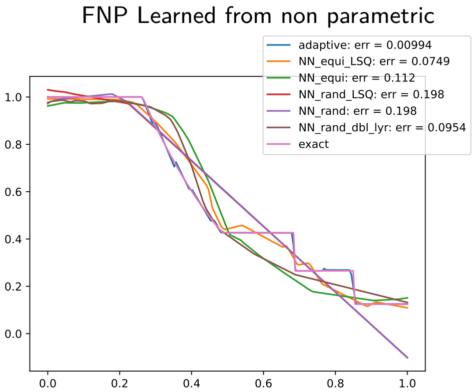


FNP Learned from non parametric



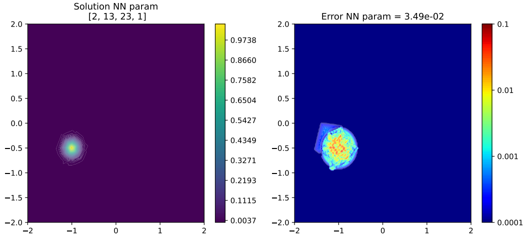
Sod shock tube (comparison with non adaptive)



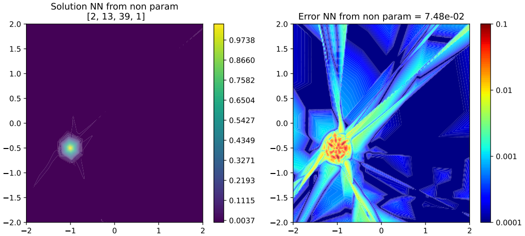


2D moving Gaussian

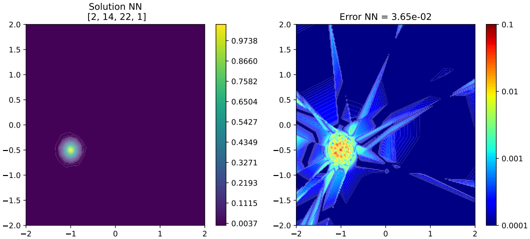
Adaptive ALL



Adaptive FNP



Adaptive non parametric

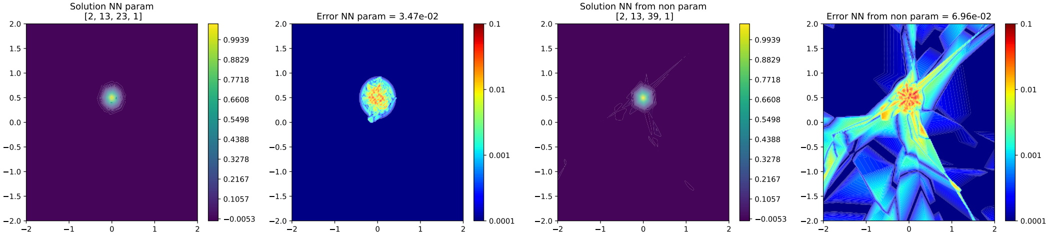


Initial time

2D moving Gaussian

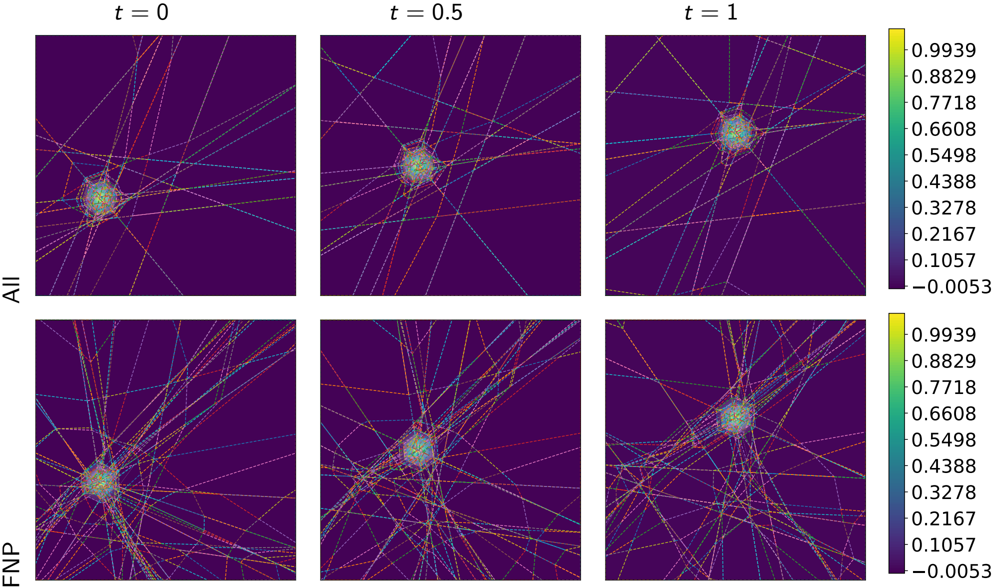
Adaptive ALL

Adaptive FNP

