Physics-Informed Neural Networks: COMSOL Multiphysics® and MATLAB®

Martin Kožíšek
kozisek@humusoft.cz

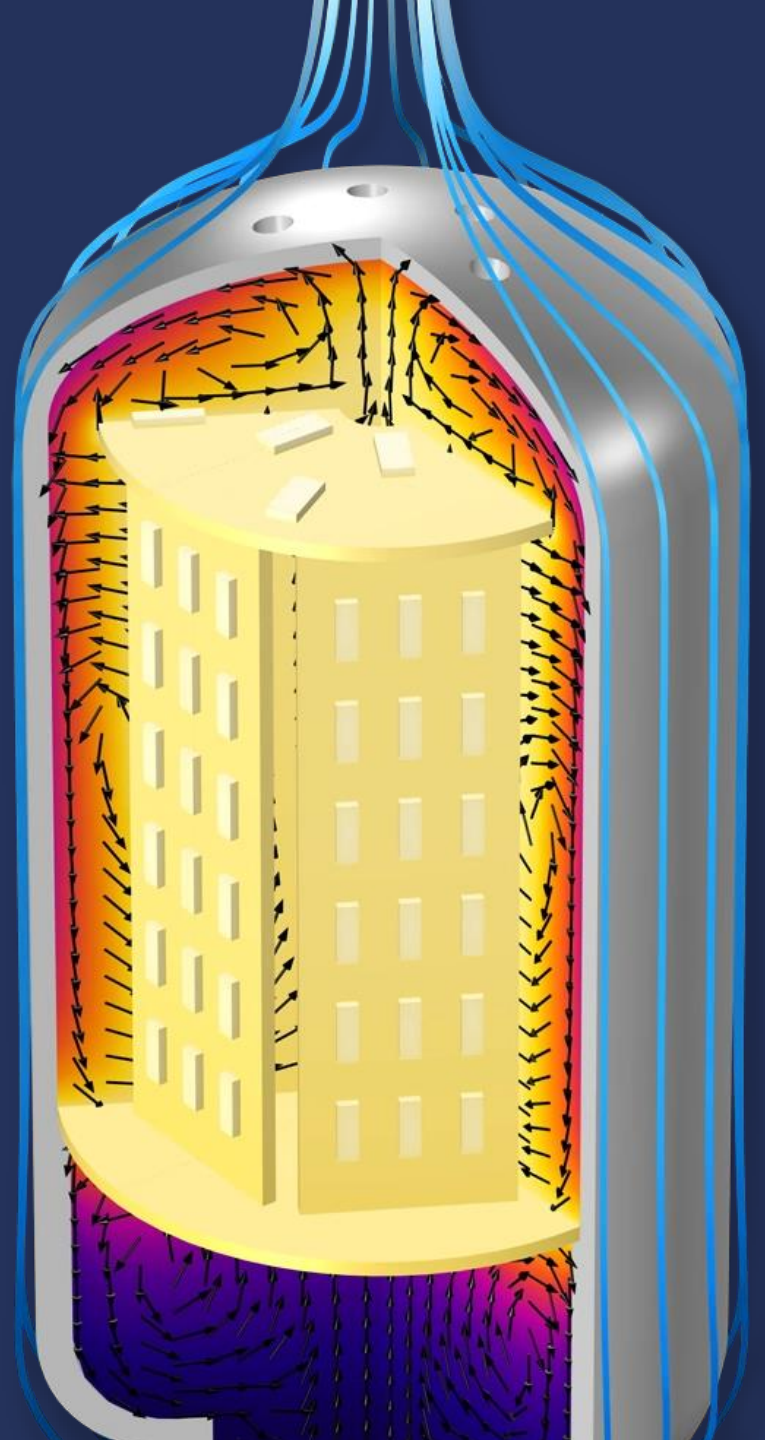
Jaroslav Jirkovský
jirkovsky@humusoft.cz

Schedule

1. Introduction to Surrogate Models in COMSOL
2. Deep Neural Networks in COMSOL
   - Workflow
   - DNN Architecture
   - One Neuron
   - Activation Function
   - Analytical Expression For a Deep Neural Network Model
   - Influence of Network Architecture and Activation Function
3. Learning and Validation Process
4. Example in COMSOL
5. More Realistic Example in COMSOL
6. Introduction Neural Networks in MATLAB
7. Physics-Informed Neural Networks
8. Example in MATLAB

Introduction to Surrogate Models in COMSOL

Martin Kožíšek
kozisek@humusoft.cz
www.linkedin.com/in/martinkozisek/

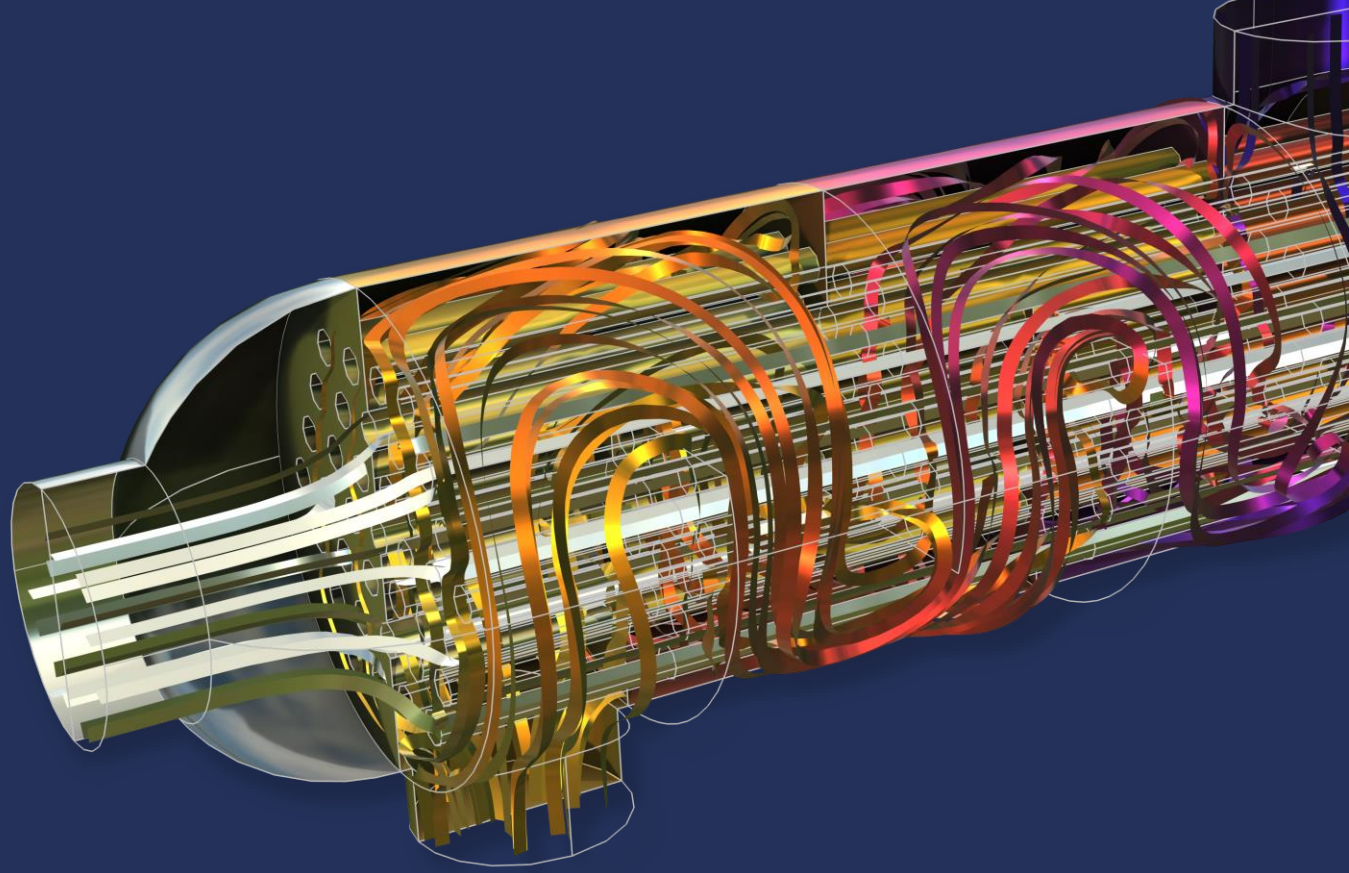


What Are Surrogate Models?

A simplified model that replaces a complex computational model (e.g., a simulation) without significant loss of accuracy.

Three (or Four?) Types of Surrogate Models in COMSOL Multiphysics

1. **DNN Based (Deep Neural Network)**
   Capturing complex, nonlinear relationships in large sets of data (Digital Twins), included in COMSOL Multiphysics® without add-on products
2. **GP Based (Gaussian Process)**
   A probabilistic framework that provides both predictions and uncertainty (Opitmization and Uncertainty Quantification), can handle only thousands of data tables, requires add-on modules.
3. **PCE Based (Polynomial Chaos Expansion)**
   Representing the influence of random variables (Sensitivity Analysis), can handle only thousands of data tables, requires add-on modules.
4. (Other Methods: Model Order Reduction)

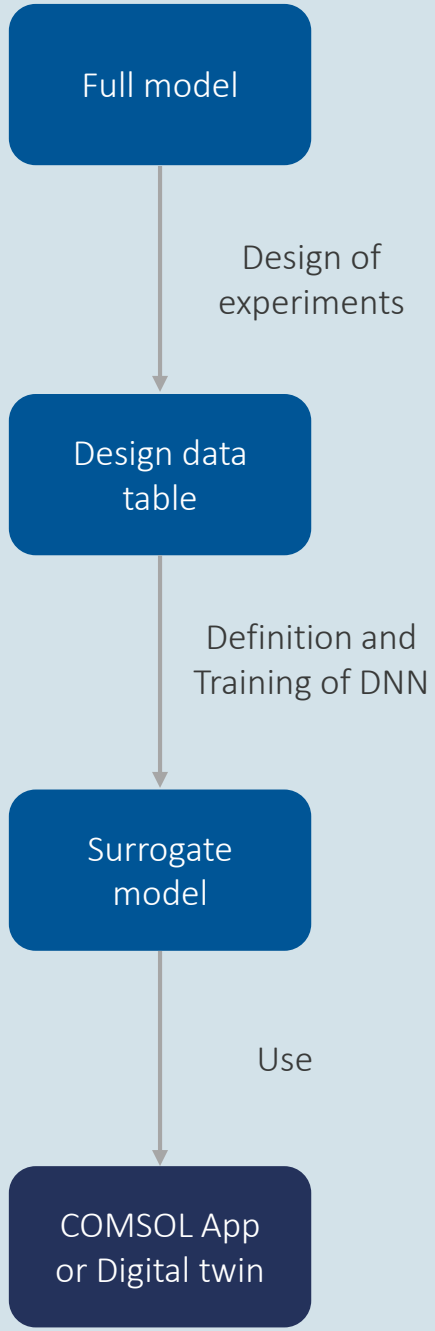


Deep Neural Networks in COMSOL Multiphysics

Martin Kožíšek
kozisek@humusoft.cz
www.linkedin.com/in/martinkozisek/

Workflow

- Setting Up a full simulation model
- Computing for a „some“ range of parameters (Design of Experiments)
  - Manual range (parametric sweep)
  - Latin hypercube sampling (covers the space of parameters)
- Definition and training in a new function type: Deep Neural Network
- Surrogate model: multidimensional function approximation
- Calling the Deep Neural Network function, e.g. in App or in Digital Twin

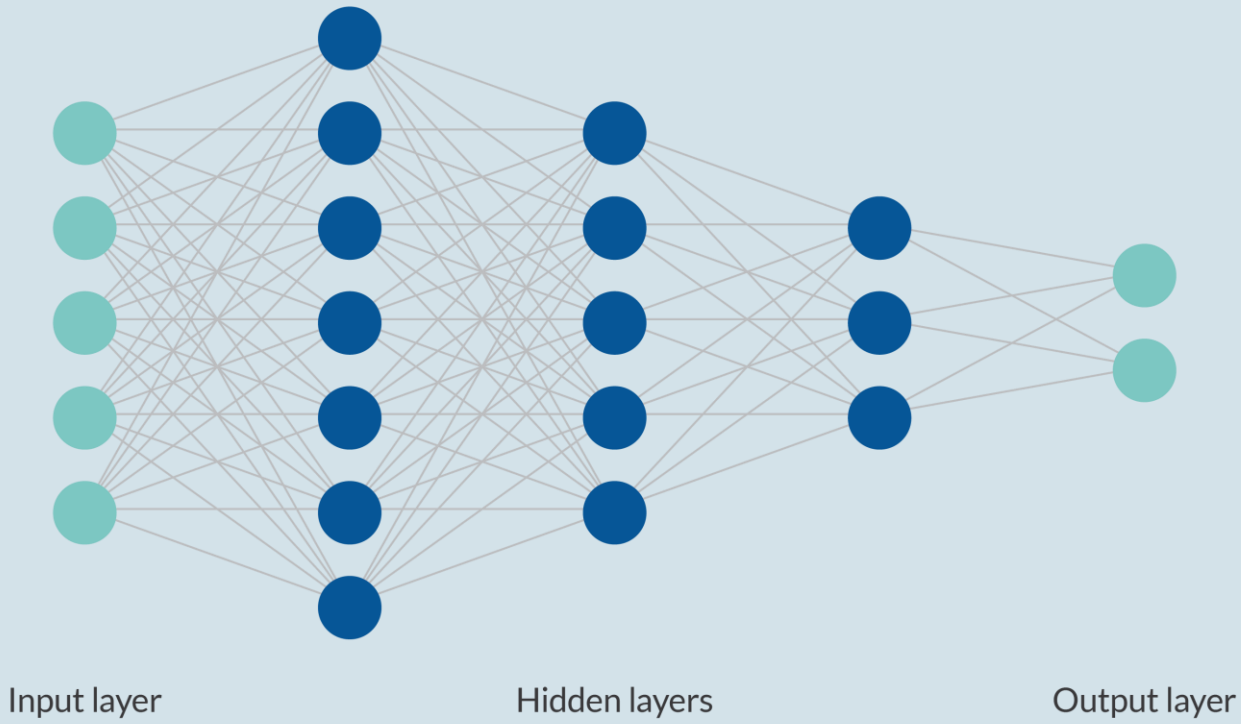


DNN Architecture

What does DNN look like, and how do you design it?

- Input layer: number of inputs
- A series of hidden layers: iterative process, knowledge, empirical testing, trial and error
- An output layer: number of outputs

*Notes: Each layer consists of a number of neurons. Too few layers/neurons may lead to **underfitting**. Excess layers/neurons can cause **overfitting**.*



The figure shows a graph for a network with three hidden layers, five input nodes, and two output nodes. COMSOL Multiphysics uses Dense feed-forward networks (Dense = every neuron is connected to every neuron in the adjacent layer, Feed-Forward = data flows in one direction from input to output).

