5th workshop CSMA Junior

Mai 14th-16th 2022

Île de Porquerolles

Deep learning, real-time simulation and model-order reduction

Application to the constitutive modeling of materials

Practical session

Filippo Masi

filippo.masi@ec-nantes.fr

Nantes Université, École Centrale Nantes, CNRS, GeM

CENTRALE NANTES

cnrs

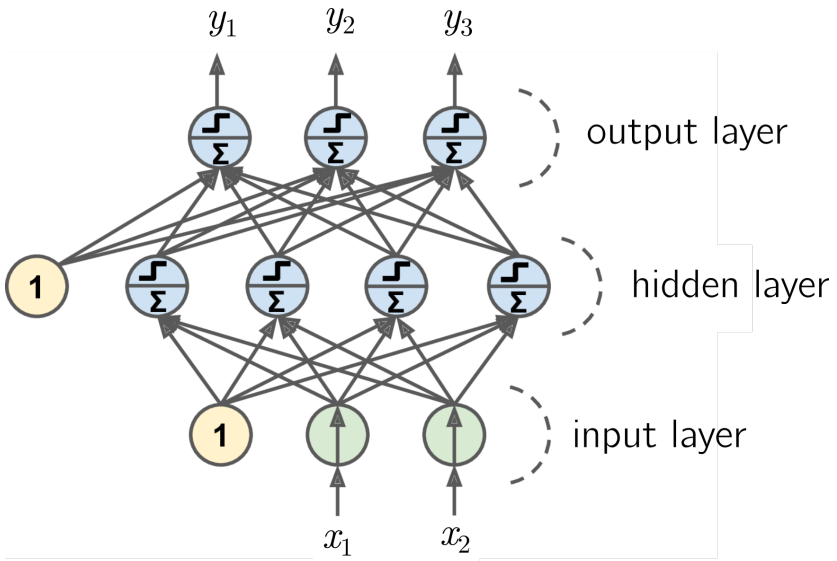
Nantes Université

European Research Council

Established by the European Commission

Neural Networks in a nutshell

Multilayer perceptrons aka feed-forward neural networks



$$\underbrace{\boldsymbol{y}^{(k)}}_{\in\mathbb{R}^n} = \underbrace{\mathcal{A}^{(k)}}_{\text{activation}} \left(\underbrace{\boldsymbol{y}^{(k-1)^{\mathrm{T}}}}_{\in\mathbb{R}^m} \underbrace{\boldsymbol{W}^{(k)}}_{\in\mathbb{R}^{m\times n}} + \underbrace{\boldsymbol{b}^{(k)}}_{\in\mathbb{R}^n} \right)$$

inputs $\boldsymbol{x} \equiv \boldsymbol{y}^{(0)}$

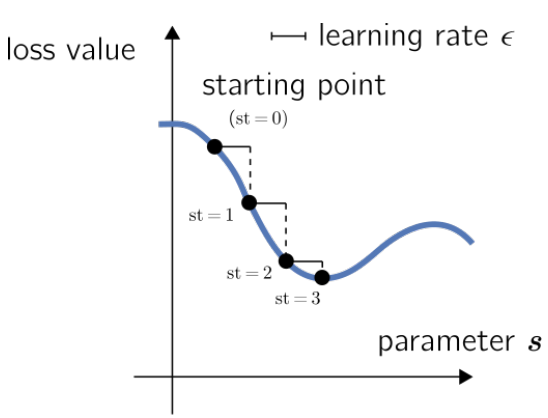
hyper-parameters

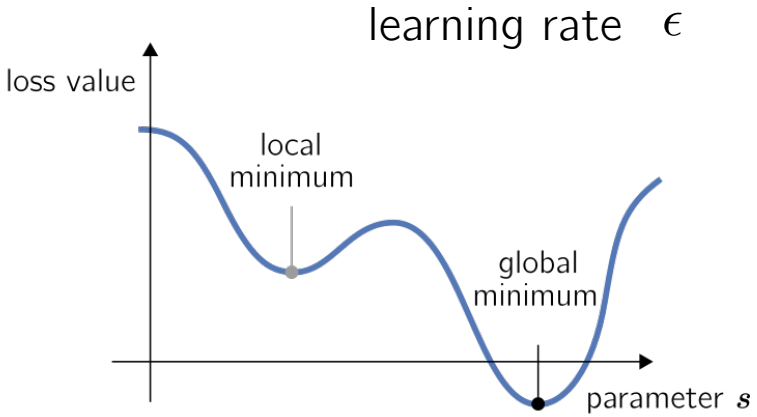
$$\boldsymbol{s} = \{\boldsymbol{W}^{(k)}, \boldsymbol{b}^{(k)}\} \quad \forall\, k \in [0, n]$$

Neural Networks in a nutshell

Gradient descent

$$\boldsymbol{s}^{(\text{next step})} = \boldsymbol{s} - \epsilon \frac{\partial}{\partial \boldsymbol{s}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{s})$$



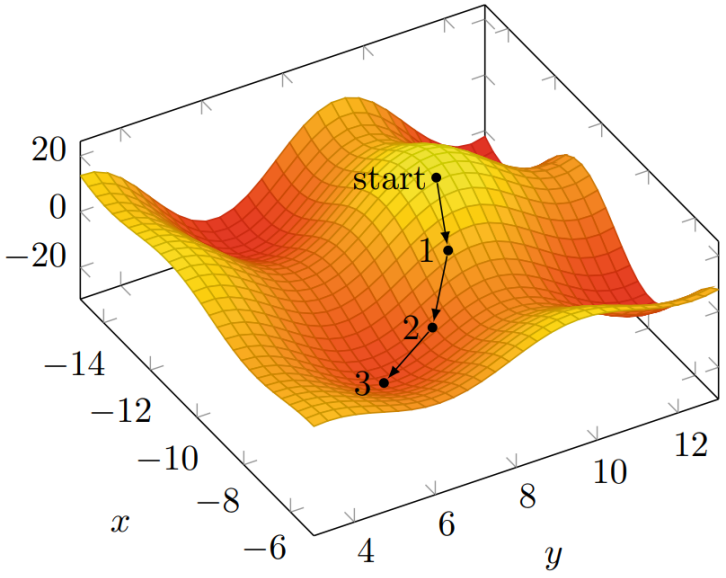


loss function

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{s}) = \frac{1}{N} \sum_{i=1}^{M} \ell(\boldsymbol{o}^i, \bar{\boldsymbol{o}}^i)$$