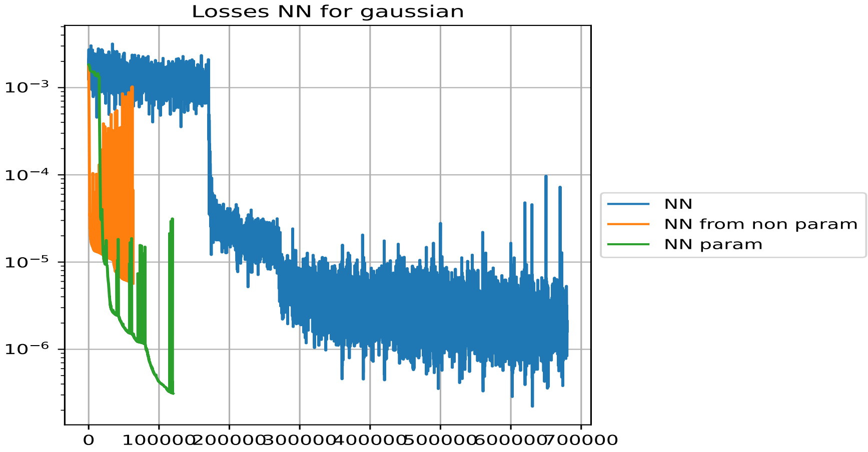


Final time

2D moving Gaussian

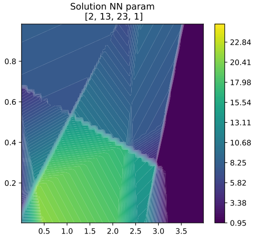


2D moving Gaussian

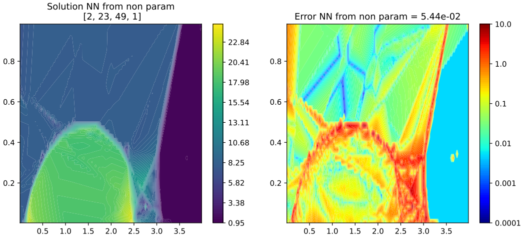


2D Double Mach Reflection for Euler's Equations

Adaptive ALL

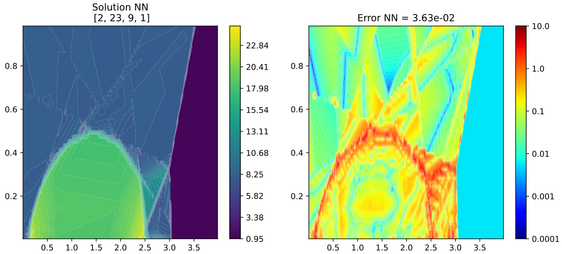


Adaptive FNP

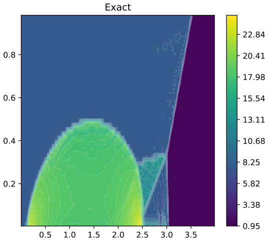


Adaptive non parametric

Final time

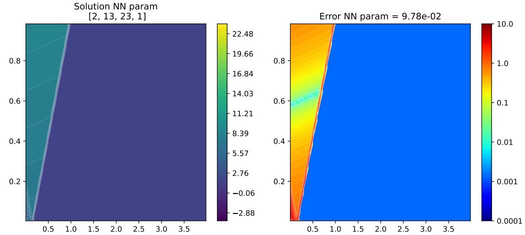


Exact

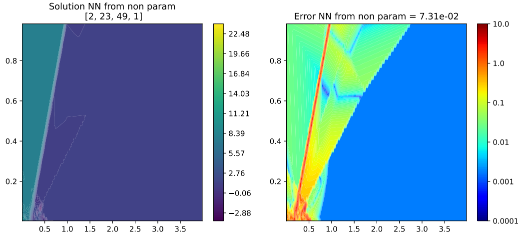


2D Double Mach Reflection for Euler's Equations