The Neuron

Each neuron responds to n inputs by returning a scalar output.

$$x_j = f\left(b_j + \sum_{i=1}^{n} x_i w_i\right)$$

Output

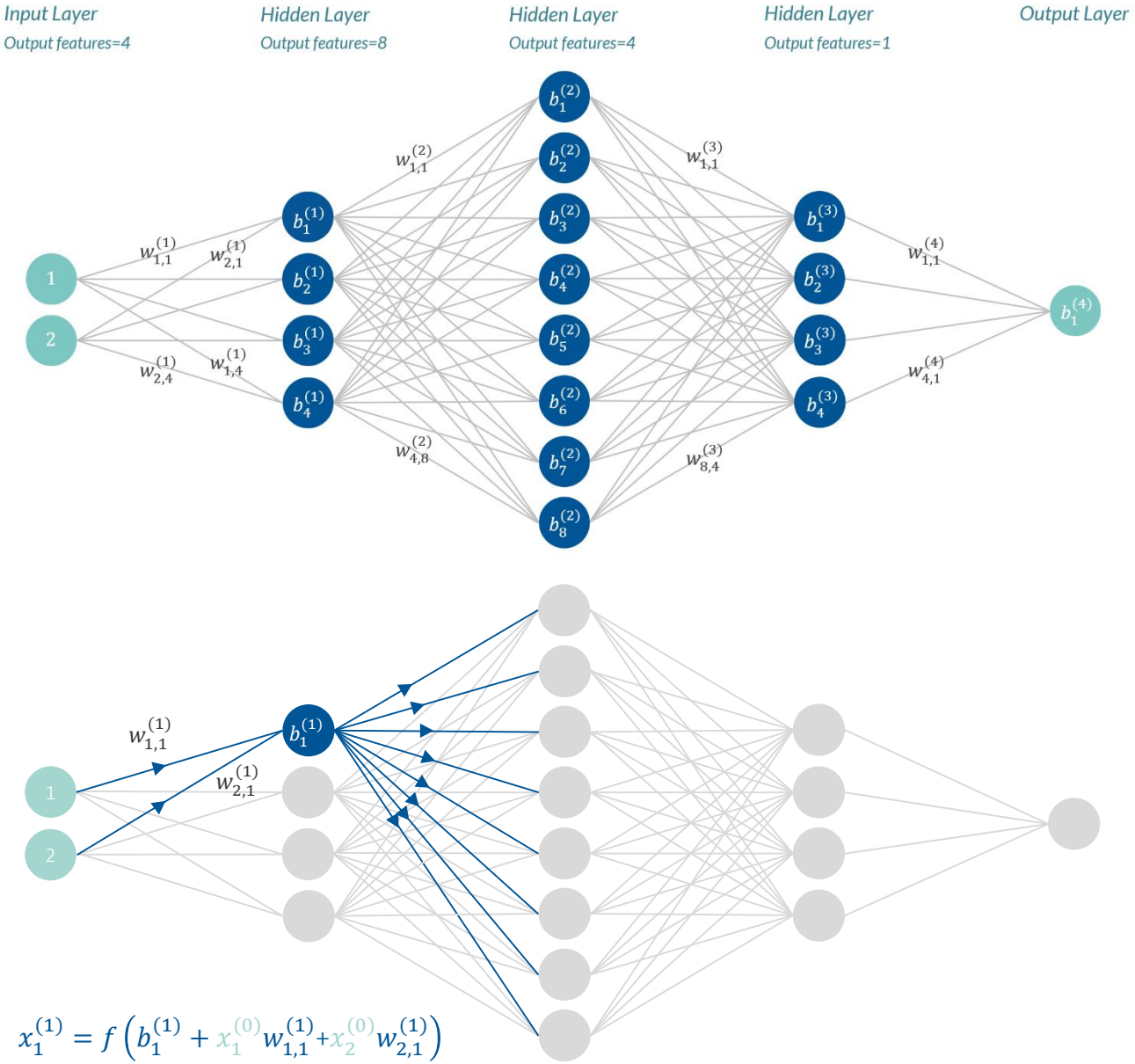
Activation function

Bias

Input i

Weight i

Note: DNN Training is searching for optimal values of Biases and Weights

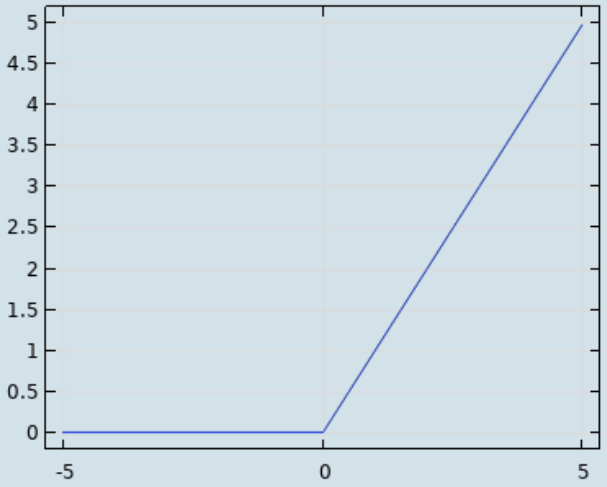


Activation Functions

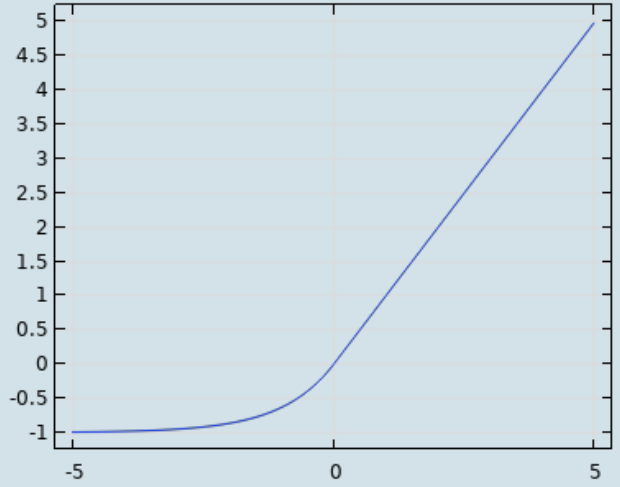
Activation Functions in COMSOL Multiphysics 6.3:

- **Linear**: no activation, just a weighted sum. Outputs: (–∞, ∞)
- **ReLU**: returns *max(0, x)* zero or linear for x > 0. Outputs: [0, ∞),
- **ELU**: returns x for x > 0, or e^x pro x ≤ 0. It is smooth ReLU. Outputs: [0, ∞).
- **Sigmoid**: $\frac{1}{1+e^{-x}}$ with outputs: (0, 1)
- **Tanh**: $\frac{e^x-e^{-x}}{e^x+e^{-x}}$ with outputs: (–1, 1)

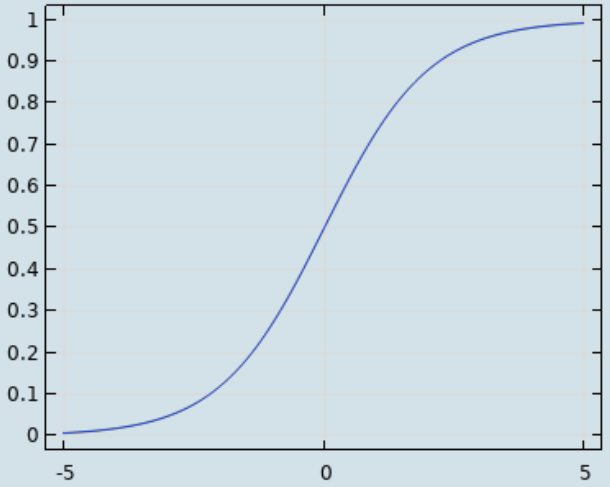
Rectified Linear Unit (ReLU)



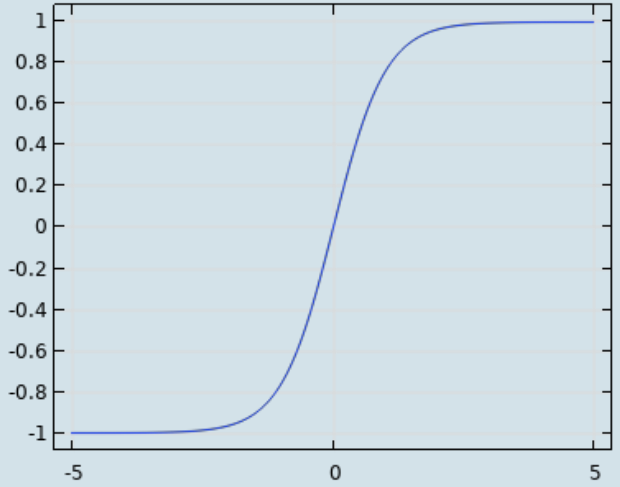
Exponential Linear Unit (ELU)



Sigmoid



Hyperbolic Tangent



Analytical Expression for DNN Example

Thermal Microactuator is a MEMS device that converts an electric input signal into mechanical motion.

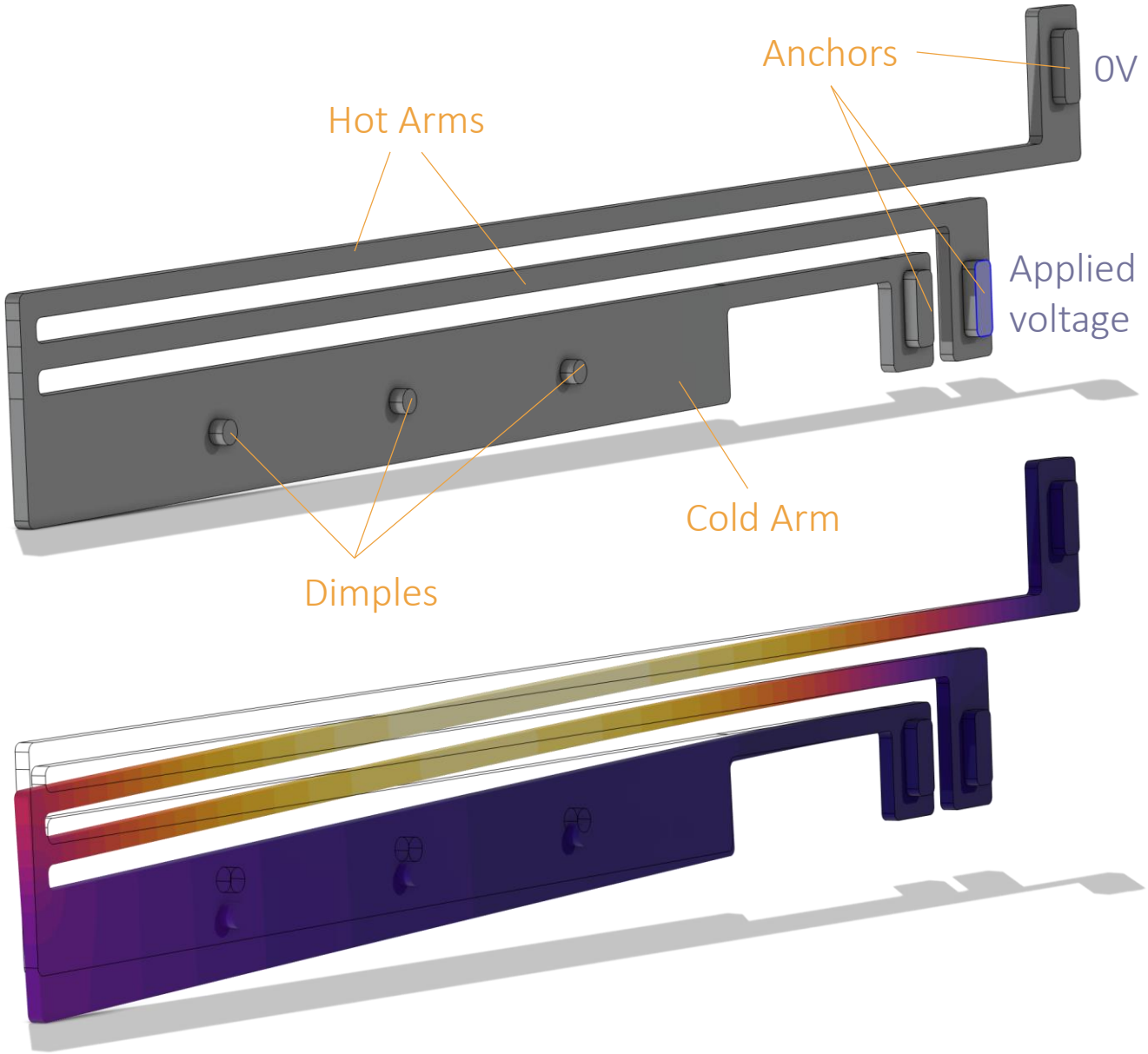
Simulation model settings:

- Electric current conduction,
- Heat conduction with Joule heating
- Thermal expansion (Stress and Strain)

Parameters:

- Actuator length
- Applied voltage

Model computes maximum displacement



Analytical Expression for DNN Example

Thermal Microactuator is a MEMS device that converts an electric input signal into mechanical motion.

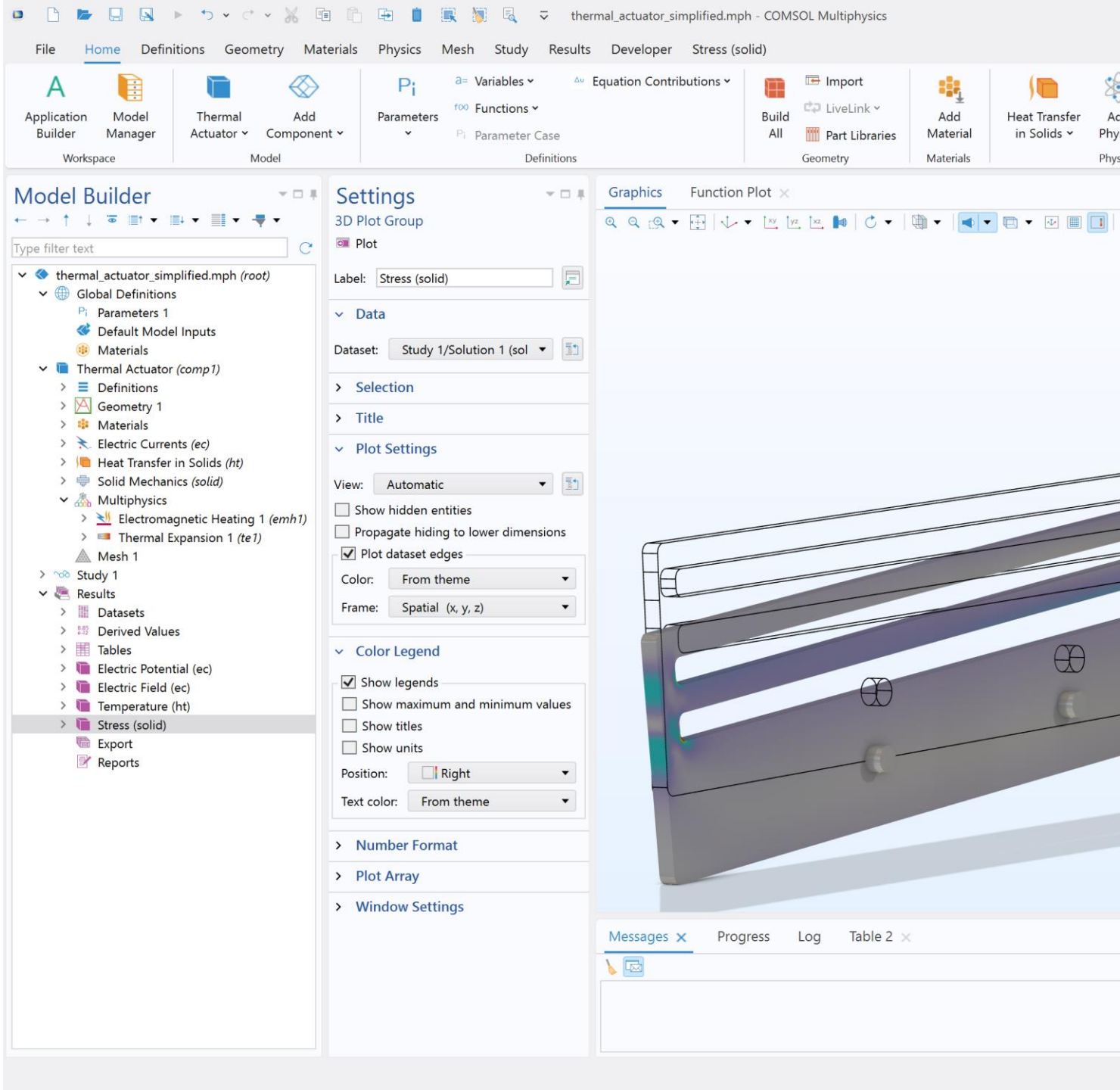
Simulation model settings:

- Electric current conduction,
- Heat conduction with Joule heating
- Thermal expansion (Stress and Strain)

Parameters:

- Actuator length
- Applied voltage

Model computes maximum displacement



Analytical Expression for DNN Example

Thermal Microactuator is a MEMS device that converts an electric input signal into mechanical motion.