hyper-parameters

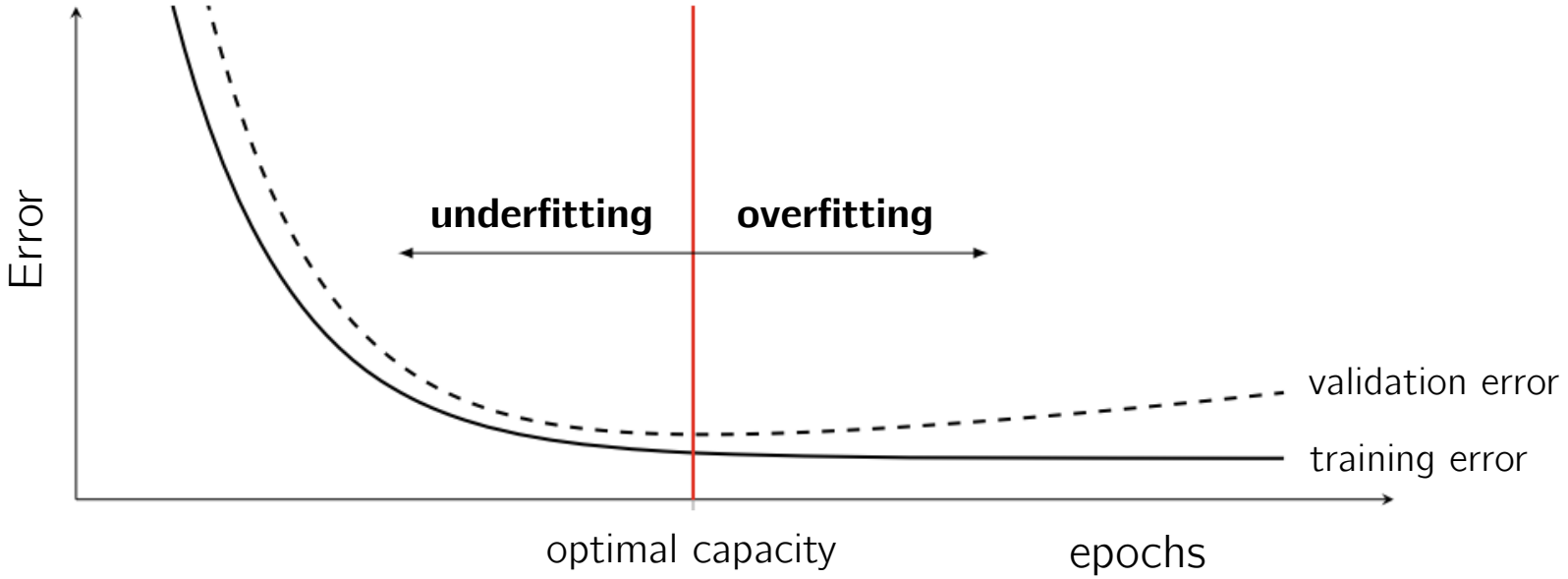
$$\arg\min_{\boldsymbol{s}} \left(\mathcal{L}(\boldsymbol{x}, \boldsymbol{s})\right)$$



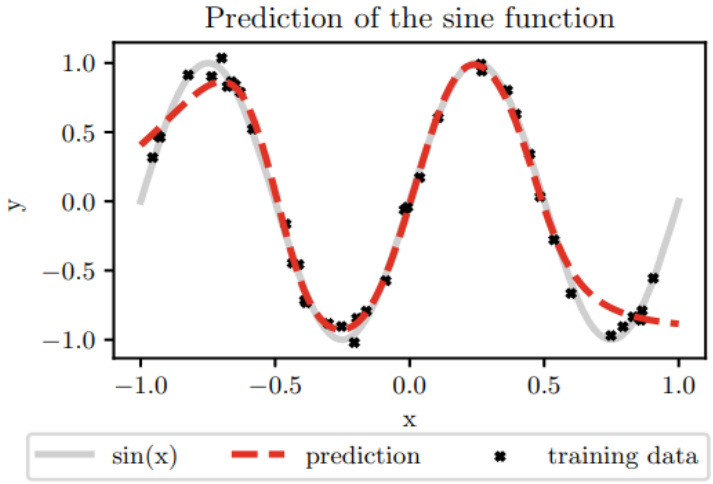
Neural Networks in a nutshell

Gradient descent

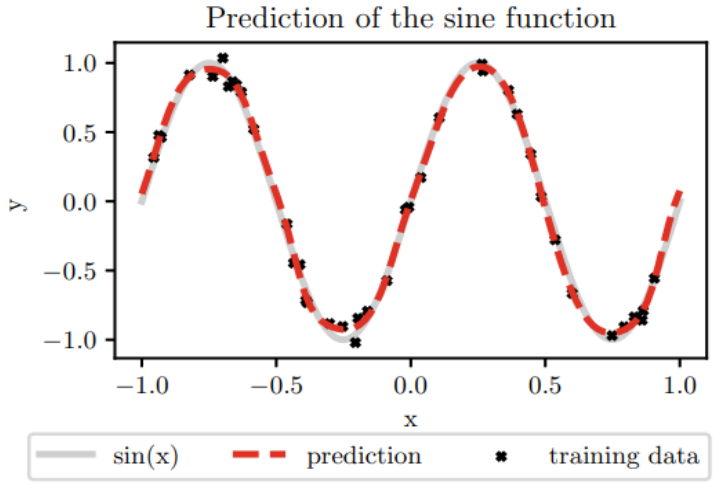
$$\boldsymbol{s}^{(\text{next step})} = \boldsymbol{s} - \epsilon \frac{\partial}{\partial \boldsymbol{s}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{s})$$



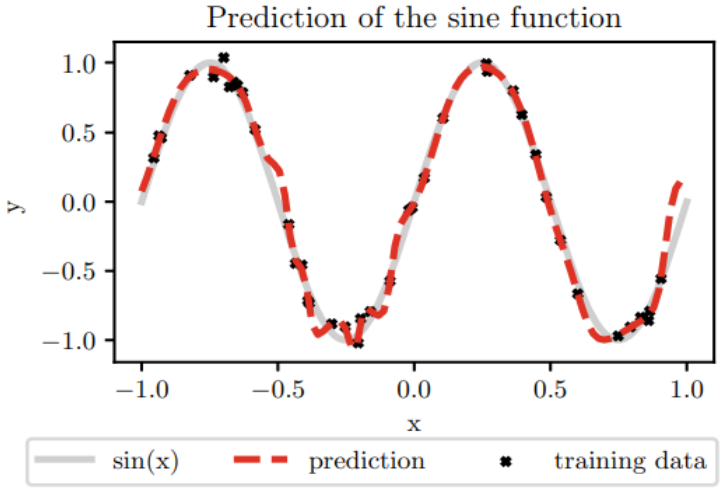
Neural Networks in a nutshell



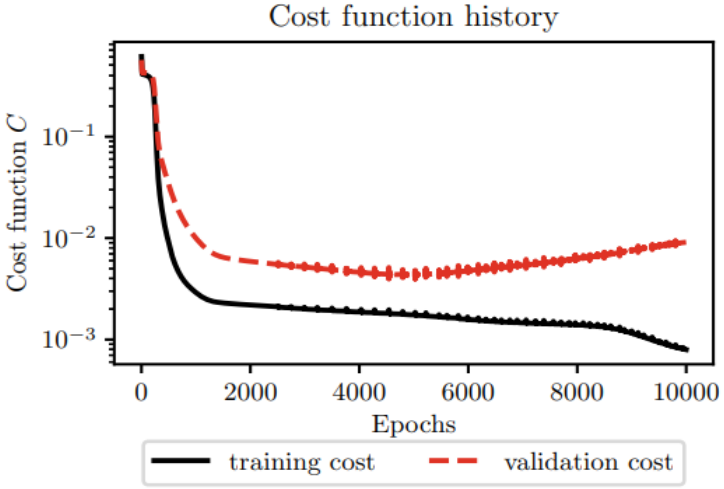
after 500 training epochs



after 5'000 training epochs



after 10'000 epochs of training

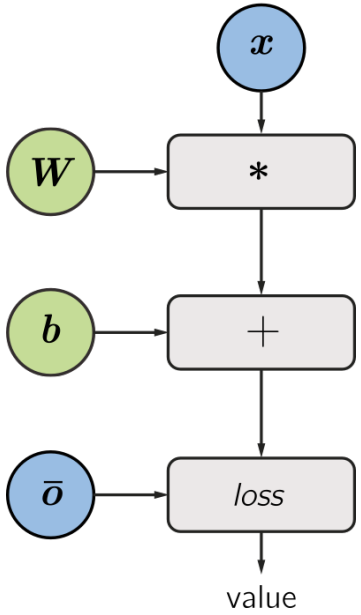


Neural Networks in a nutshell

Gradient computation

$$\boldsymbol{s}^{(\text{next step})} = \boldsymbol{s} - \epsilon \frac{\partial}{\partial \boldsymbol{s}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{s})$$