Adaptive ALL

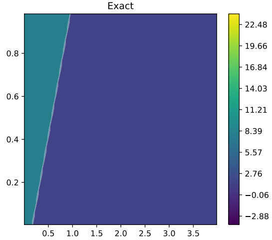


Adaptive FNP



Exact

Initial time



2D Double Mach Reflection for Euler's Equations

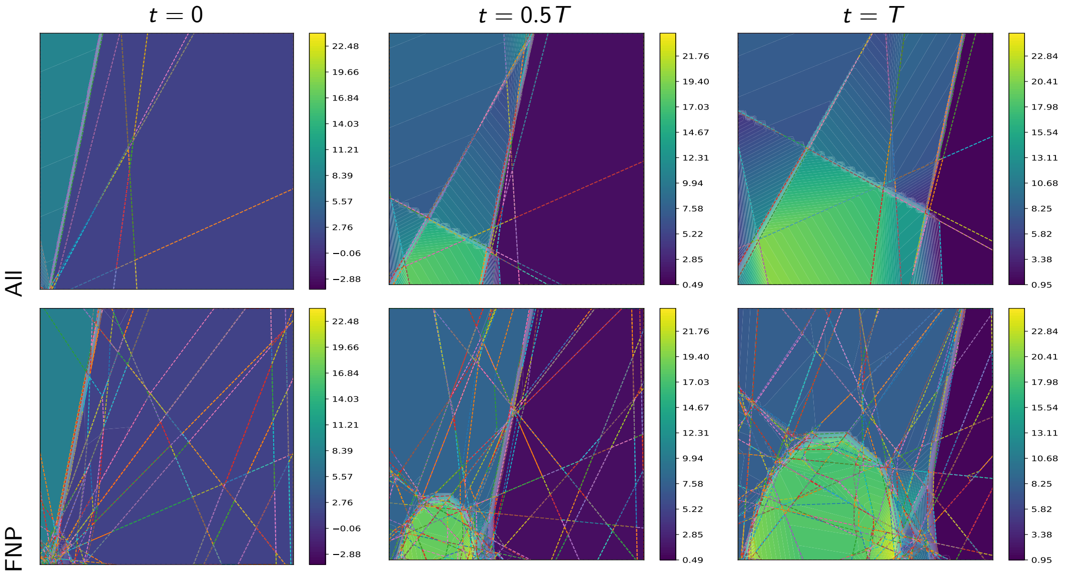


Table of contents

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

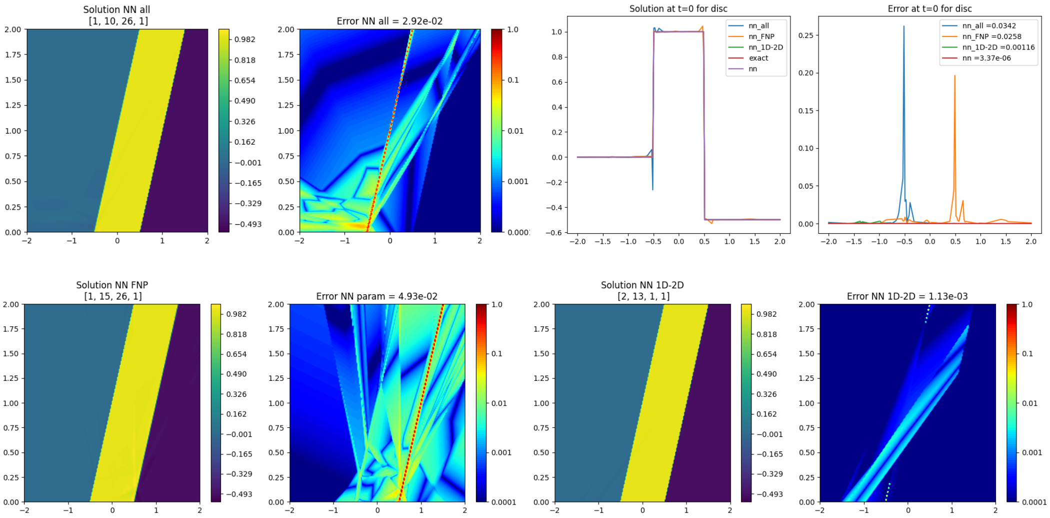
- Add nodes and layer iteratively
- Use ReLUs to define breaking lines to get mesh
- Error estimator for refining in the right position
- MOR with a given reference parameter