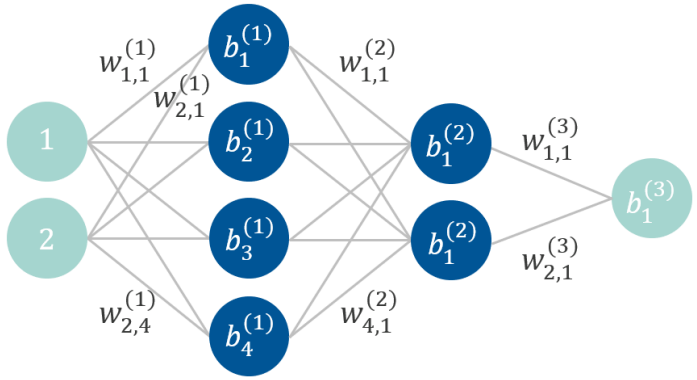
Simulation model settings:

- Electric current conduction,
- Heat conduction with Joule heating
- Thermal expansion (Stress and Strain)

Parameters:

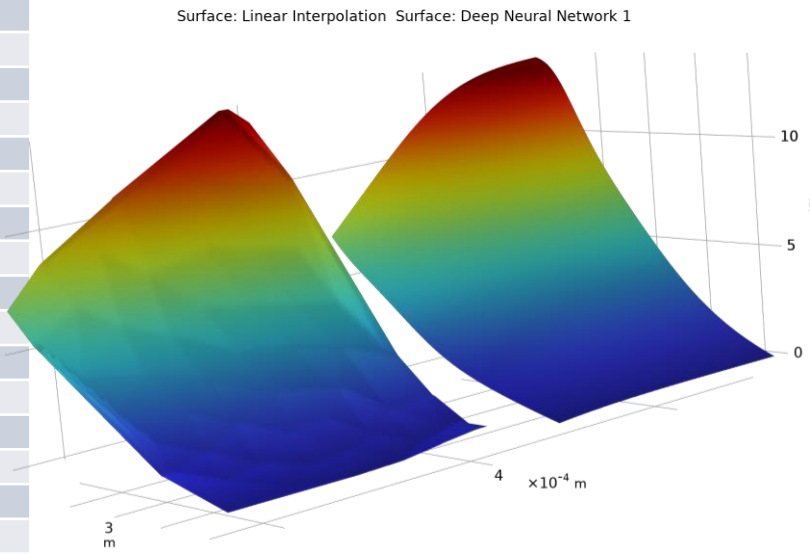
- Actuator length
- Applied voltage

Model computes maximum displacement



dnn(x1,x2) =
tanh(w3_11(tanh(w2_11*(tanh(w1_11*(x1)+w1_21*(x2)+b1_1))+w2_21*(tanh(w1_12*(x1)+w1_22*(x2)+b1_2))+w2_31*(tanh(w1_13*(x1)+w1_23*(x2)+b1_3))+w2_41*(tanh(w1_14*(x1)+w1_24*(x2)+b1_4))+b2_1))+w3_21*(tanh(w2_12*(tanh(w1_11*(x1)+w1_21*(x2)+b1_1))+w2_22*(tanh(w1_12*(x1)+w1_22*(x2)+b1_2))+w2_32*(tanh(w1_13*(x1)+w1_23*(x2)+b1_3))+w2_42*(tanh(w1_14*(x1)+w1_24*(x2)+b1_4))+b2_2))+b3_1)*

w1_11	w1_12	w1_13	w1_14
6.479052	-83.5153	-740.349	1028.273
w1_21	**w1_22**	**w1_23**	**w1_24**
-0.01446	-2.50149	0.217145	-0.23627
b1_1	**b1_2**	**b1_3**	**b1_4**
-0.03275	9.940952	-2.92705	2.314871
w2_11	**w2_12**	**w2_21**	**w2_22**
1.882123	131.0688	0.004636	-0.17448
w2_31	**w2_32**	**w2_41**	**w2_42**
-235.551	-211.39	-61.658	-81.0836
b2_1	**b2_2**		
-175.955	-115.824		
w3_11	**w3_21**		
-141.037	-148.391		
b3_1			
8.237812			

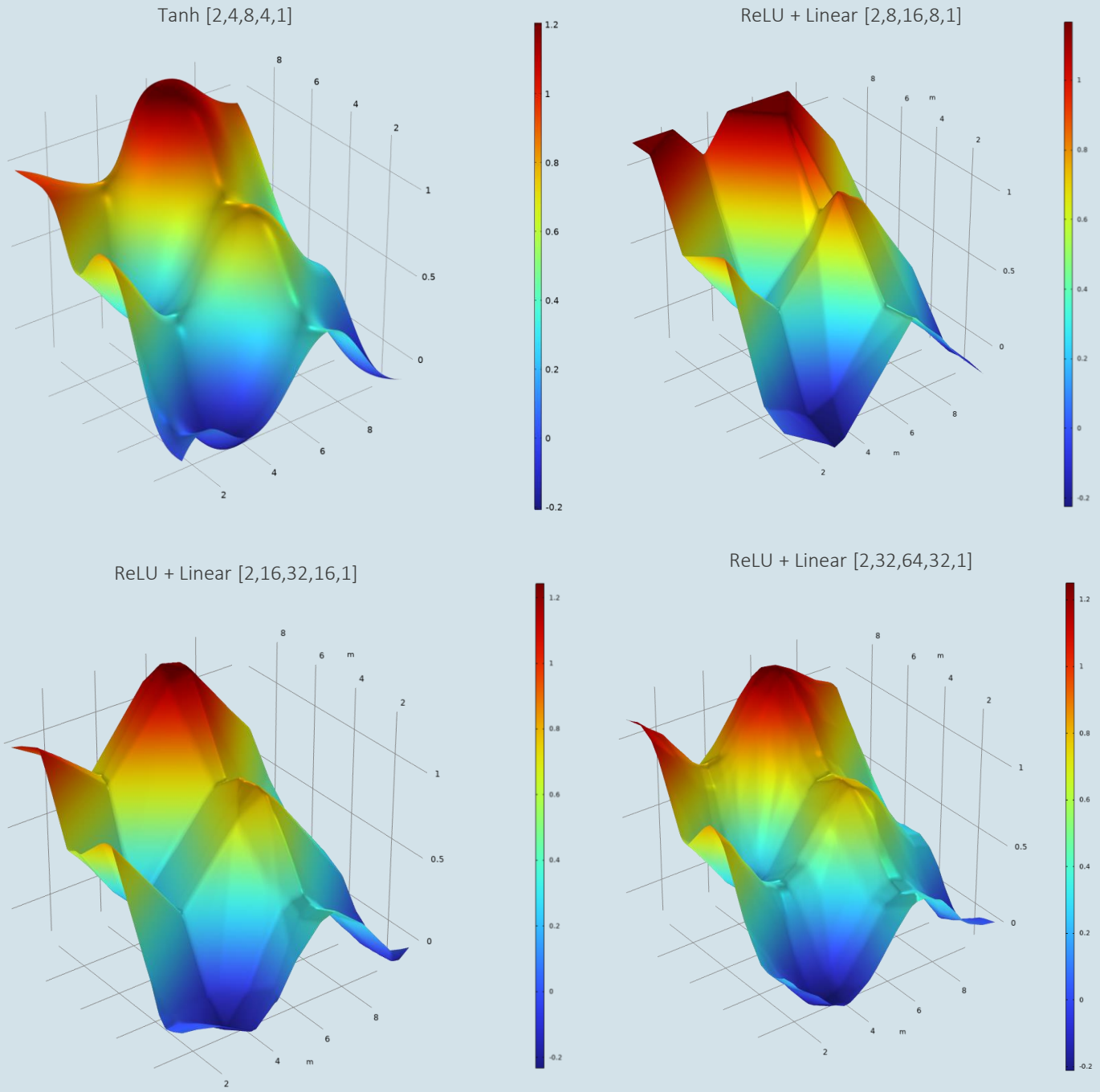


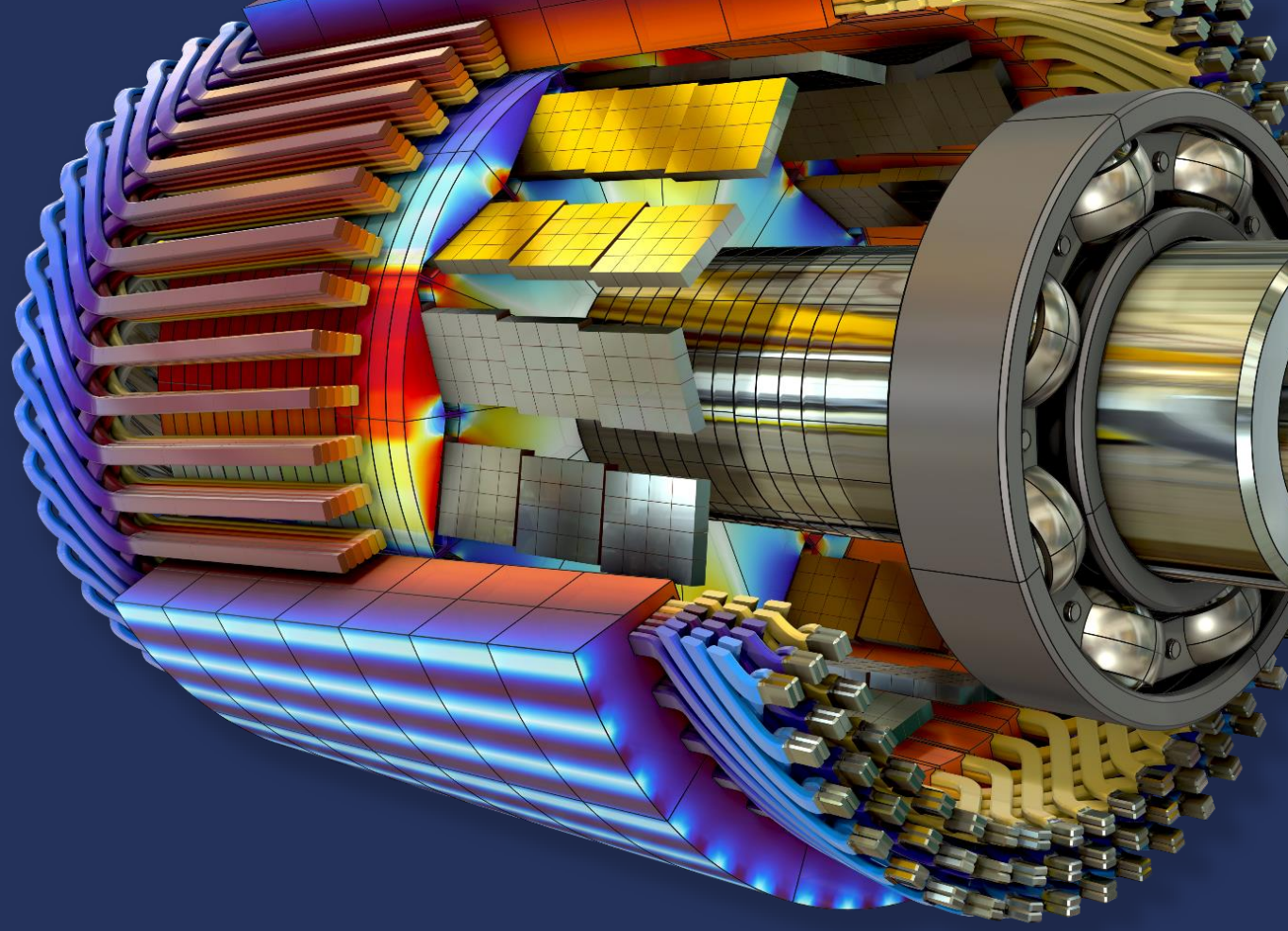
The Table shows optimized values of weights and biases. The data needs to be scaled before the network can be optimized, since the network uses the tanh activation function, and the output values of tanh are limited to (-1, 1).

Influence of Network Architecture and Activation Function

For regression tasks, as an alternative to using the tanh activation function, a combination of the ReLU and Linear activation functions can be used. Comparision:

- Tanh [2,4,8,4,1]
- ReLU + Linear [2,4,8,4,1] – Loss 0.2
- ReLU + Linear [2,8,16,8,1]
- ReLU + Linear [2,16,32,16,1]
- ReLU + Linear [2,32,64,32,1]



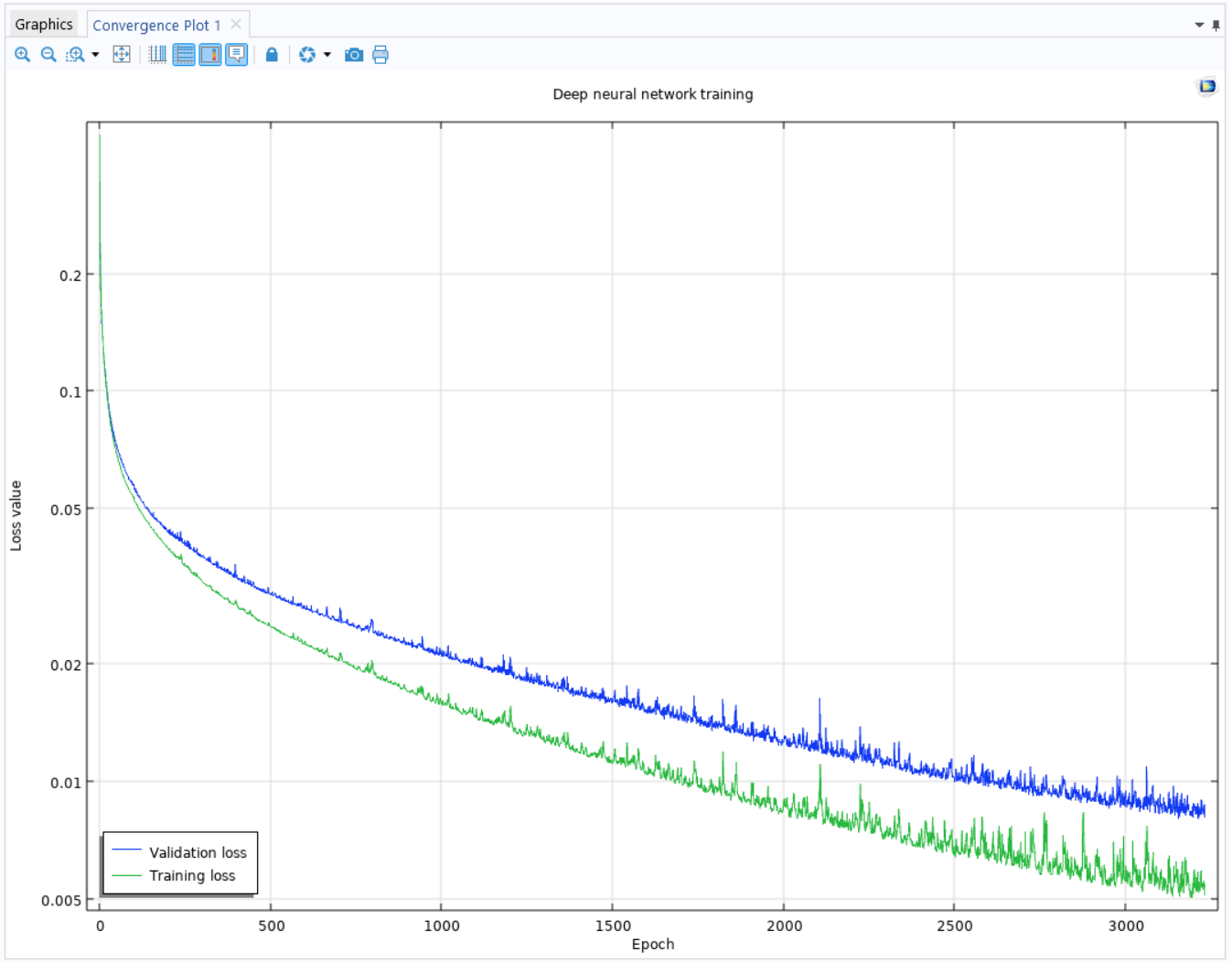


Training and Validation Process

Martin Kožíšek
kozisek@humusoft.cz
www.linkedin.com/in/martinkozisek/

Training a DNN

- Training involves optimizing weights and biases to minimize error.
- Objective of training: Align the surrogate model closely with the finite element model.
- COMSOL uses the ADAM (Adaptive Moment Estimation) algorithm, which evaluates the first (mean) and second (variance) moments of the gradient of the loss function to optimize weights and biases.
- Error measurement is calculated via the loss function.
- Different types of loss functions can be used.
- The default loss function is *Root-mean-square error* (RMSE).