computation graph



Computation graphs enable to treat computation as data: a computable expression is encoded as a machine-readable data structure

Exercise #1

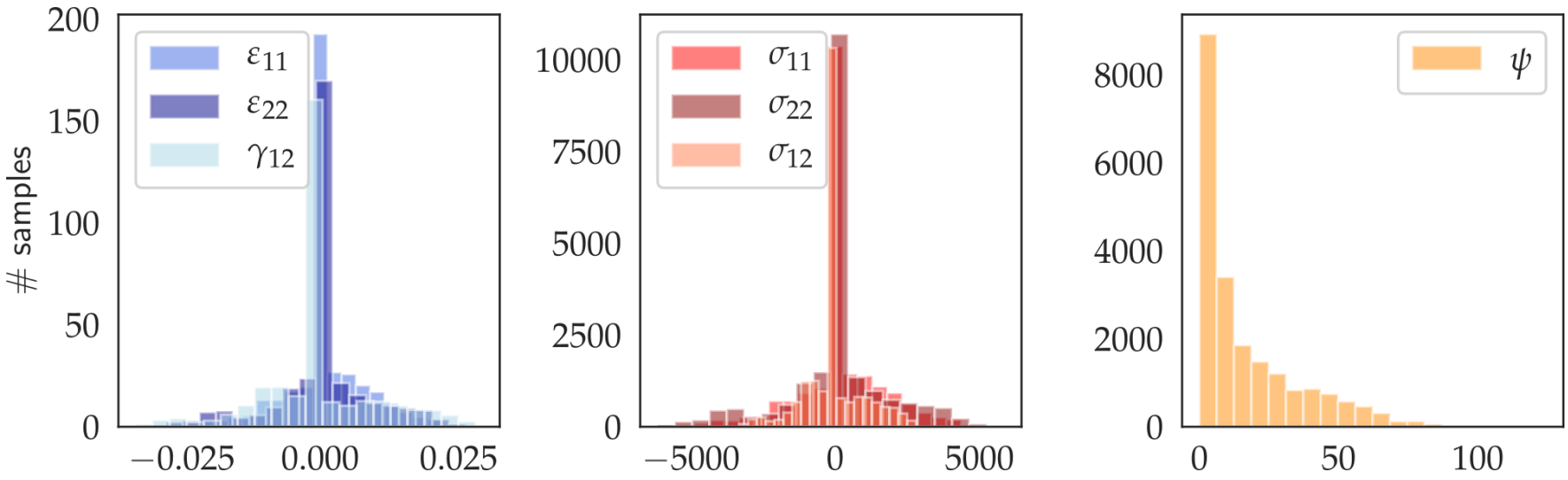
Exercise #1

Three-dimensional elasticity (plane strain)

$$\sigma_{ij} = \begin{pmatrix} \sigma_{11} & \sigma_{12} & 0 \\ \sigma_{21} & \sigma_{22} & 0 \\ 0 & 0 & \sigma_{33} \end{pmatrix} \qquad \varepsilon_{ij} = \begin{pmatrix} \varepsilon_{11} & \varepsilon_{12} & 0 \\ \varepsilon_{21} & \varepsilon_{22} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\tilde{\sigma}_{ij} = (\sigma_{11}, \sigma_{22}, \sigma_{33}, \sigma_{12}) \qquad \tilde{\varepsilon}_{ij} = (\varepsilon_{11}, \varepsilon_{22}, \varepsilon_{33}, \gamma_{12})$$

Datasets (random loading path, strain-driven)

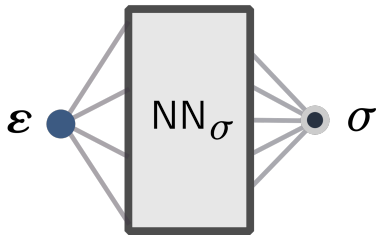


Neural Networks for constitutive modeling

Neural Networks (NN)

$$\boldsymbol{\sigma} = \mathrm{NN}\left(\boldsymbol{\varepsilon}\right)$$

black-box



$$\underbrace{\|\boldsymbol{\sigma} - \mathrm{NN}_{\boldsymbol{\sigma}}\|}_{\text{loss in } \boldsymbol{\sigma}}$$

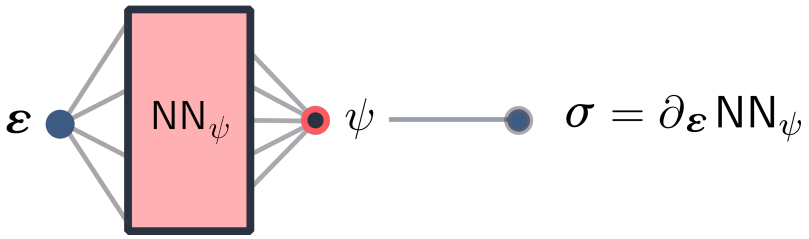
```
if self.material == 'elastic':
     strain = inputs
     nStress = self.NN_Stress(strain)
     return nStress
```

Neural Networks for constitutive modeling

Thermodynamics-based Artificial Neural Networks (TANN)

$$\boldsymbol{\sigma} = \mathrm{TANN}\left(\boldsymbol{\varepsilon}\right)$$

thermodynamics



$$\underbrace{\underbrace{\|\psi - \mathrm{NN}_\psi\|}_{\text{loss in } \psi} + \underbrace{\|\boldsymbol{\sigma} - \partial_{\boldsymbol{\varepsilon}} \mathrm{NN}_\psi\|}_{\text{loss in } \boldsymbol{\sigma}}}_{1^{\text{st}} \text{ principle}}$$

```
if self.material == 'elastic':
     nStrain = inputs
     uStrain = self.DeNormalize(nStrain,self.prm_E)
     NStrain = self.Normalize(uStrain,self.prm_E)

     state = NStrain
     nEnergy = self.NN_Energy(state)
     uEnergy = self.DeNormalize(nEnergy,self.prm_F)

     Stress = tf.gradients(Energy,Strain)[0]
     nStress = self.Normalize(Stress,self.prm_S)

     return [nEnergy,nStress]
```

Exercise #1

```
./workshop/elasticity/TANN_training
./workshop/elasticity/NN_training
```