Perspectives

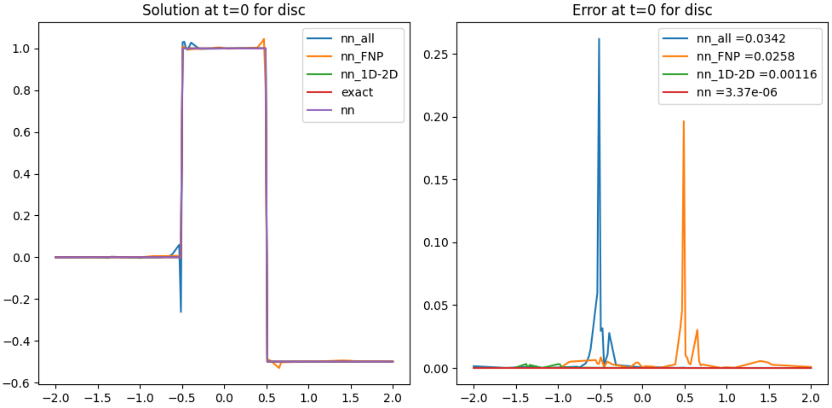
- Try different configurations for parameter dependent weights and biases
- Hyperparameter tuning (how many new nodes all together, maximum number of nodes etc)
- 3D

THANKS!!