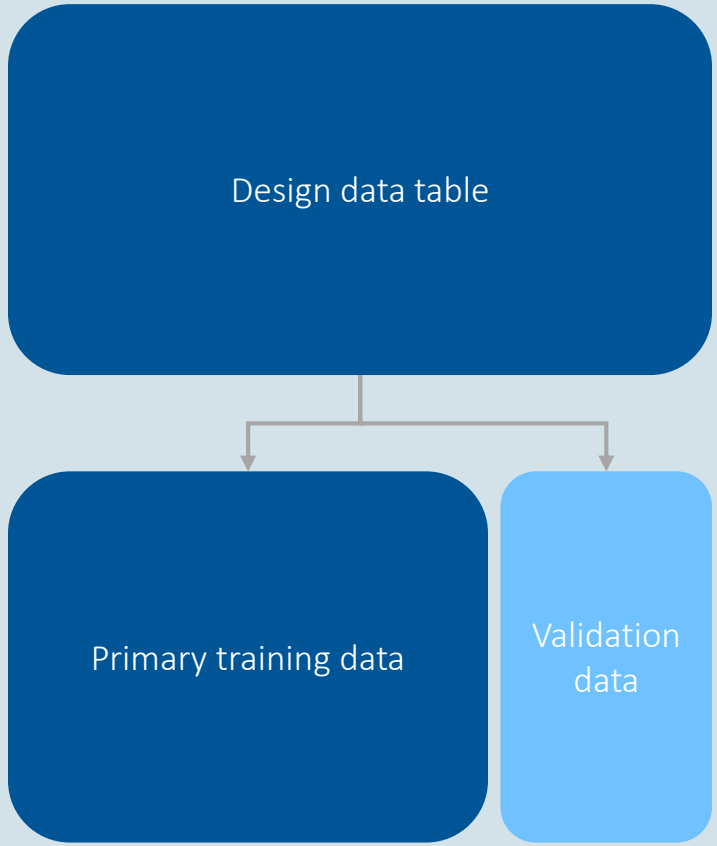
ADAM algorithm is adaptive stochastic gradient descent method (SGD). The stochastic method randomly selects one small mini-batch of data ata time, to perform an update on the network parameters. The randomness in selecting data points introduces variability in the gradient estimates, which helps escaping local minima in the loss landscape, potentially leading to better generalization on unseen data. As a result, the updates to the network parameters are noisy, as can be seen from the convergence plots.

Training and Validation Loss

Training data are split into Primary training data and Validation data.

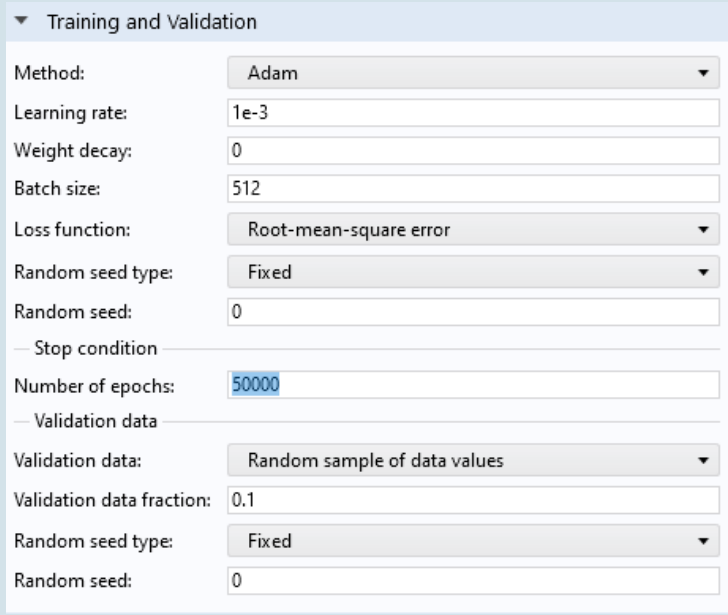
- *Training loss*:
  - Reflects model performance on the primary training data.
- *Validation loss*:
  - Indicates model performance on a separate, unseen subset of data. Why?
    If we train and test the model on the same data, it can learn the specifics of that data, leading to poor generalization.

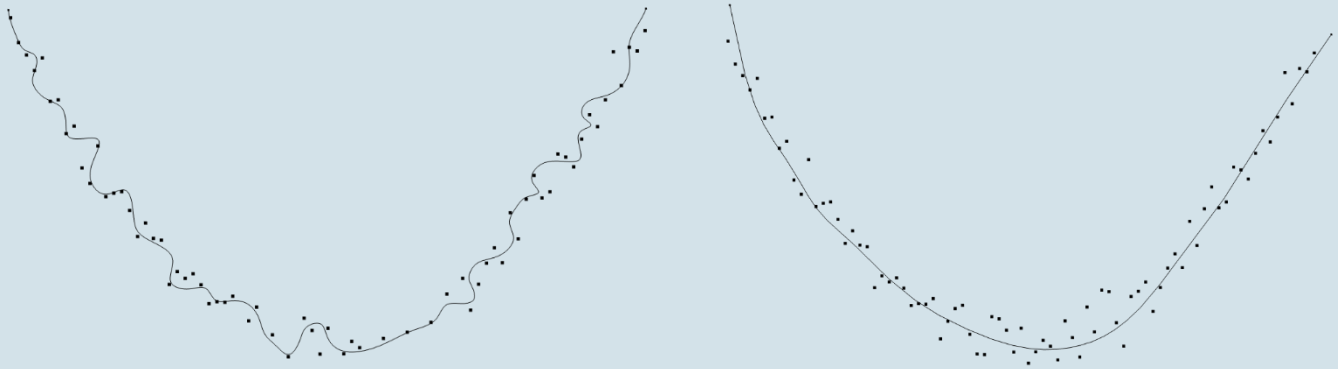
Note: When I was a child, I loved solving equations. Every math book had a few step-by-step examples, and I learned from these examples. Then, I solved new problems using what I had learned and checked my answers with the correct solutions—that's like training loss and validation loss.



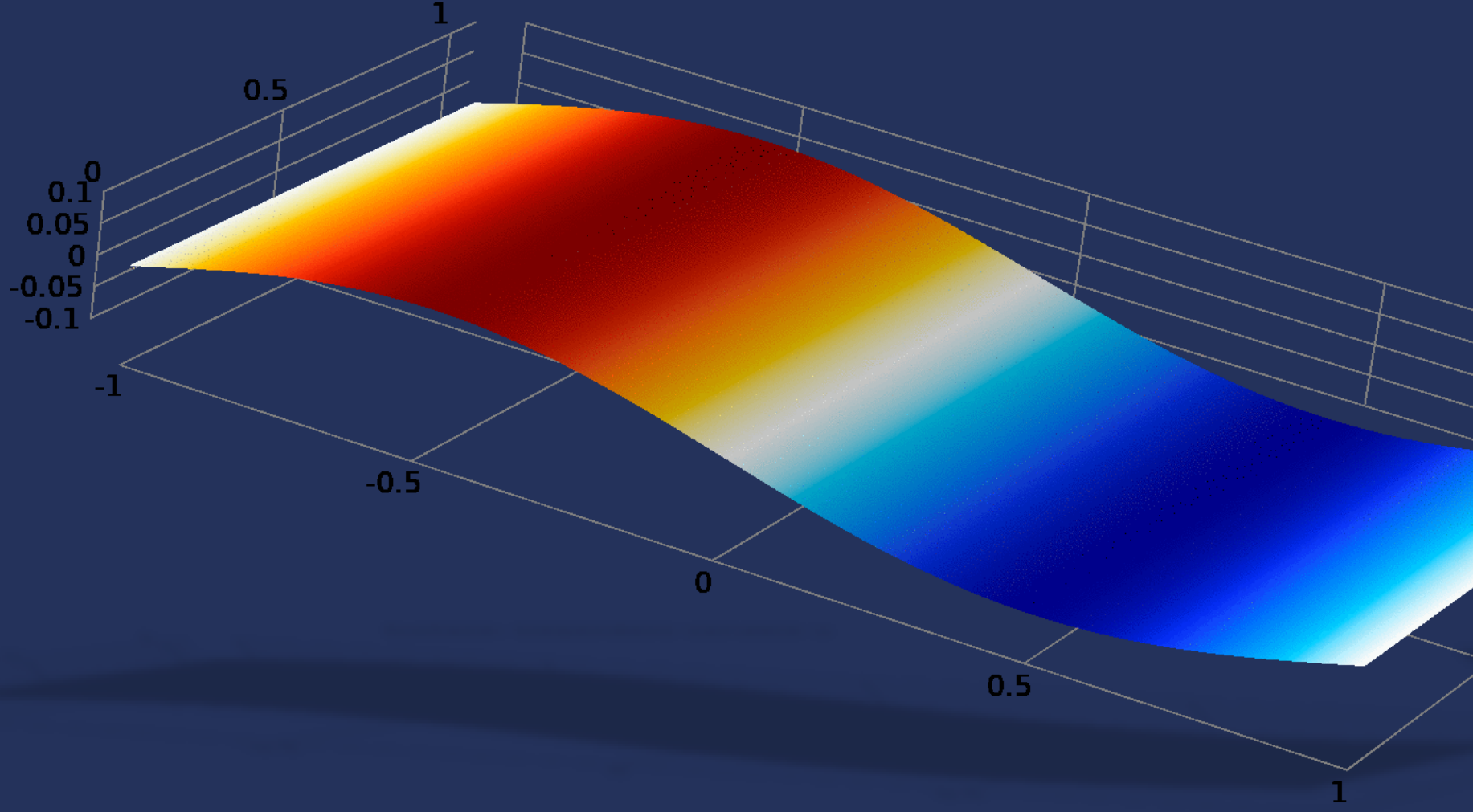
Training and Validation Settings

- *Learning rate*: the step size during the optimization process. Too small rate can lead to the model getting stuck in a local minimum. Too large rate can result in overshooting the minimum and poor convergence.
- *Batch size*: the division of training data into subsets during the optimization process. Too small batch can lead to noisy gradient updates and longer training times. Too large batch might lead to poor generalization.
- *Number of epochs*: number of complete passes through the entire dataset. Too few epochs can result in underfitting, where the model has not adequately learned from the training data. Too many epochs can lead to overfitting, where the model learns the noise in the training data and performs poorly on new, unseen data.
- *Validation data fraction:* the size of the validation data sample.





Visualization of an overfitted model (left) and Well-fitted model (right)

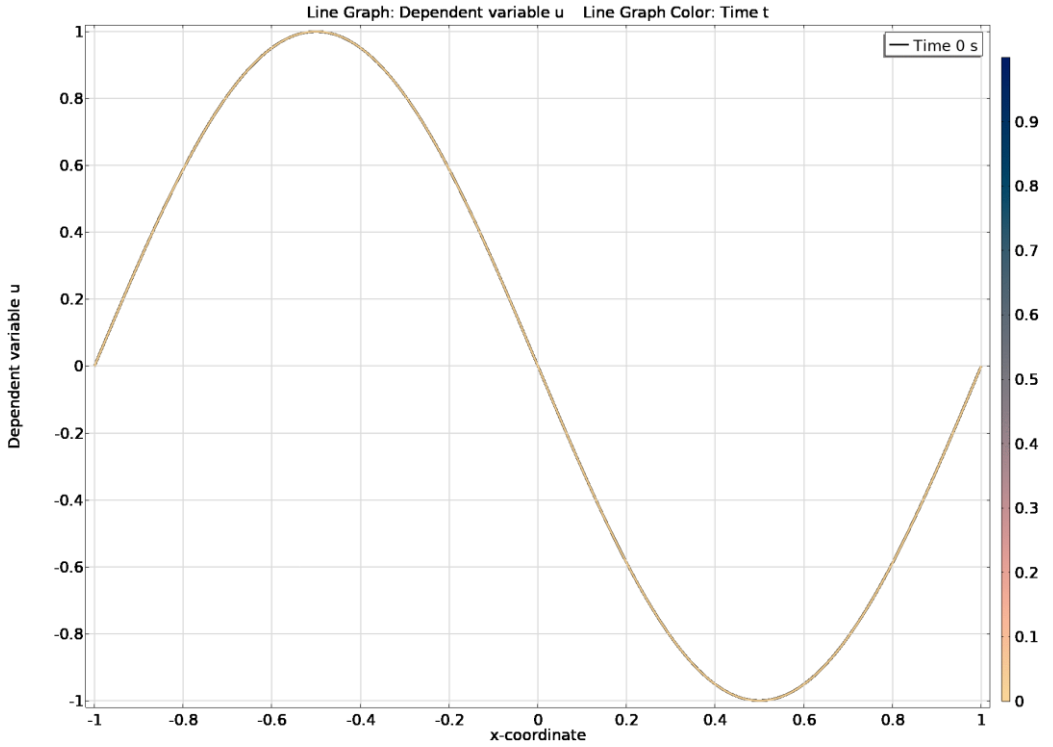


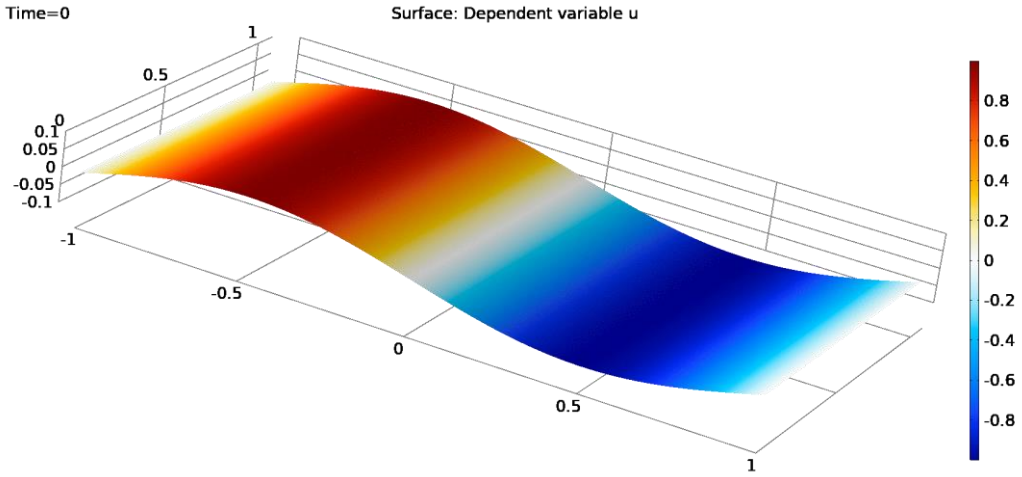
Example in COMSOL Multiphysics

Martin Kožíšek
kozisek@humusoft.cz
www.linkedin.com/in/martinkozisek/

1D Burger‘s equation

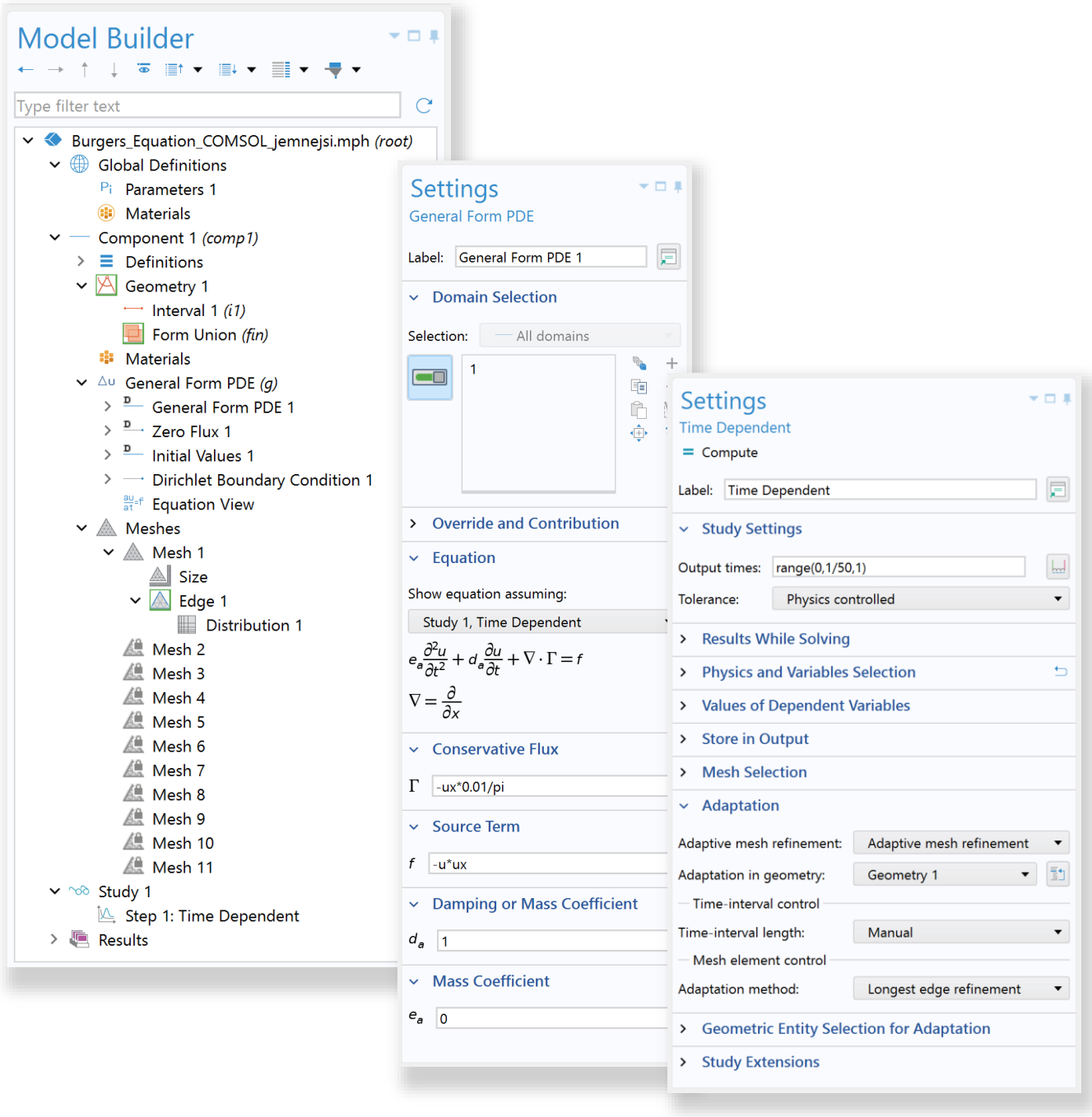
- Equation: $\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \frac{0.01}{\pi}\frac{\partial^2 u}{\partial x^2} = 0$
- Space and time domain: $x \in \langle -1, 1 \rangle$, $t \in \langle 0, 1 \rangle$
- Initial conditions: $u(x, 0) = -sin(\pi x)$
- Boundary conditions: $u(-1, t) = 0$, $u(1, t) = 0$