$\boldsymbol{\sigma} = \mathrm{NN}(\boldsymbol{\varepsilon})$

tanh

hyperbolic tangent

$\boldsymbol{\sigma} = \mathrm{TANN}(\boldsymbol{\varepsilon})$

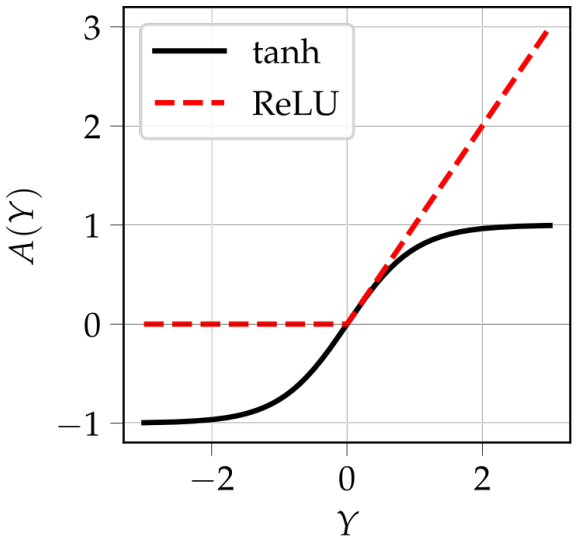
GELU

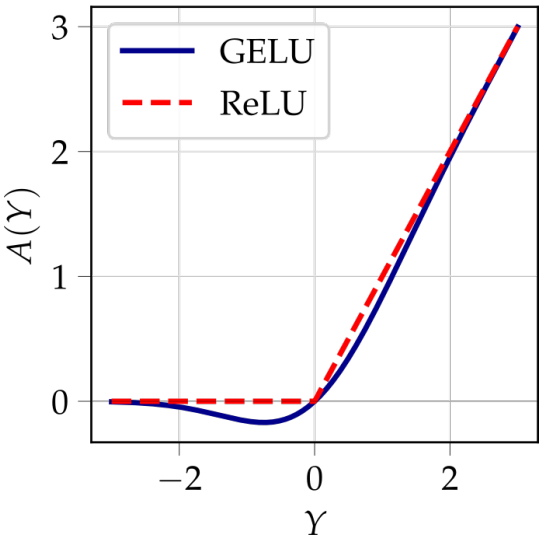
Gaussian Error Linear Unit

$\boldsymbol{\sigma} = \mathrm{TANN}(\boldsymbol{\varepsilon})$

ReLU

Rectified Linear Unit



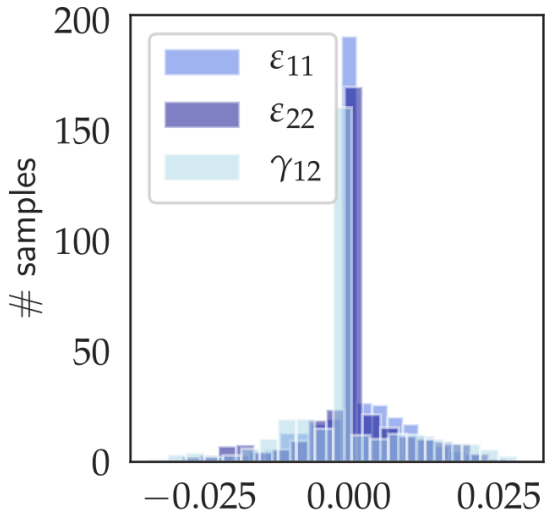


Normalization inputs/outputs

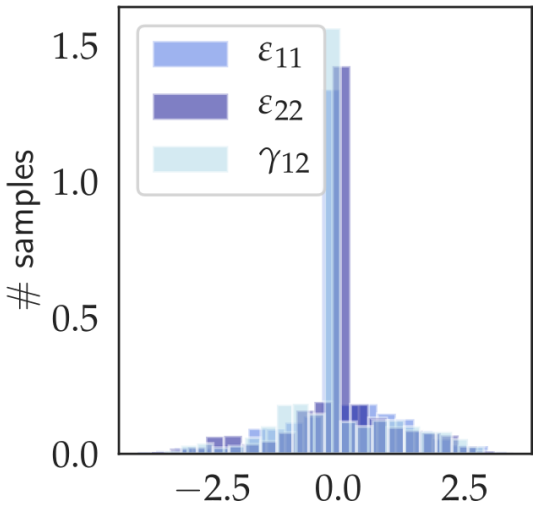
vectorial: $$\bar{x}_{kj} = \frac{x_{kj} - \beta_j}{\alpha_j} \qquad \alpha_j = \underset{k}{\mathrm{std}}(x_{kj}), \quad \beta_j = \underset{k}{\mu}(x_{kj})$$

scalar: $$\bar{x}_k = \frac{x_k - \beta}{\alpha} \qquad \alpha = \max_k(x_k), \quad \beta = 0$$

x : physical



$\bar{x}$: normalized



Normalization inputs/outputs

vectorial: $$\bar{x}_{kj} = \frac{x_{kj} - \beta_j}{\alpha_j} \qquad \alpha_j = \underset{k}{\mathrm{std}}(x_{kj}), \quad \beta_j = \underset{k}{\mu}(x_{kj})$$

scalar: $$\bar{x}_k = \frac{x_k - \beta}{\alpha} \qquad \alpha = \max_k(x_k), \quad \beta = 0$$

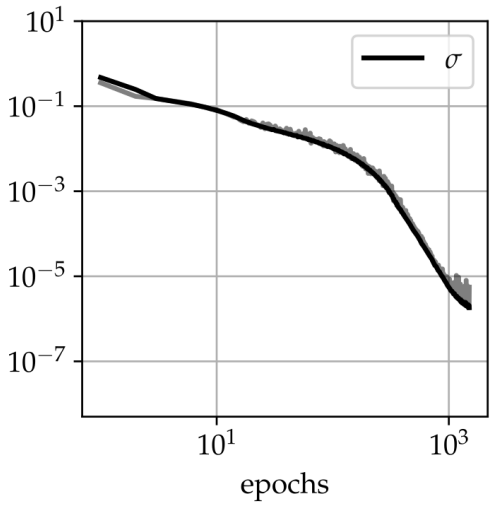
Hint

```
prm_E = [alpha_E, beta_E] # strain
prm_S = [alpha_S, beta_S] # stress
prm_F = [alpha_F, beta_F] # free energy

params = [prm_E,prm_S,prm_F]


– Nnet.SetParams(params)

– TANNnet.SetParams(params)
```

