Machine Learning in Computational Mechanics

Task sheet 1

Group: Cruel Winter

Haotian Wu, 504641, haotian.wu@campus.tu-berlin.de

Shijie Xu, 505319, s.xu@campus.tu-berlin.de

Zhen Duan, 498761, z.duan@campus.tu-berlin.de

Xilin Li, 498754, xilin_li@tu-berlin.de

Aufgabe 2: Introduction to PyTorch

The figure 1 shows the reference configuration and the deformed configuration as scatter plots, where the deformed configuration is coloured according to the Euclidean norm of the displacement field. The displacement field vectors are as follows:

$$\mathbf{u}(x, y) = \begin{bmatrix} 0.2 \cdot \sin(3 \cdot x) + 0.1 \cdot y^2 \\ 0.05 \cdot x^2 + 0.3 \cdot \cos(2 \cdot y) \end{bmatrix}$$

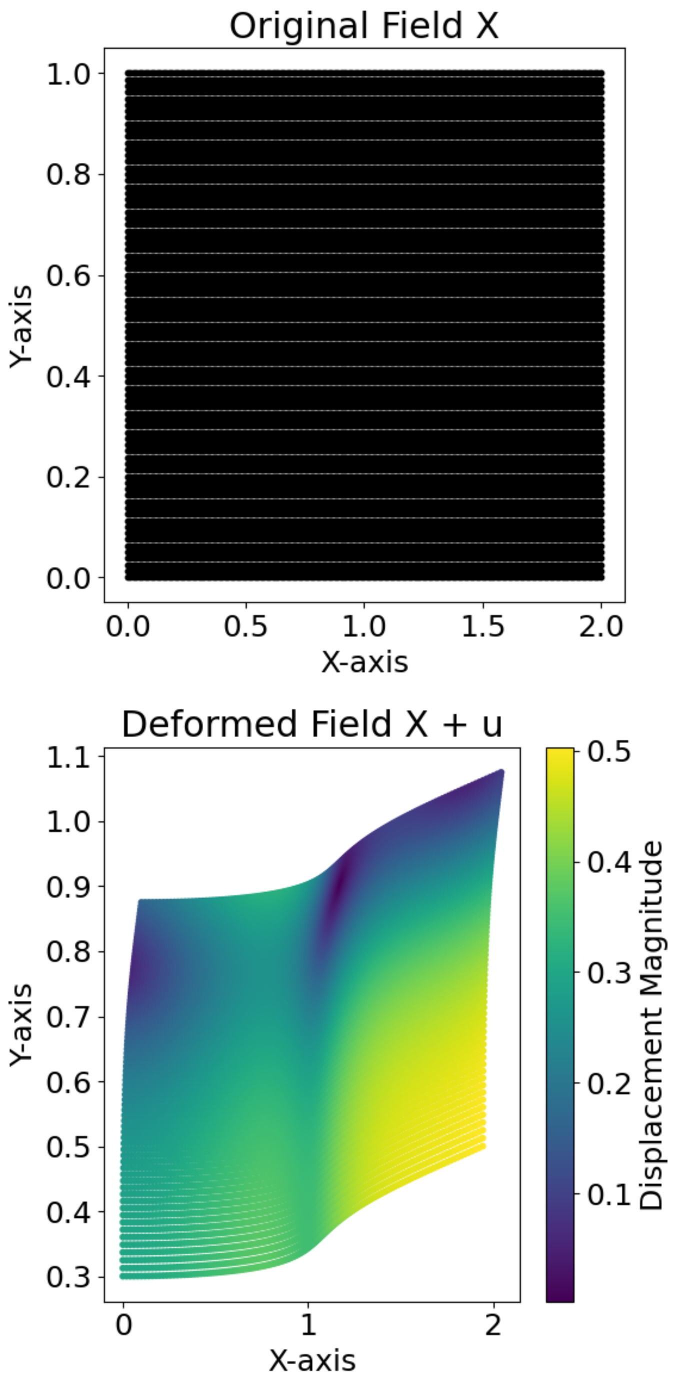

Figure 1: Undeformed and deformed configuration with coloring

Aufgabe 3: PyTorch for Linear Regression

This code uses **PyTorch** to build a simple single-layer linear neural network and trains the network to fit a set of nonlinear data with noise. The network parameters are optimised using **stochastic gradient descent (SGD)** to ultimately achieve the best fit of the model to the data.