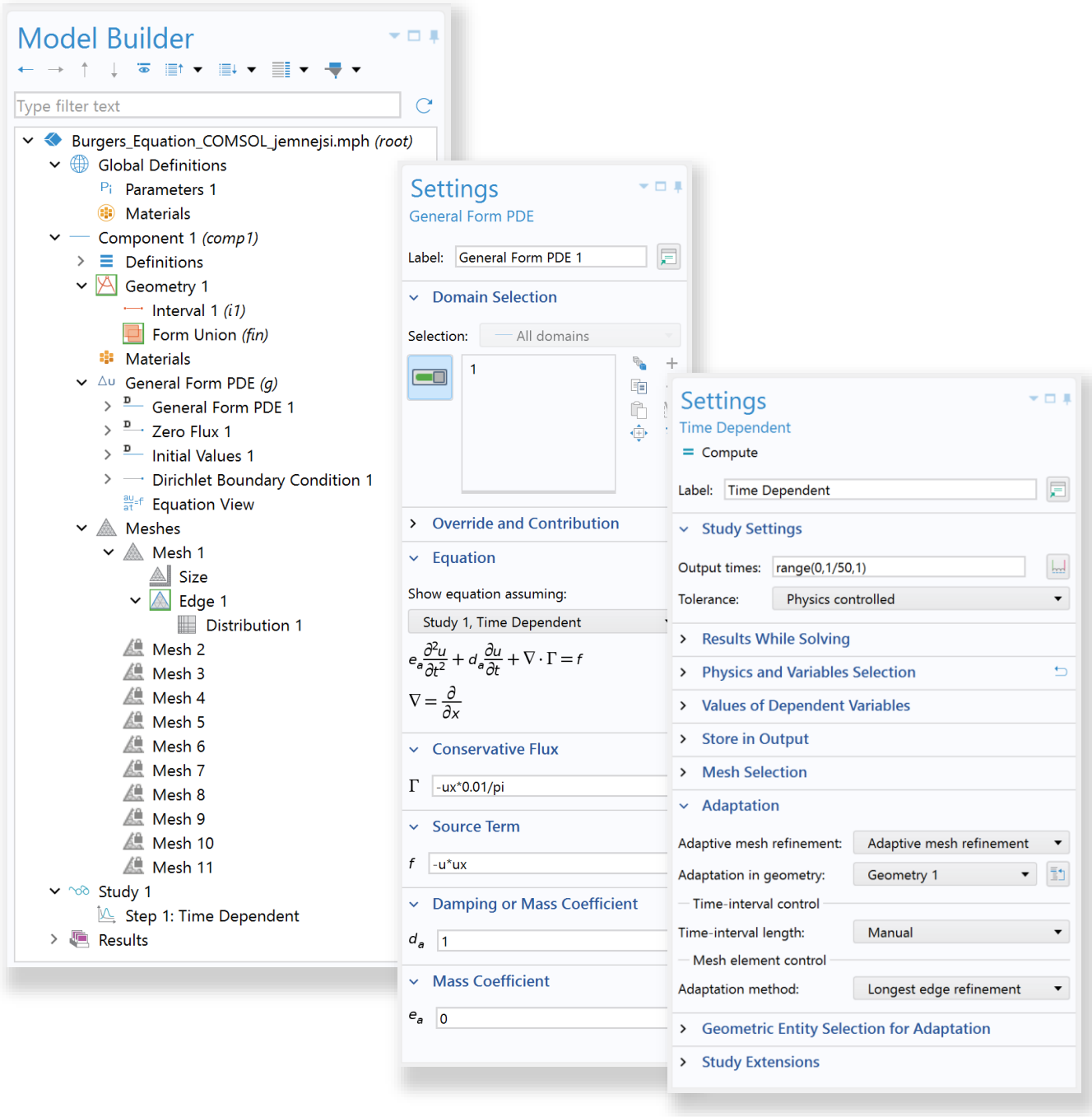
How to Type Your Own Equation in COMSOL

- 1D Component
  - Interval
- 1D General Form PDE
  - Filling the equation into the template
  - Initial Condition Settings
  - Dirichlet Boundary Condition
- User Controled Mesh
  - Edge with Distribution
  - 1000 elements
- Time Dependent Study
  - Output times: range(0,1/50,1)
  - Adaptive mesh refinement



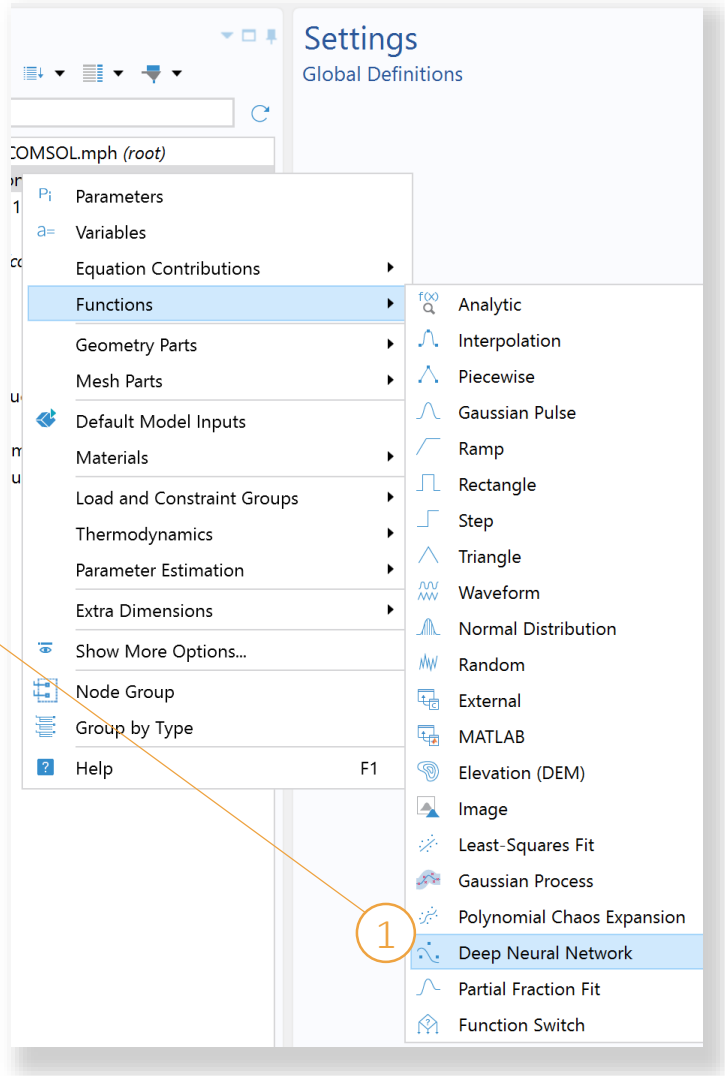
How to Type Your Own Equation in COMSOL

- 1D Component
  - Interval
- 1D General Form PDE
  - Filling the equation into the template
  - Initial Condition Settings
  - Dirichlet Boundary Condition
- User Controled Mesh
  - Edge with Distribution
  - 1000 elements
- Time Dependent Study
  - Output times: range(0,1/50,1)
  - Adaptive mesh refinement



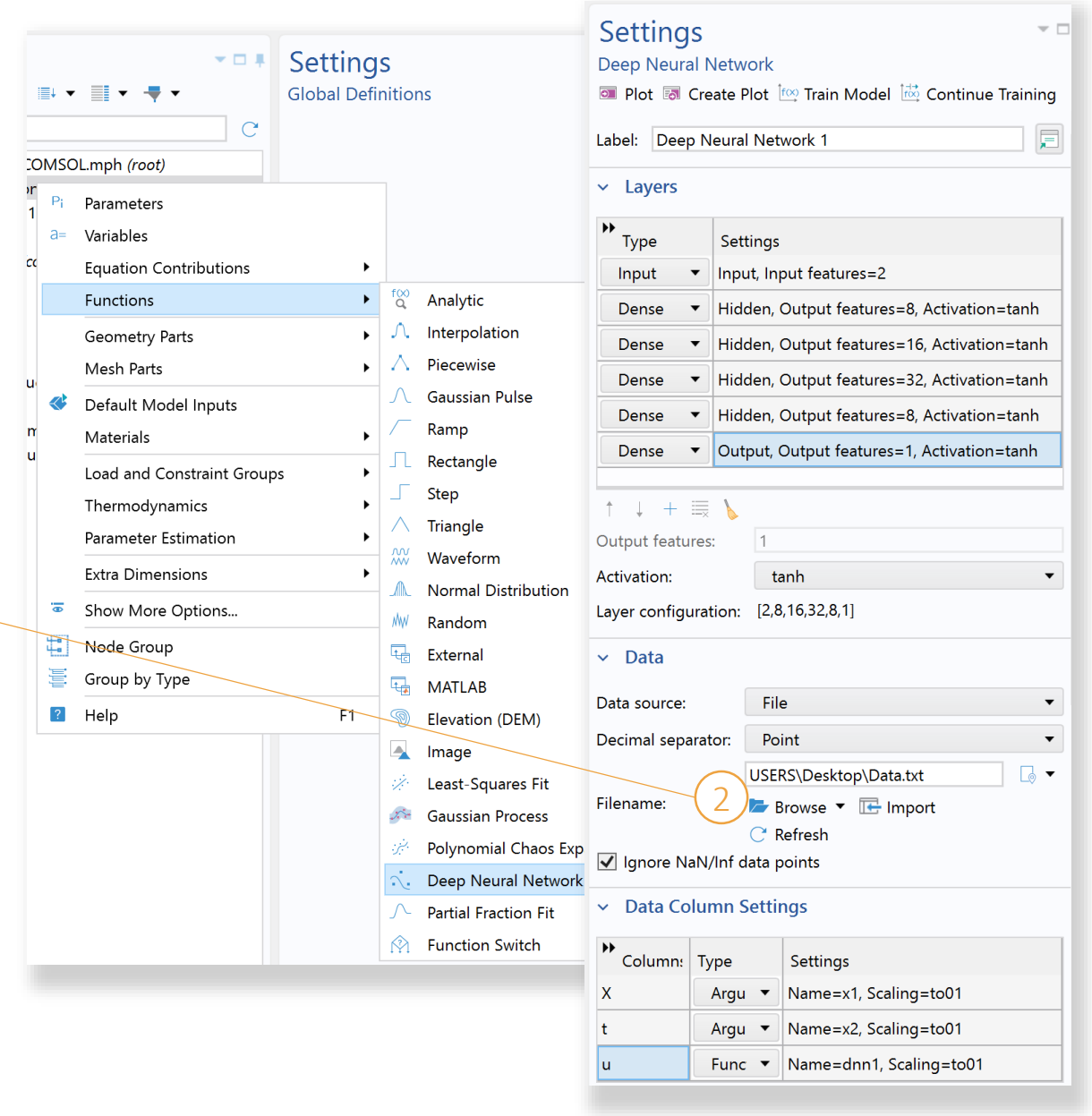
Baking a Deep Neural Network (DNN)

- Export Data
- Functions: Deep Neural Network
  - Data: exported *.txt table
  - Data Columns:
    arguments: *x, t*
    function value: *u*
  - Layers and Activations:
    [2, 8, 16, 32, 8, 1] with *tanh* activation
  - Training and Validation:
    Default settings + Train on GPU
    First 10000 epochs with rate 1e-3
    Other 5000 epochs with rate 1e-4
    Continue Training
- Plot DNN in the *x-t-u* space



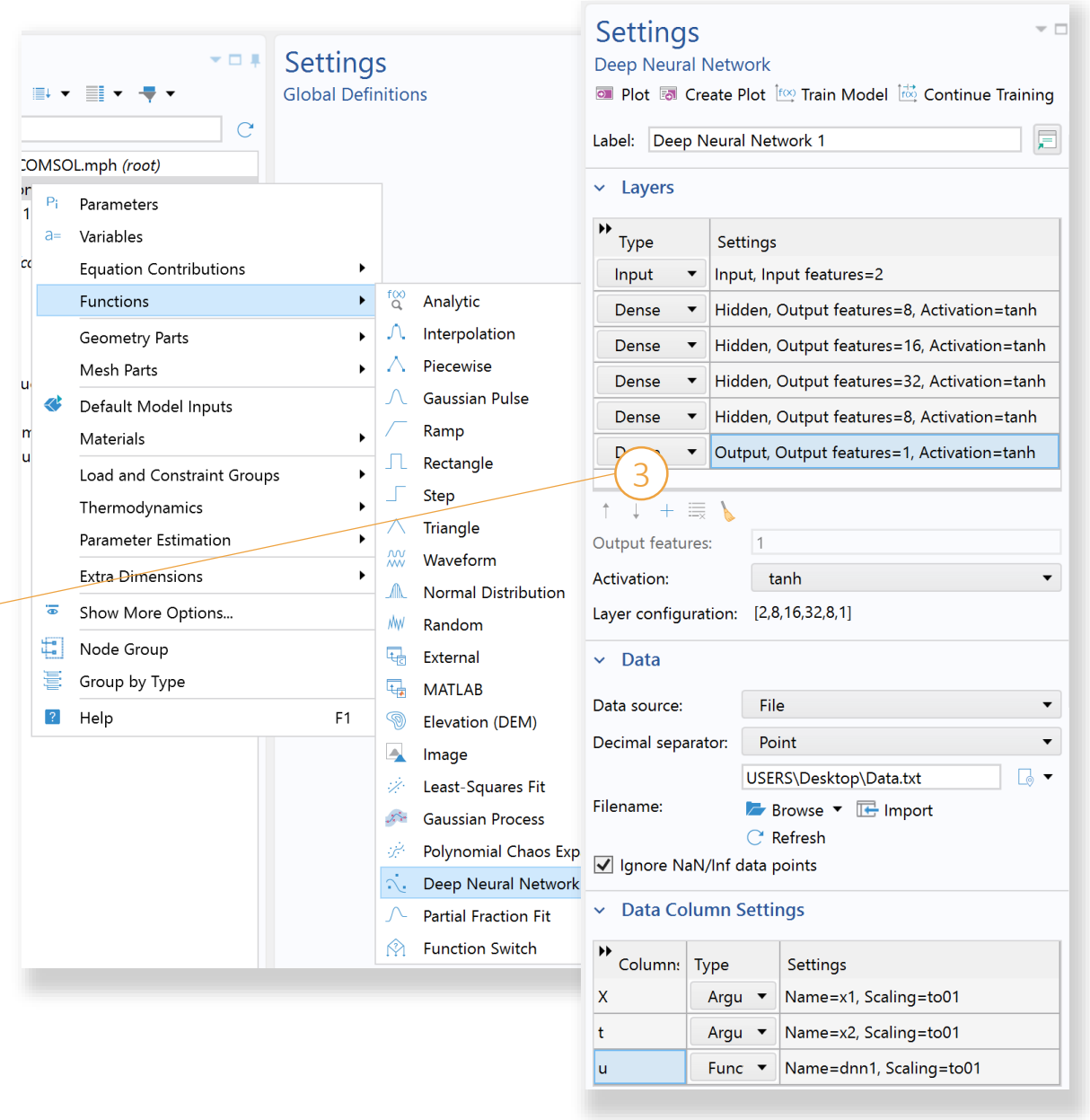
Baking a Deep Neural Network (DNN)