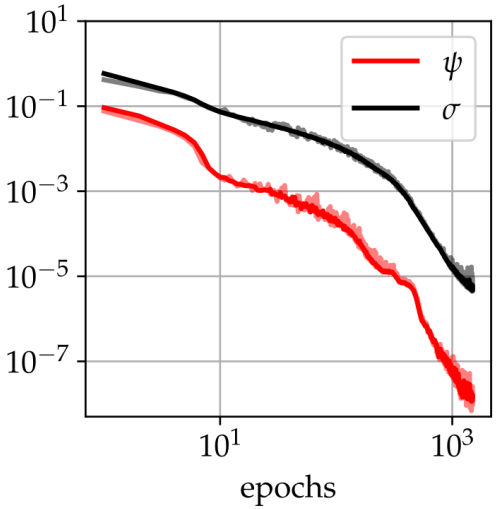
Results

Training curves

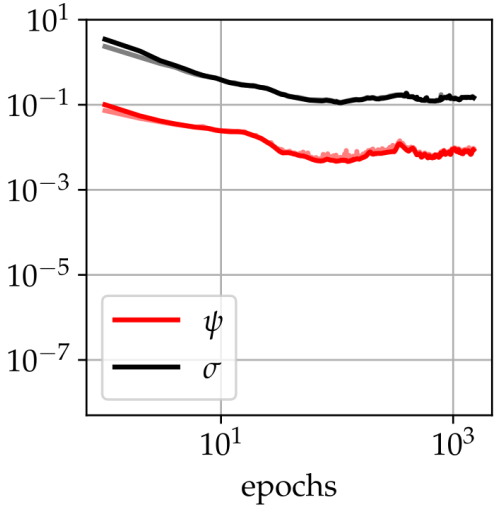
$\boldsymbol{\sigma} = \mathrm{NN}(\boldsymbol{\varepsilon})$ tanh



$\boldsymbol{\sigma} = \mathrm{TANN}(\boldsymbol{\varepsilon})$ GELU



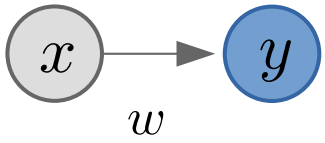
$\boldsymbol{\sigma} = \mathrm{TANN}(\boldsymbol{\varepsilon})$ ReLU



- NN and TANN (GELU) comparable accuracy in stress, but..
- TANN (ReLU) not training

Vanishing gradients (1/2)

Vanishing gradients – first order

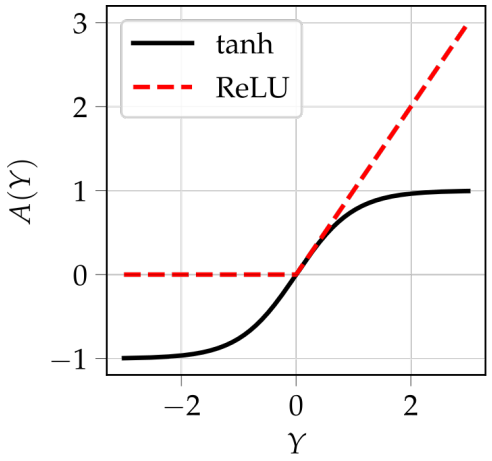


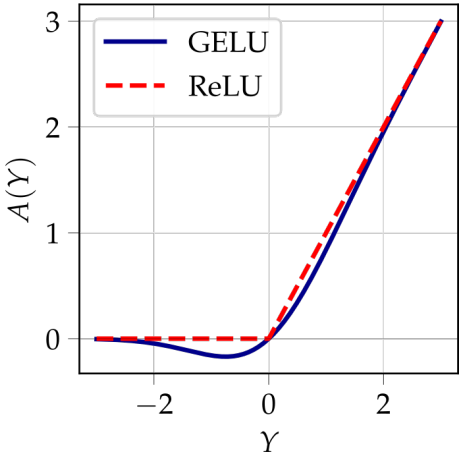
output: $y = \mathcal{A}(wx)$

$$w^{(\text{next step})} = w - \epsilon \frac{\partial \ell}{\partial w}$$

loss: $\ell = \underbrace{\|y - y_{\text{label}}\|}_{\ell_y}$

$$\frac{\partial \ell}{\partial w} = (x \; \partial_x \mathcal{A})$$



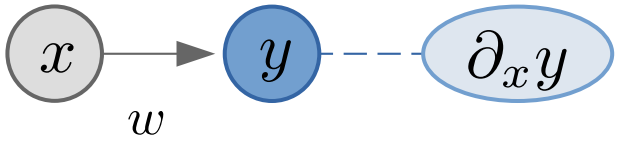


if $\partial_x \mathcal{A} = 0$

then ℓ_y is not minimized

Vanishing gradients (2/2)

Vanishing gradients – higher orders



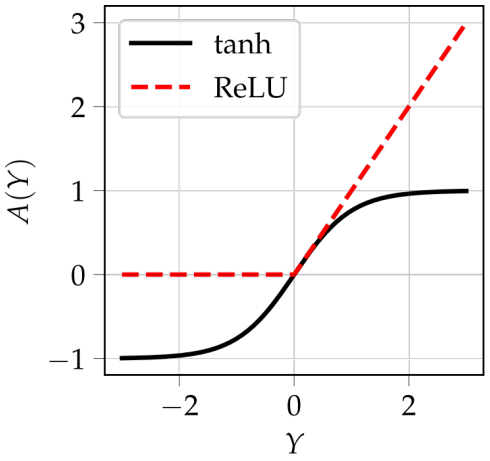
output: $y = \mathcal{A}(wx)$

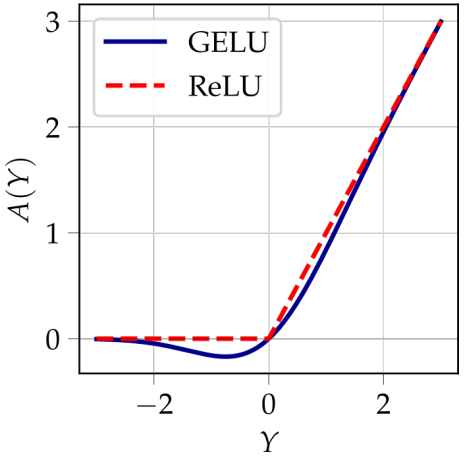
derivative: $\partial_x y = w\, \partial_x \mathcal{A}$

$$w^{(\text{next step})} = w - \epsilon \frac{\partial \ell}{\partial w}$$

loss: $\ell = \underbrace{\|y - y_{\text{label}}\|}_{\ell_y} + \underbrace{\|\partial_x y - y'_{\text{label}}\|}_{\ell_{y'}}$

$$\frac{\partial \ell}{\partial w} = (x\, \partial_x \mathcal{A}) + \left(x\, \partial^2_{x^2} \mathcal{A} + \partial_x \mathcal{A}\right)$$





if $\partial^2_{x^2} \mathcal{A} = 0$

then $\ell_{y'}$ is not minimized

Exercise #2

Exercise #2

Use the trained networks (NN and TANN) at inference:

- predict the material behavior for unseen loading paths

```
./workshop/elasticity/TANN_inference
./workshop/elasticity/NN_inference
```

Hint

define a function prediction(inputs)

either

inside the (TA)NN class

```
def prediction(self, inputs):
    if self.material == 'elastic':
        strain = inputs
        …
        # self.predict_on_batch()
        …
    return stress
```