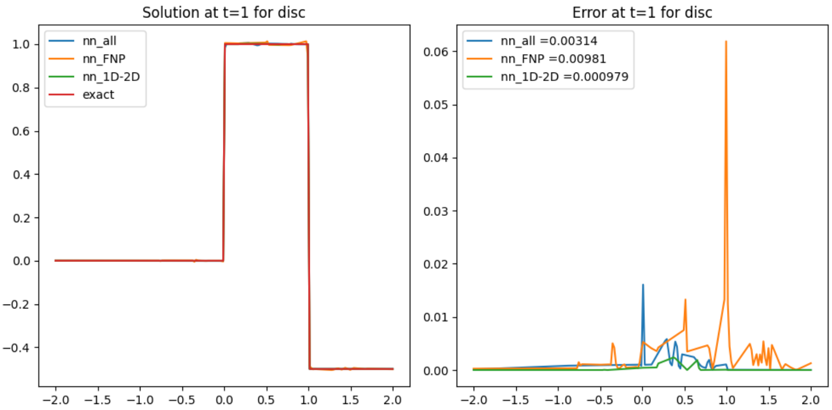
Moving discontinuity



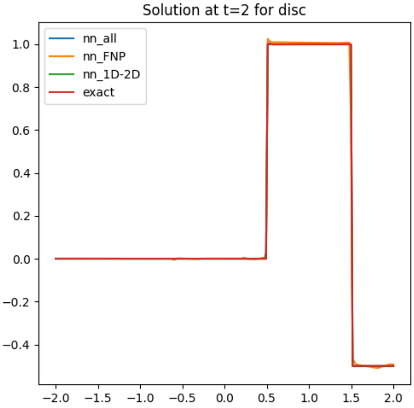
Moving discontinuity

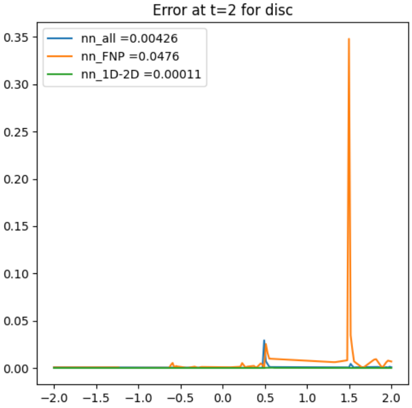


Moving discontinuity



Moving discontinuity

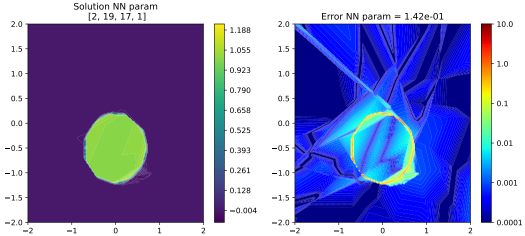




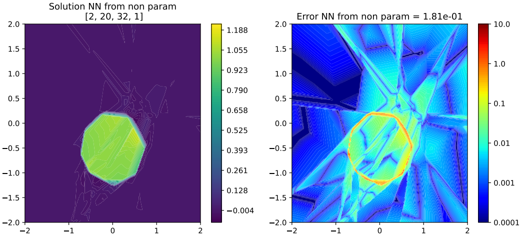
Moving discontinuity

2D moving circle

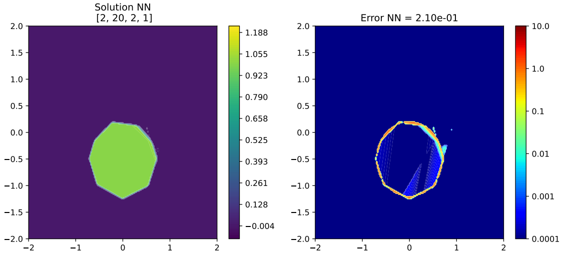
Adaptive ALL



Adaptive FNP



Adaptive non parametric

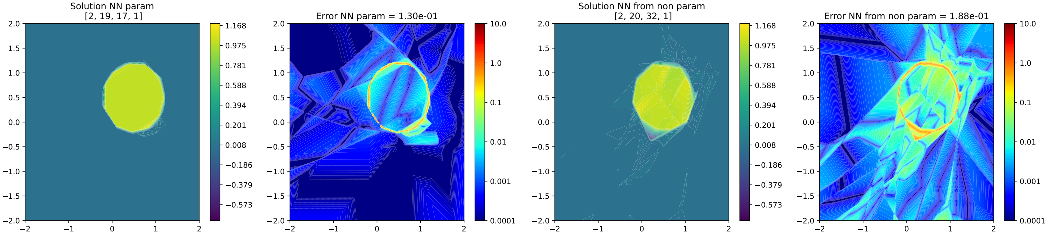


Initial time

2D moving circle

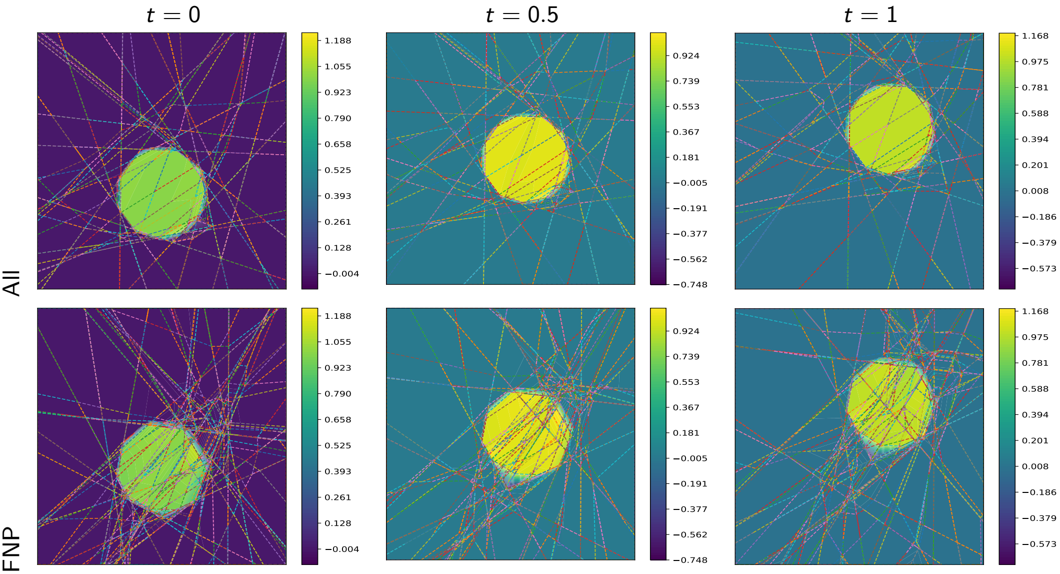
Adaptive ALL

Adaptive FNP

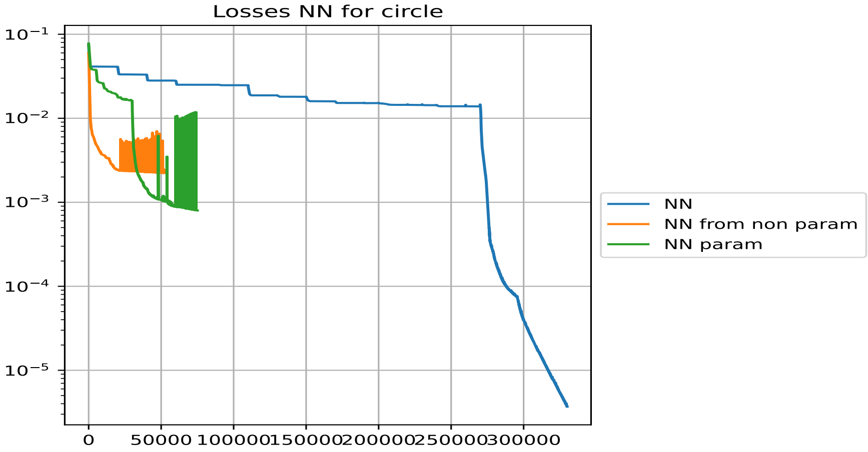


Final time

2D moving circle



2D moving circle



2D Double Mach Reflection for Euler's Equations