- Export Data
- Functions: Deep Neural Network
  - Data: exported *.txt table (2)
  - Data Columns:
    arguments: *x, t*
    function value: *u*
  - Layers and Activations:
    [2, 8, 16, 32, 8, 1] with *tanh* activation
  - Training and Validation:
    Default settings + Train on GPU
    First 10000 epochs with rate 1e-3
    Other 5000 epochs with rate 1e-4
    Continue Training
- Plot DNN in the *x-t-u* space



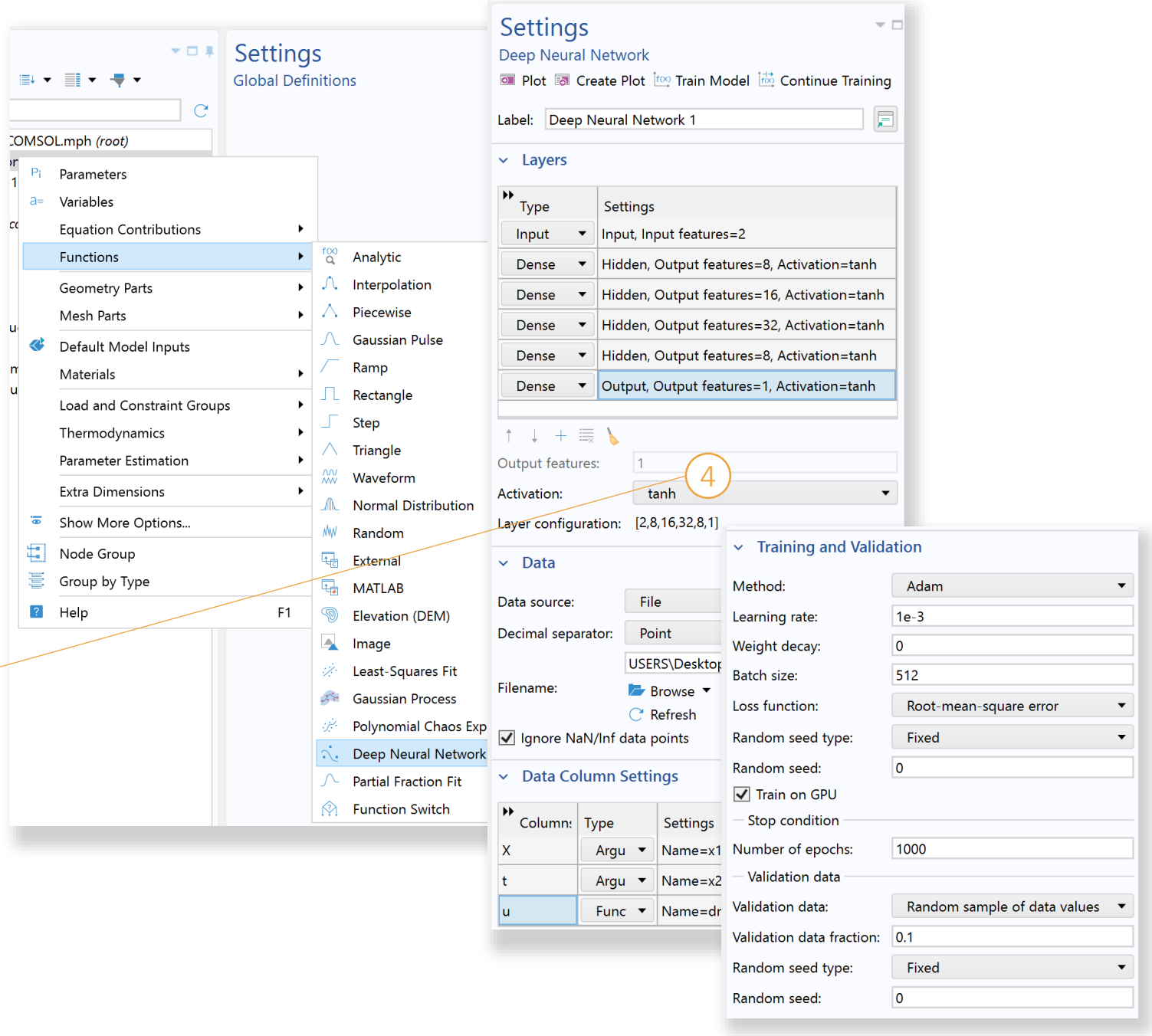
Baking a Deep Neural Network (DNN)

- Export Data
- Functions: Deep Neural Network
  - Data: exported *.txt table
  - Data Columns:
    arguments: *x, t*
    function value: *u*
  - Layers and Activations:
    [2, 8, 16, 32, 8, 1] with *tanh* activation
  - Training and Validation:
    Default settings + Train on GPU
    First 10000 epochs with rate 1e-3
    Other 5000 epochs with rate 1e-4
    Continue Training
- Plot DNN in the *x-t-u* space



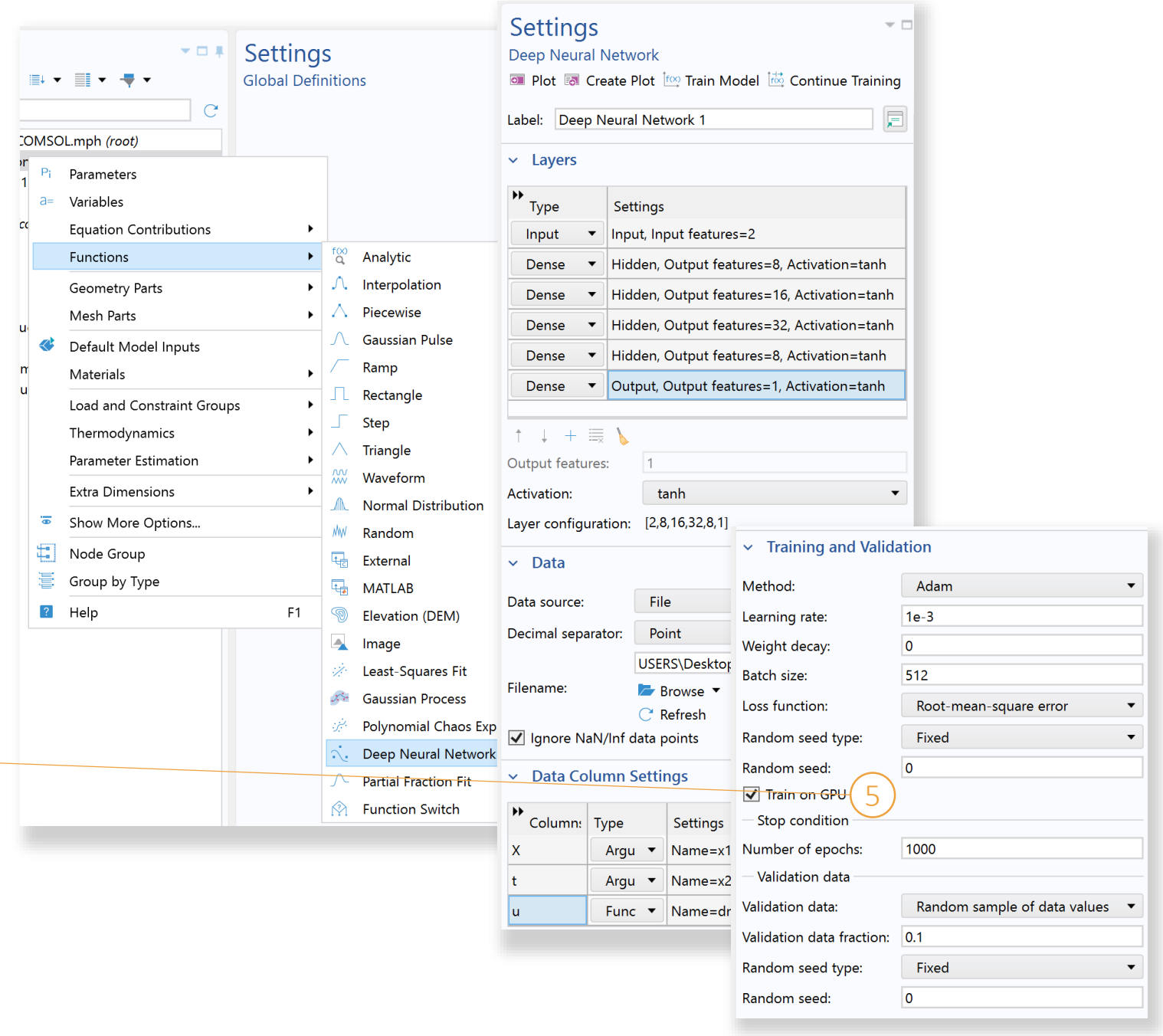
Baking a Deep Neural Network (DNN)

- Export Data
- Functions: Deep Neural Network
  - Data: exported *.txt table
  - Data Columns:
    arguments: *x, t*
    function value: *u*
  - Layers and Activations:
    [2, 8, 16, 32, 8, 1] with *tanh* activation
  - Training and Validation:
    Default settings + Train on GPU
    First 10000 epochs with rate 1e-3
    Other 5000 epochs with rate 1e-4
    Continue Training
- Plot DNN in the *x-t-u* space



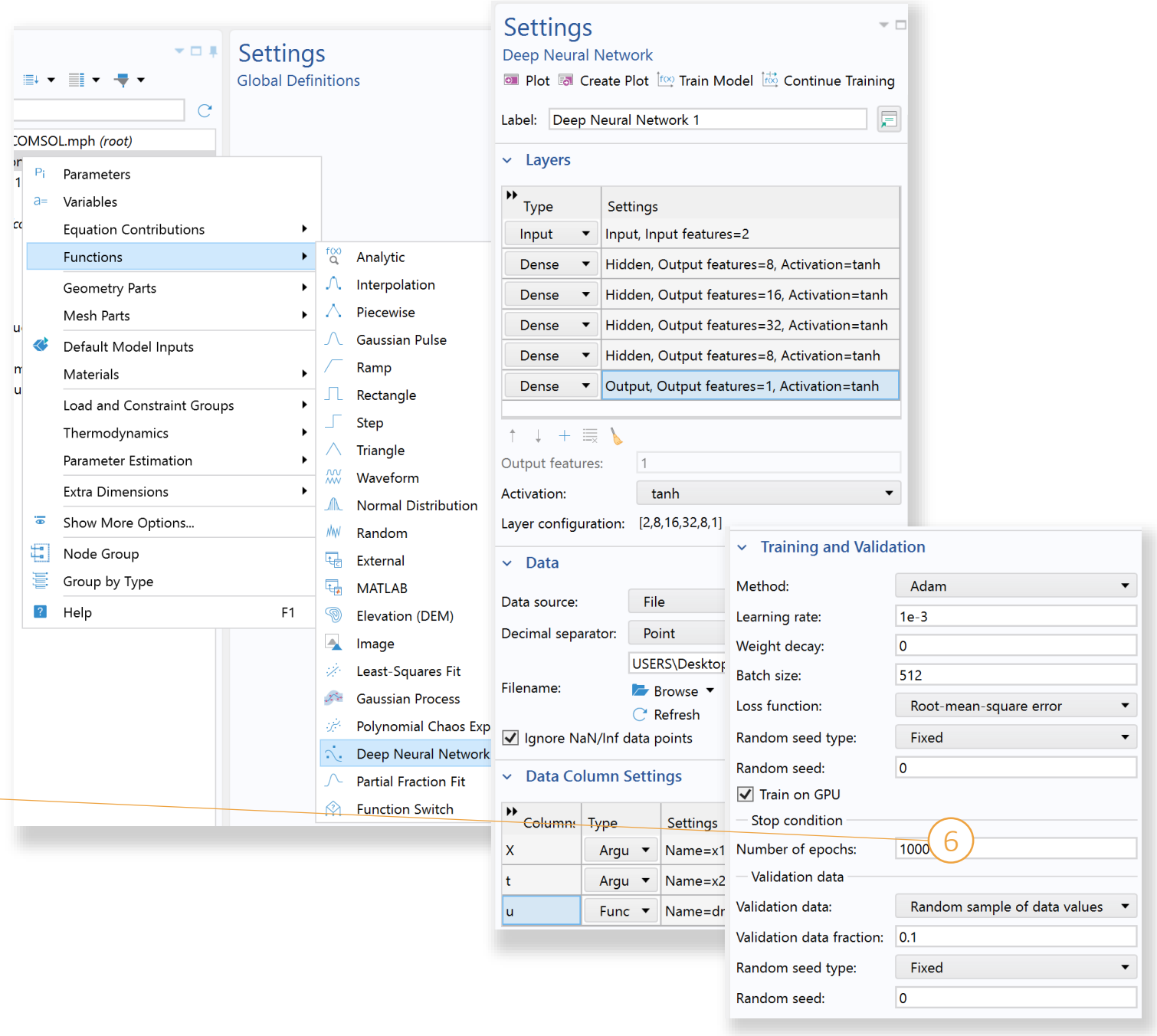
Baking a Deep Neural Network (DNN)

- Export Data
- Functions: Deep Neural Network
  - Data: exported *.txt table
  - Data Columns:
    arguments: *x, t*
    function value: *u*
  - Layers and Activations:
    [2, 8, 16, 32, 8, 1] with *tanh* activation
  - Training and Validation:
    Default settings + Train on GPU
    First 10000 epochs with rate 1e-3
    Other 5000 epochs with rate 1e-4
    Continue Training
- Plot DNN in the *x-t-u* space



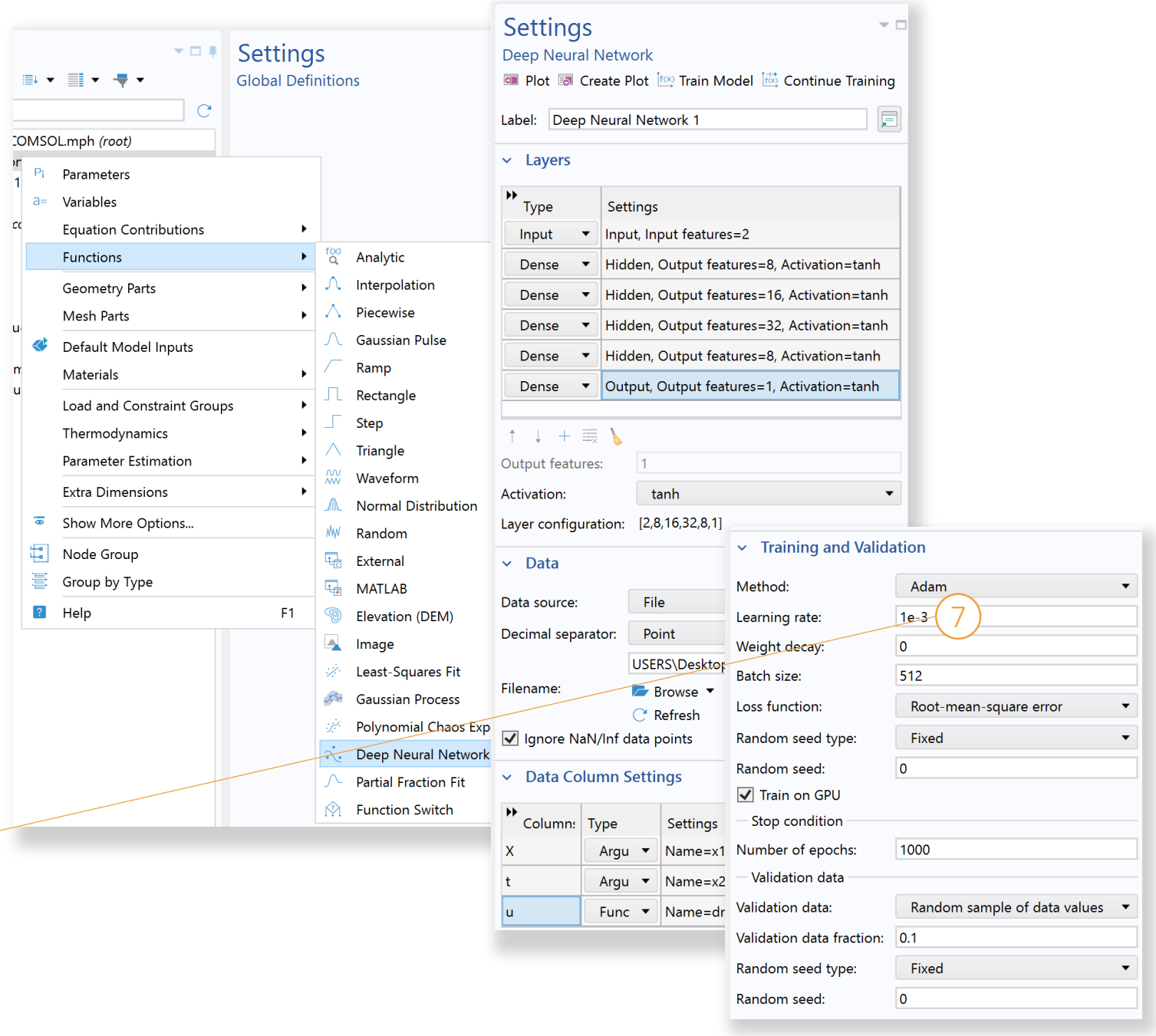
Baking a Deep Neural Network (DNN)