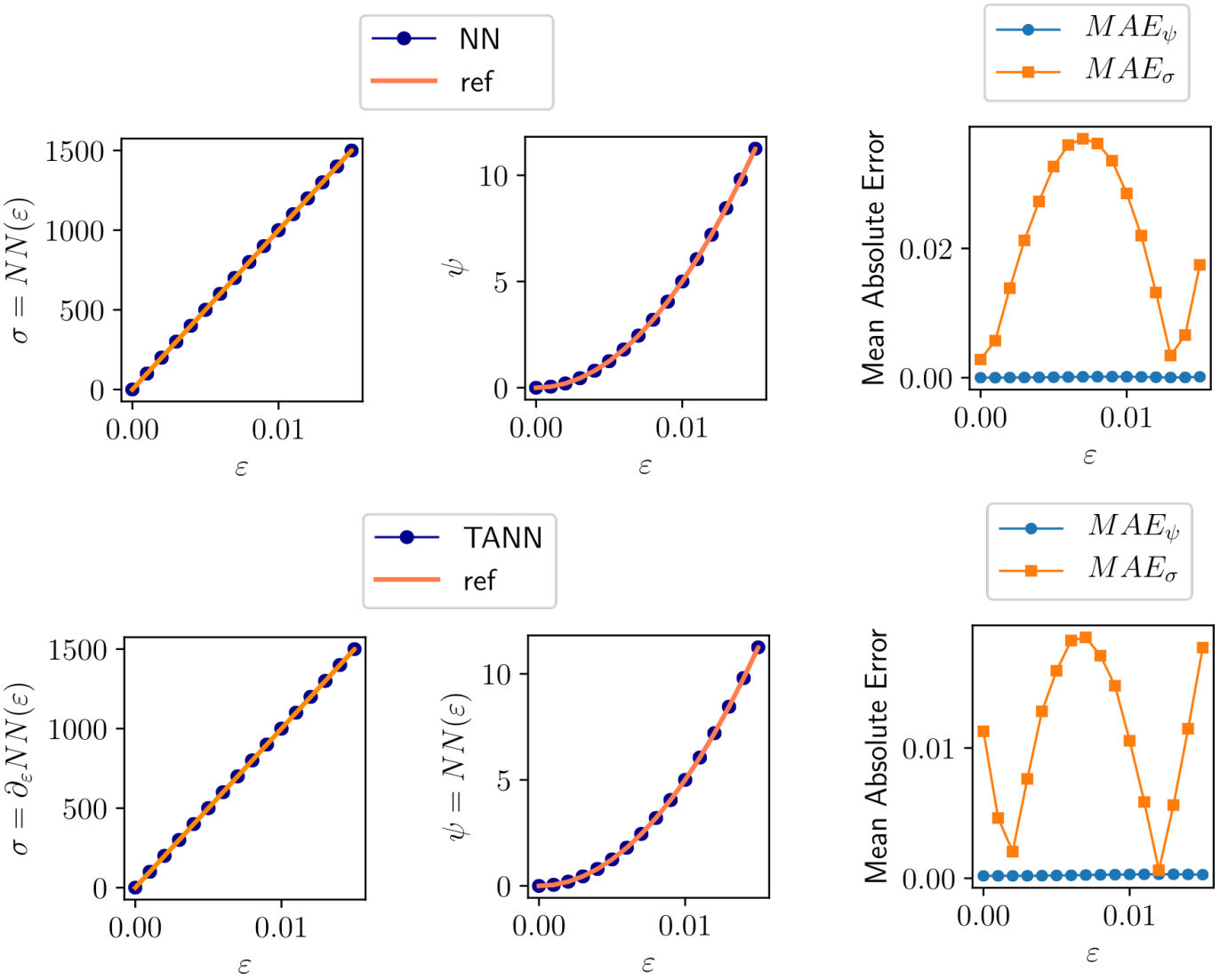
or

directly in the notebook

```
def prediction(strain):
    …
    # model.predict_on_batch()
    …
    return stress
```

and call it recursively for each strain value (or increment)

Results

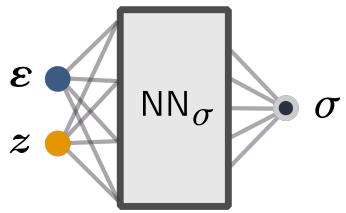

NN
ref
$\sigma = NN(\varepsilon)$
ψ
ε
MAE_ψ
MAE_σ
Mean Absolute Error
TANN
ref
$\sigma = \partial_\varepsilon NN(\varepsilon)$
$\psi = NN(\varepsilon)$
ε
MAE_ψ
MAE_σ
Mean Absolute Error
ε

Exercise #3

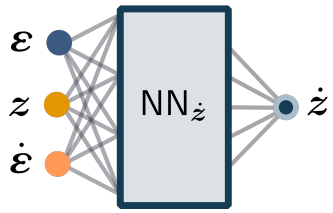
Neural Networks for constitutive modeling – general case

Three-dimensional elasto-plasticity (Von Mises, plane strain)

black-box



evolution equation

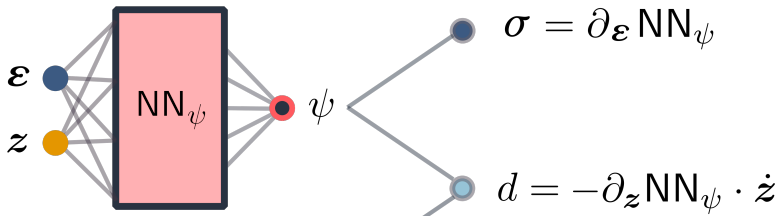


$$\underbrace{\|\boldsymbol{\sigma} - \mathsf{NN}_{\boldsymbol{\sigma}}\|}_{\text{loss in } \boldsymbol{\sigma}} + \underbrace{\|\dot{\boldsymbol{z}} - \mathsf{NN}_{\dot{\boldsymbol{z}}}\|}_{\text{loss in } \dot{\boldsymbol{z}}}$$

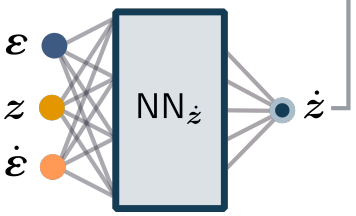
Train the network(s)