- Export Data
- Functions: Deep Neural Network
  - Data: exported *.txt table
  - Data Columns:
    arguments: *x, t*
    function value: *u*
  - Layers and Activations:
    [2, 8, 16, 32, 8, 1] with *tanh* activation
  - Training and Validation:
    Default settings + Train on GPU
    First 10000 epochs with rate 1e-3
    Other 5000 epochs with rate 1e-4
    Continue Training
- Plot DNN in the *x-t-u* space



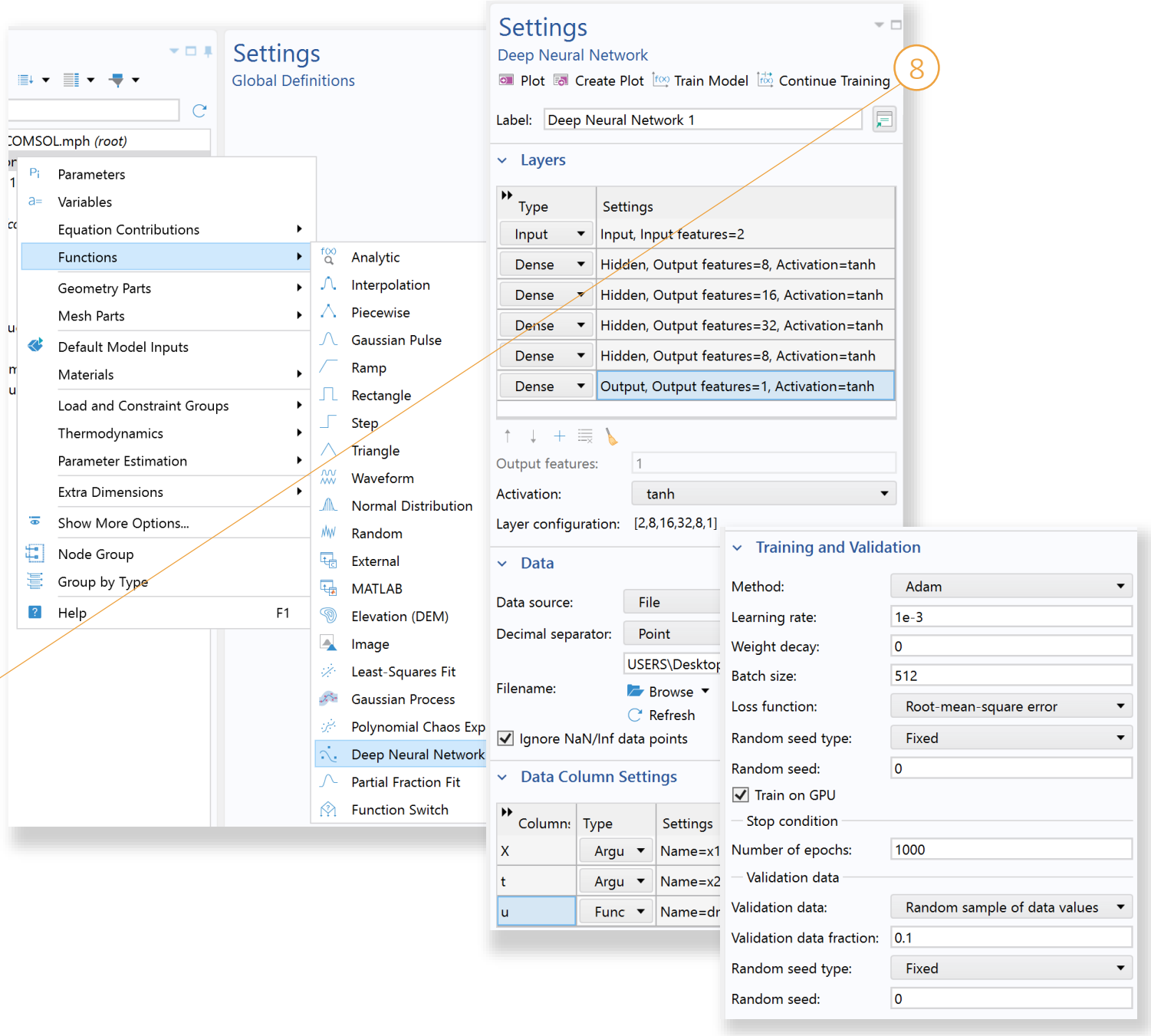
Baking a Deep Neural Network (DNN)

- Export Data
- Functions: Deep Neural Network
  - Data: exported *.txt table
  - Data Columns:
    arguments: *x, t*
    function value: *u*
  - Layers and Activations:
    [2, 8, 16, 32, 8, 1] with *tanh* activation
  - Training and Validation:
    Default settings + Train on GPU
    First 10000 epochs with rate 1e-3
    Other 5000 epochs with rate 1e-4 (7)
    Continue Training
- Plot DNN in the *x-t-u* space



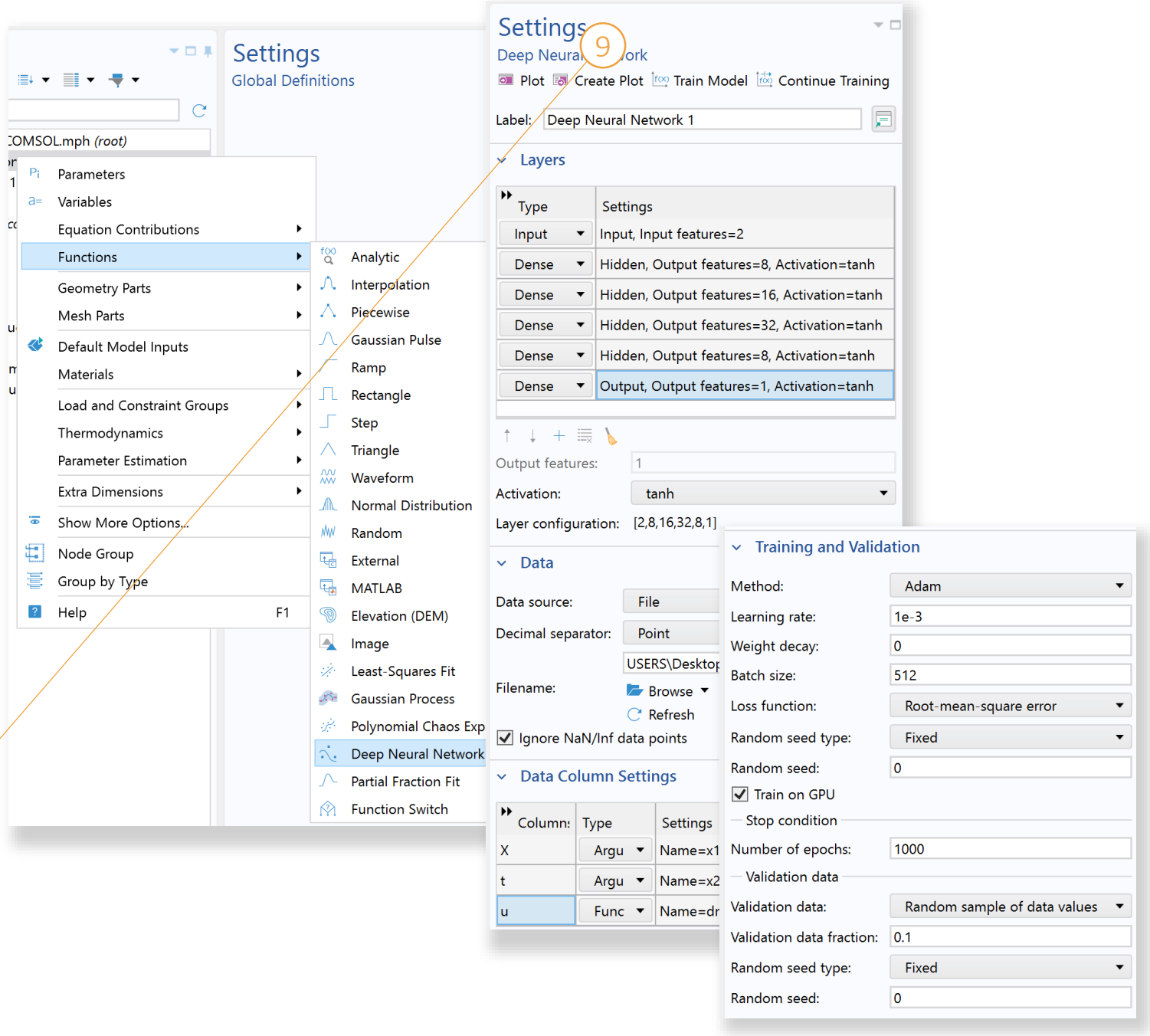
Baking a Deep Neural Network (DNN)

- Export Data
- Functions: Deep Neural Network
  - Data: exported *.txt table
  - Data Columns:
    arguments: *x, t*
    function value: *u*
  - Layers and Activations:
    [2, 8, 16, 32, 8, 1] with *tanh* activation
  - Training and Validation:
    Default settings + Train on GPU
    First 10000 epochs with rate 1e-3
    Other 5000 epochs with rate 1e-4
    Continue Training (8)
- Plot DNN in the *x-t-u* space



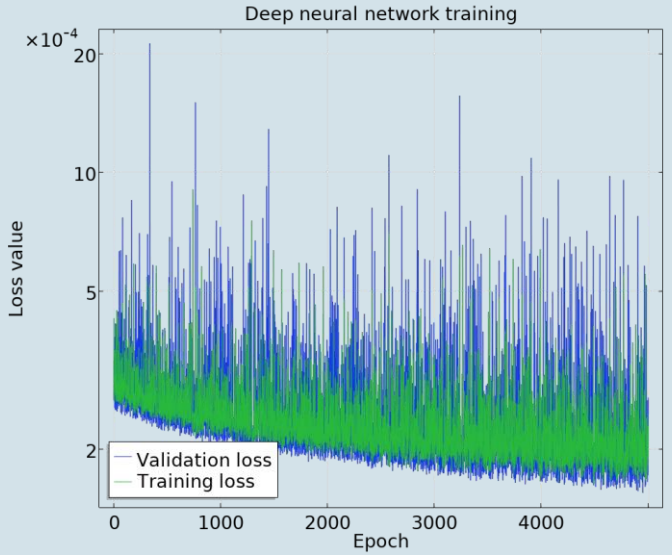
Baking a Deep Neural Network (DNN)

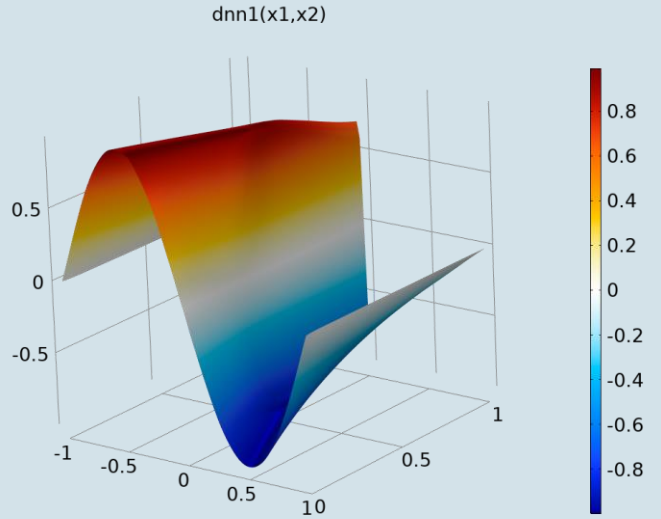
- Export Data
- Functions: Deep Neural Network
  - Data: exported *.txt table
  - Data Columns:
    arguments: *x, t*
    function value: *u*
  - Layers and Activations:
    [2, 8, 16, 32, 8, 1] with *tanh* activation
  - Training and Validation:
    Default settings + Train on GPU
    First 10000 epochs with rate 1e-3
    Other 5000 epochs with rate 1e-4
    Continue Training
- Plot DNN in the *x-t-u* space (9)

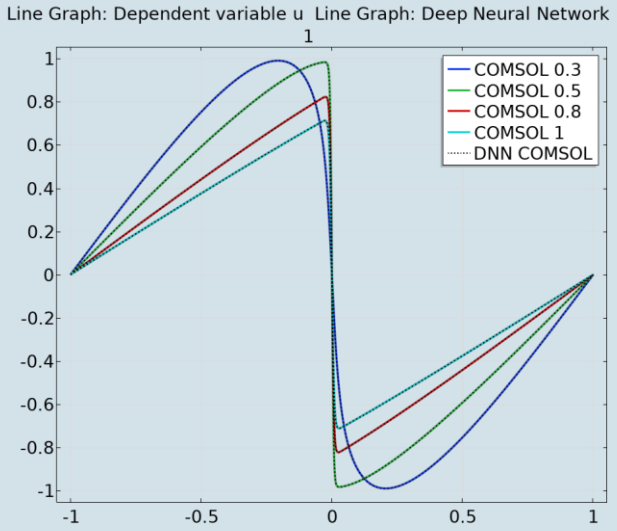


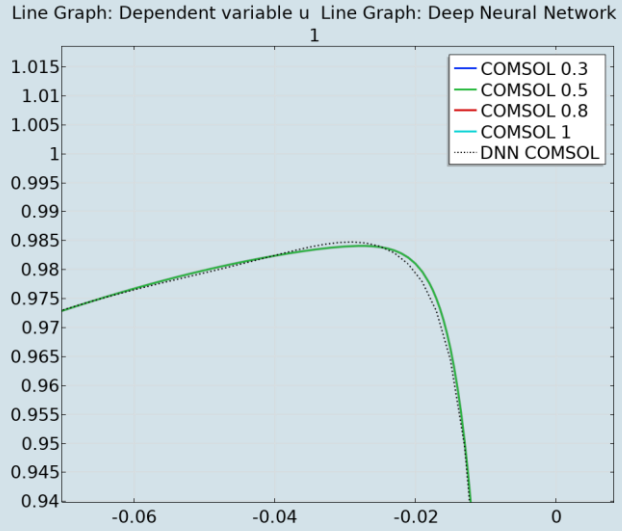
Baking a Deep Neural Network (DNN)