- fixed number of epochs and mini batch size

thermodynamics



evolution equation



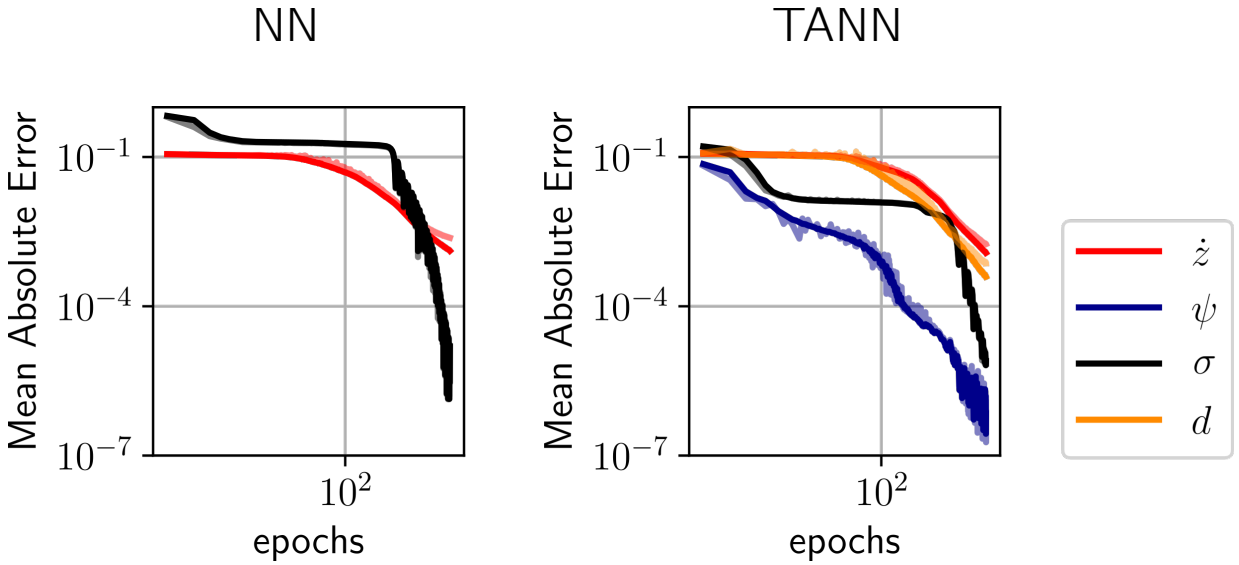
$$\underbrace{\|\dot{\boldsymbol{z}} - \mathsf{NN}_{\dot{\boldsymbol{z}}}\|}_{\text{loss in } \dot{\boldsymbol{z}}} + \underbrace{\|\psi - \mathsf{NN}_{\psi}\|}_{\text{loss in } \psi} +$$

$$\underbrace{\|\boldsymbol{\sigma} - \partial_{\boldsymbol{\varepsilon}} \mathsf{NN}_{\psi}\|}_{\text{loss in } \boldsymbol{\sigma}} + \underbrace{\|d + \partial_{\boldsymbol{z}} \mathsf{NN}_{\psi} \cdot \dot{\boldsymbol{z}}\|}_{\text{loss in } d}$$

`./workshop/plasticity/TANN_training`
`./workshop/plasticity/NN_training`

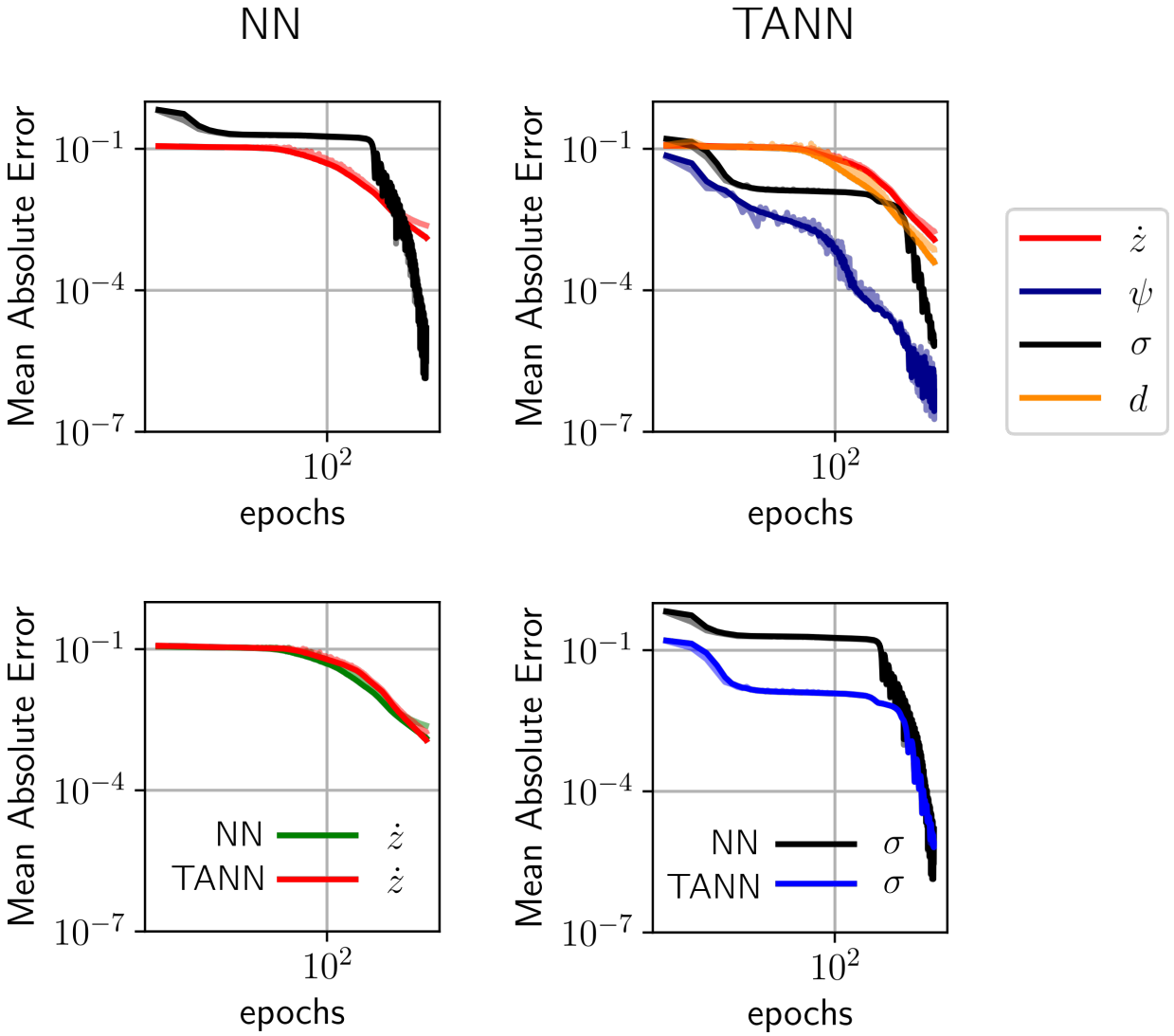
Results

Training curves



Results

Training curves



Exercise #4

Exercise #4

Use the trained networks (NN and TANN) at inference:

- predict the material behavior for unseen loading paths

```
./workshop/plasticity/TANN_inference
./workshop/plasticity/NN_inference
```