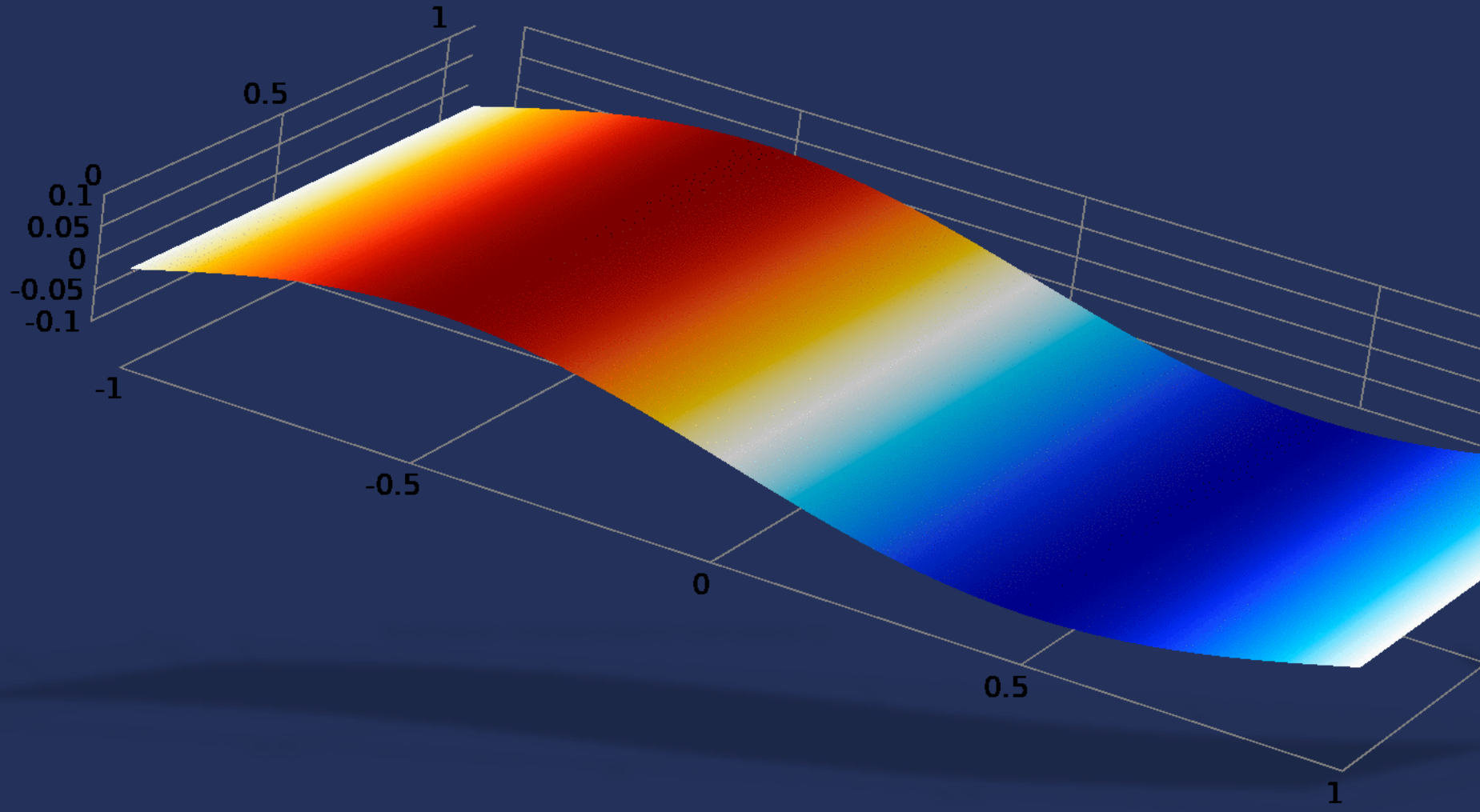
- Export Data
- Functions: Deep Neural Network
  - Data: exported *.txt table
  - Data Columns:
    arguments: *x, t*
    function value: *u*
  - Layers and Activations:
    [2, 8, 16, 32, 8, 1] with *tanh* activation
  - Training and Validation:
    Default settings + Train on GPU
    First 10000 epochs with rate 1e-3
    Other 5000 epochs with rate 1e-4
    Continue Training
- Plot DNN in the *x-t-u* space









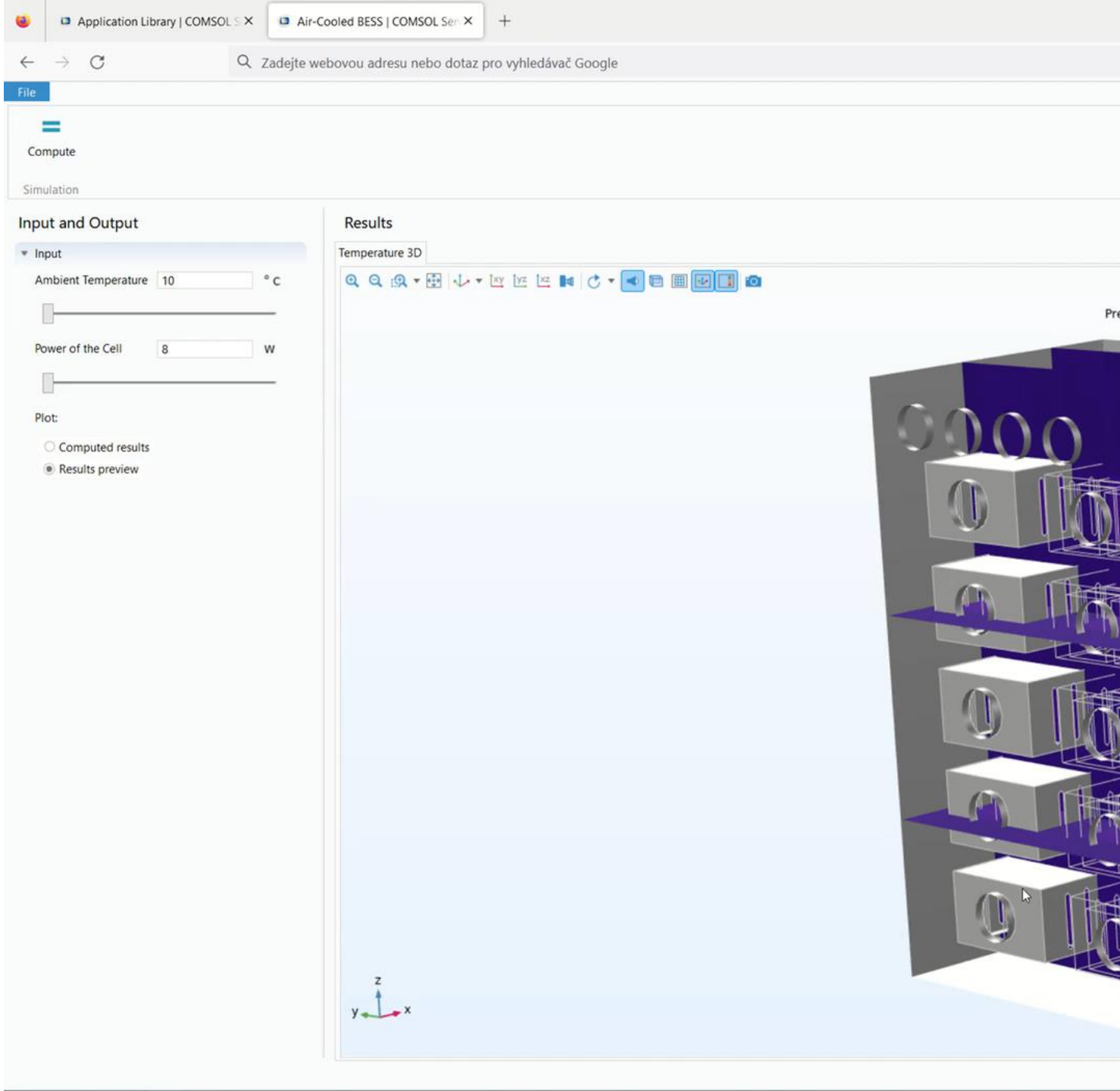


More Realistic Example

Martin Kožíšek
kozisek@humusoft.cz
www.linkedin.com/in/martinkozisek/

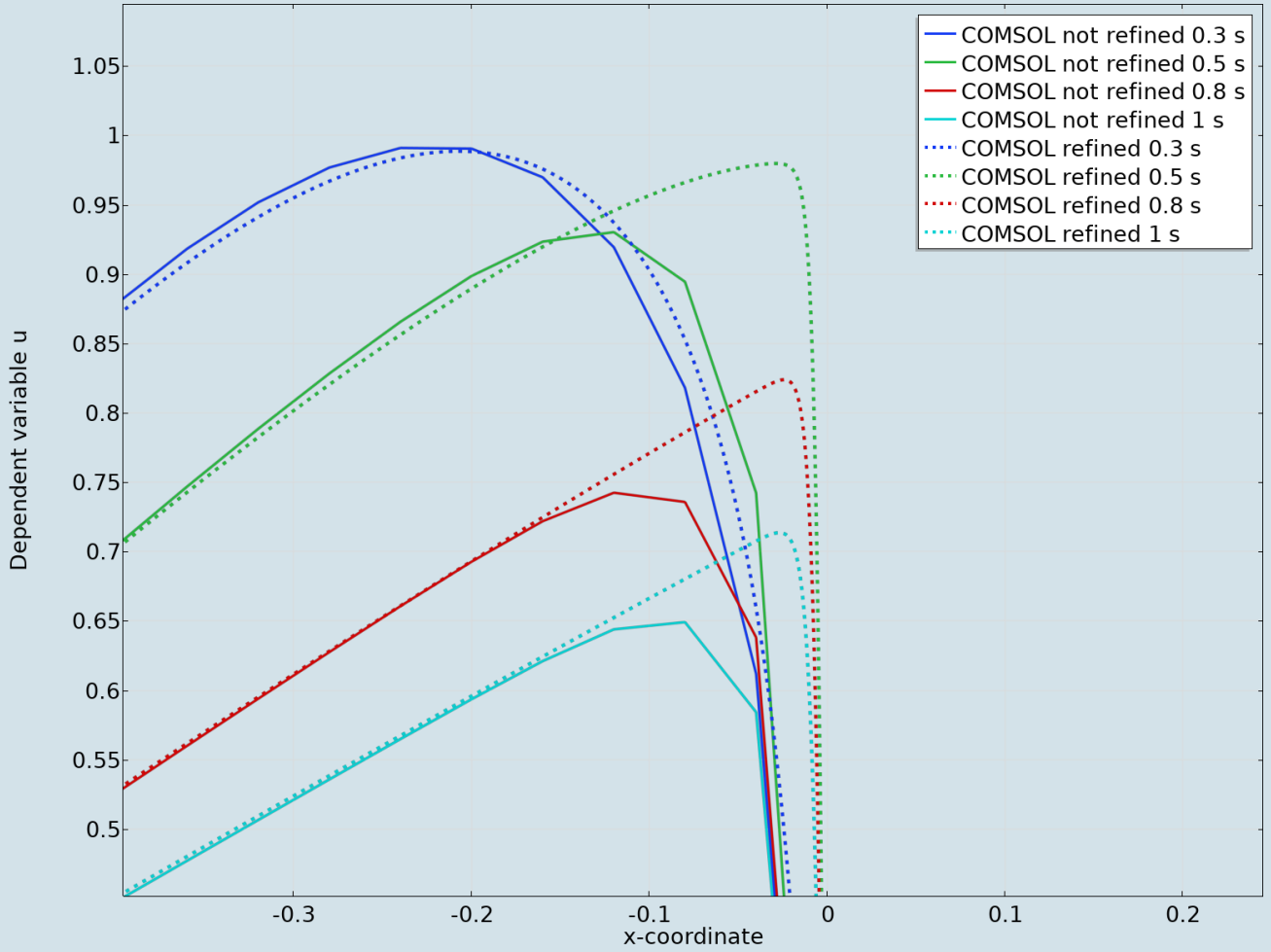
More Realistic Example: Less Training Data

- Design Space *x-t-u* is perfect showcase, but you would probably use interpolation instead of DNN.
- DNN is technology for larger sets of data and more complex systems.
- Imagine *x-y-z-T-P* training data space:
  - Air Cooled BESS
  - Training data for 55 parameter values (105 M rows in the table)
  - DNN [5,50,100,200,300,200,100,50,1]
  - Digital Twin in application
- <u>In this case – you will have probably sparse space of training data</u>



More Realistic Example: Less Training Data

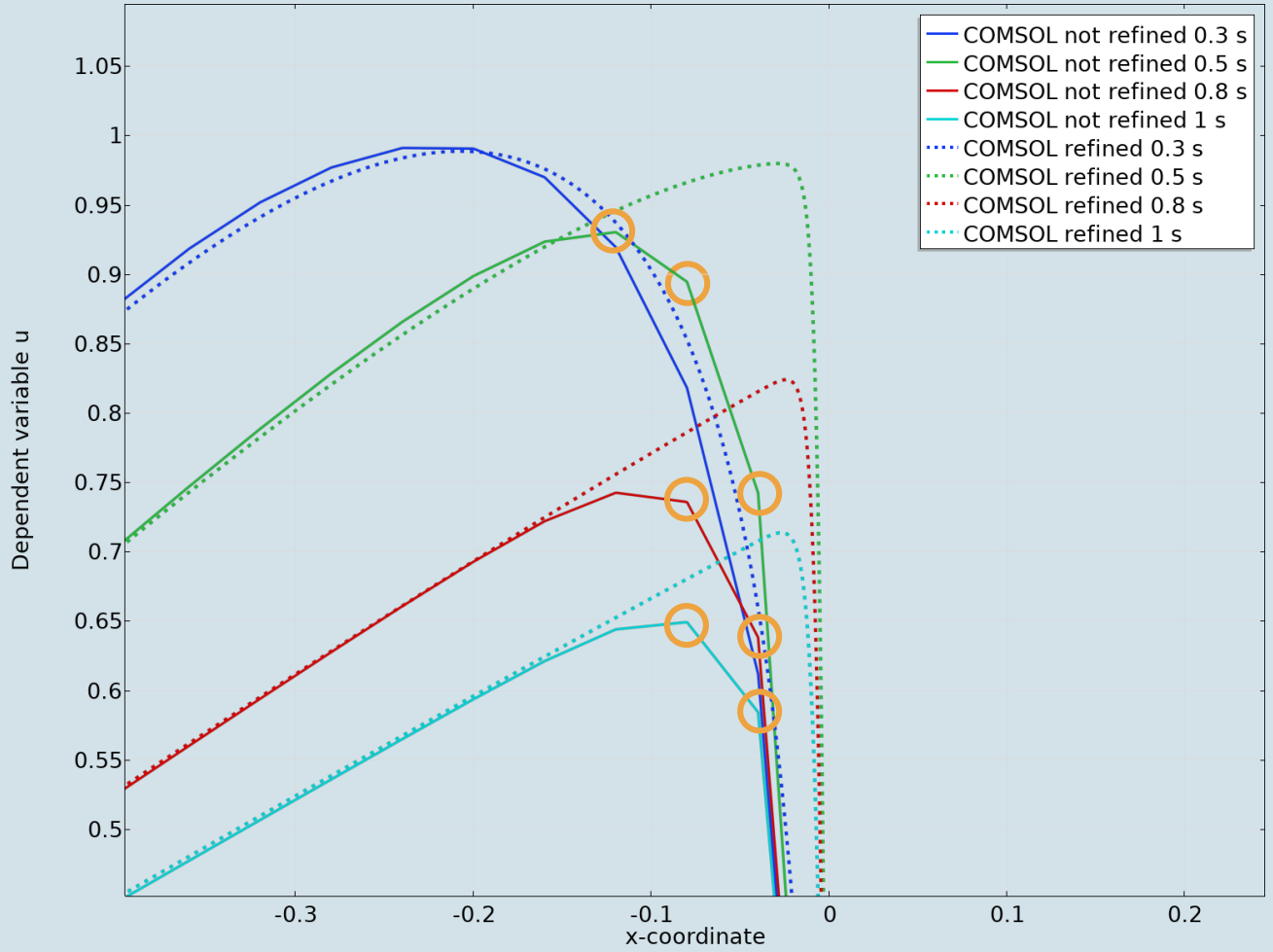
- 1D Component
  - Interval
- 1D General Form PDE
  - Filling the equation into the template
  - Initial Condition Settings
  - Dirichlet Boundary Condition
- User Controled Mesh
  - Edge with Distribution
  - ~~1000 elements~~ 50 elements
- Time Dependent Study
  - Output times: range(0,1/50,1)
  - ~~Adaptive mesh refinement~~



More Realistic Example: Less Training Data

- 1D Component
  - Interval
- 1D General Form PDE
  - Filling the equation into the template
  - Initial Condition Settings
  - Dirichlet Boundary Condition
- User Controled Mesh
  - Edge with Distribution
  - ~~1000 elements~~ 50 elements
- Time Dependent Study
  - Output times: range(0,1/50,1)
  - ~~Adaptive mesh refinement~~

What if there is only a small amount of data due to high computational cost? Training data are either missing or have bad precision in potentially important regions! DNN has no chance to approximate the real-physics correctly.



More Realistic Example: Less Training Data

- 1D Component
  - Interval
- 1D General Form PDE
  - Filling the equation into the template
  - Initial Condition Settings
  - Dirichlet Boundary Condition
- User Controled Mesh
  - Edge with Distribution
  - ~~1000 elements~~ 50 elements
- Time Dependent Study
  - Output times: range(0,1/50,1)
  - ~~Adaptive mesh refinement~~

What if there is only a small amount of data due to high computational cost? Training data are either missing or have bad precision in potentially important regions! DNN has no chance to approximate the real-physics correctly.

One option is to include information in the loss function about whether the PDE is satisfied!