Hint

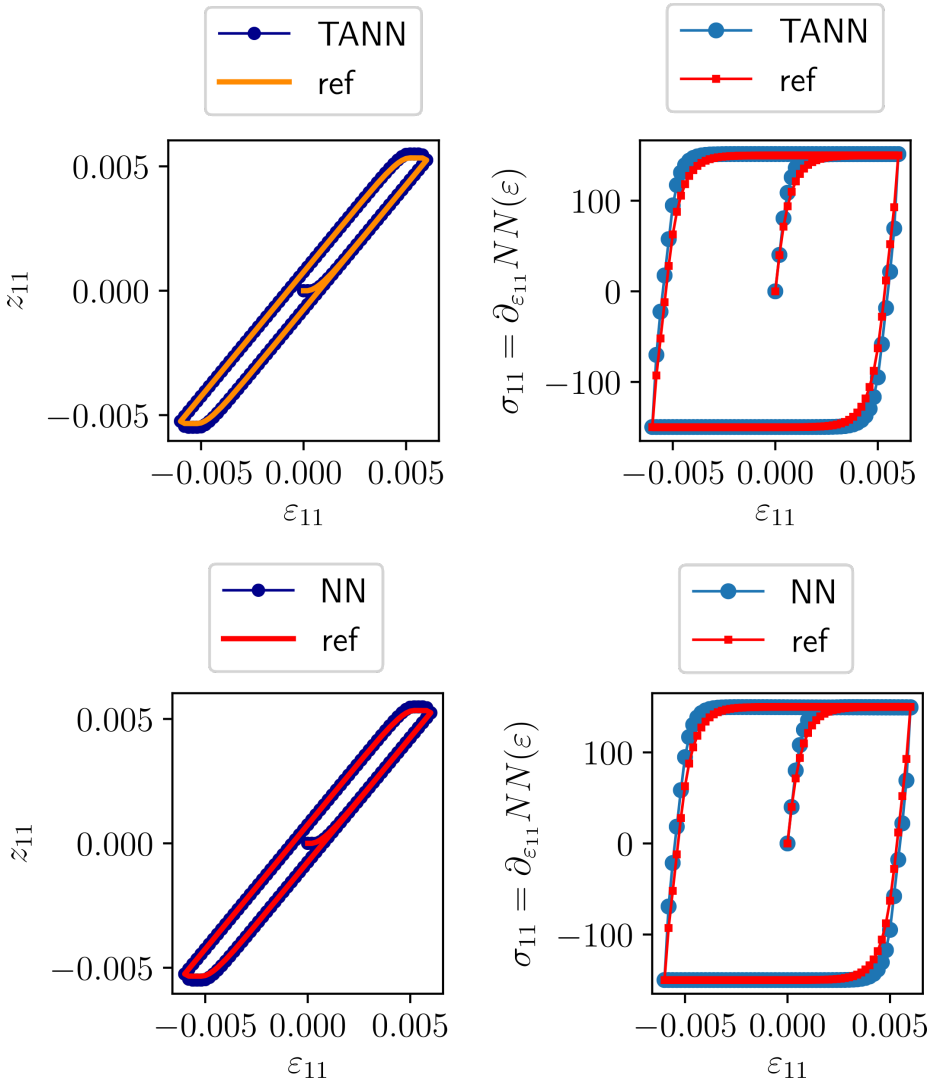
define a function prediction(inputs) inside the (TA)NN class

```
def prediction(self, inputs):
    if self.material == 'plastic':
        …
        # self.predict_on_batch()
        …
    return stress,int_svars
```

and call it recursively for each strain value (or increment)

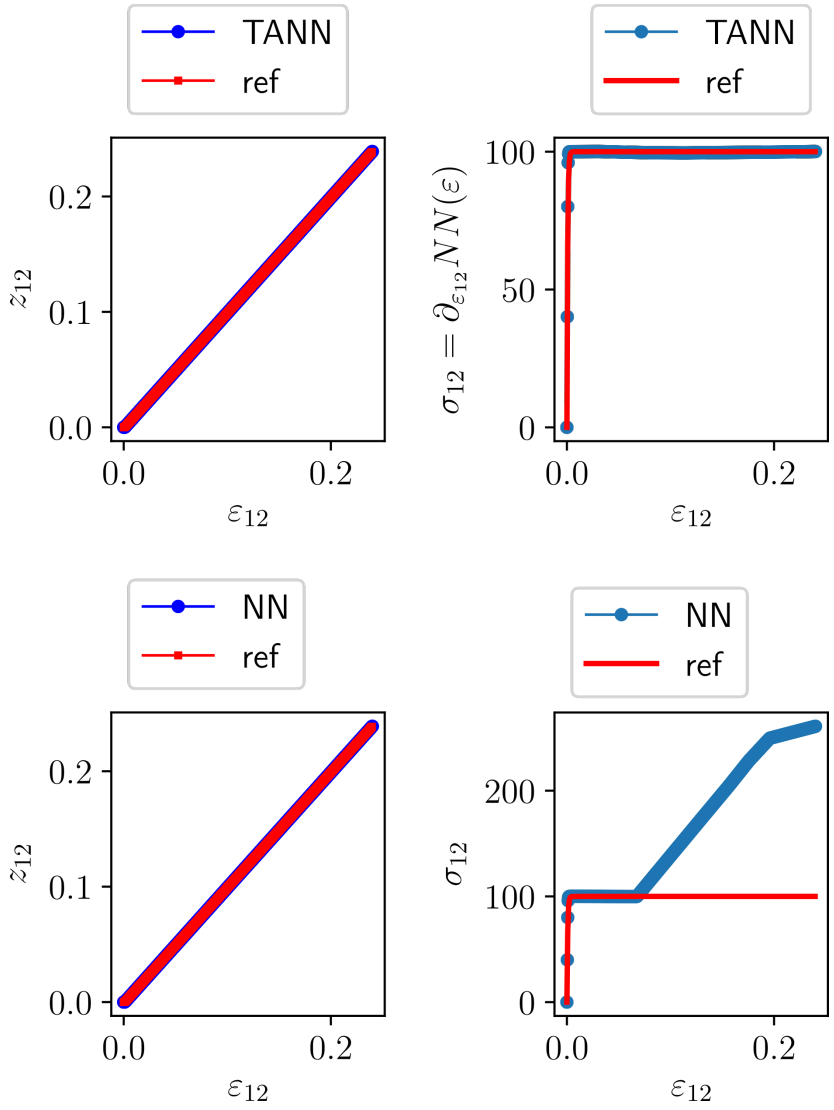
Results

Training curves



Results

Training curves



Exercise #5

Exercise #5

Use the trained networks (NN and TANN) at inference:

- Compute the Jacobian

$$\mathbb{J}_{ijkl} = \frac{\partial \dot{\sigma}_{ij}}{\partial \dot{\varepsilon}_{kl}} \approx \frac{\partial \Delta \sigma_{ij}}{\partial \Delta \varepsilon_{kl}}$$

Hint

use `tf.gradients()` or `gradientTape()`

https://www.tensorflow.org/api_docs/python/tf/gradients
https://www.tensorflow.org/api_docs/python/tf/GradientTape

Exercise #6

Exercise #6

Is the energy predicted by TANN convex? (it should for the material at hand)

- Check!

$$\mathbb{J}_{ijkl} = \frac{\partial \dot{\sigma}_{ij}}{\partial \dot{\varepsilon}_{kl}} \approx \frac{\partial \Delta \sigma_{ij}}{\partial \Delta \varepsilon_{kl}}$$

Hint

use `tf.gradients()` or `gradientTape()`

https://www.tensorflow.org/api_docs/python/tf/gradients
https://www.tensorflow.org/api_docs/python/tf/GradientTape

To summarize

What have we learned?

- Neural networks for the constitutive modeling of materials
  - In elasticity but also in plasticity
- NN may give accurate results for very simple cases (elasticity), but will **never guarantee** us that the laws of thermodynamics will be satisfied!
- TANN enforce, **by construction**, the laws of thermodynamics!
- Be careful about vanishing gradients
- **TANN > NN** even for a simple elasto-plastic material

5th workshop CSMA Junior

Mai 14th-16th 2022

Île de Porquerolles

Deep learning, real-time simulation and model-order reduction

Application to the constitutive modeling of materials

Practical session

Filippo Masi

filippo.masi@ec-nantes.fr

Nantes Université, École Centrale Nantes, CNRS, GeM