Code Structure

1. Define the Neural Network Model
2. Define the Optimizer and Loss Function
3. Generate Data
4. Train the Model
5. Visualize the Training Process
6. Analyse the result

Explanation

1 Define the Neural Network Model

First, a simple single-layer neural network model is defined. The network model `Net` inherits from PyTorch's `nn.Module` class, and its structure can be represented as:

$$y = wx + b$$

where:

- w: The weight of the model, used to linearly map the input to the output.
- b: The bias term, used to adjust the baseline of the output.

The model has only one linear layer, with one input feature and one output feature, thus it is a **linear regression model**.

2 Defining the Optimizer and Loss Function

Loss function and **optimizer** are used to train the model.

2.1 Loss Function

The **Mean Squared Error (MSE)** is used to measure the difference between the model's output and the target output. The formula is defined as:

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where:

- y_i: The true value.
- $\hat{y}_i$: The predicted value from the model.
- n: The number of samples.

2.2 Optimizer

The **Stochastic Gradient Descent (SGD)** optimizer is used to minimize the loss function, with a learning rate of 0.2. The update rule for the optimizer is:

$$W^{(t+1)} := W^{(t)} - \eta \nabla_W \mathcal{L}(W^{(t)})$$

where:

- $W^{(t)}$: The weight at the t-th iteration.
- η: The learning rate (0.2 in this case).
- $\nabla_W \mathcal{L}(W^{(t)})$: The gradient of the loss function with respect to the weight at iteration t.

3 Generating Data

Next, a set of nonlinear data is generated, which will serve as the input and target output for the neural network. - 100 random input values x and corresponding output values y are generated. The output values are determined by the following complex nonlinear function:

$$y = \sin(x) \cdot x^3 + 3x + \epsilon$$

where ϵ is a random noise term in the range [0, 0.8), used to simulate real-world data uncertainty. The roles of each term in the formula are:

- $\sin(x)$: Introduces periodic variation.
- x^3: Adds nonlinearity.
- $3x$: Adds a linear term.
- ϵ: Simulates noise and uncertainty in the data.

4 Training the Model

The training process is carried out using **gradient descent** to update the model parameters. The detailed steps are as follows:

4.1 Forward Propagation:

Pass the input x through the network to obtain the predicted output $\hat{y}$.

$$\hat{y} = wx + b$$

4.2 Calculate Loss:

Use MSE to calculate the difference between the predicted value $\hat{y}$ and the true value y.

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

4.3 Backpropagation:

Calculate the gradient of the loss function with respect to the model parameters.

4.4 Update Parameters:

Update the model parameters using SGD based on the computed gradients:

$$\theta = \theta - \alpha \cdot \nabla_\theta J(\theta)$$

where:

- θ: The model parameters (e.g., w and b).
- α: The learning rate (0.2 in this case).
- $\nabla_\theta J(\theta)$: The gradient of the loss function $J(\theta)$ with respect to the parameters θ.

4.5 Iterate Training:

Repeat the above steps for 250 iterations to progressively reduce the loss and allow the model to better fit the data.

5 Visualizing the Training Process

To better understand the training process, the model's fitting effect is visualized every 10 iterations. The visualization includes:

- **Scatter Plot**: Displays the real data points to observe the original data distribution.
- **Fitting Curve**: Shows the model's predictions as a red curve to visualize the fitting effect.
- **Loss Value**: Displays the current loss value to observe the convergence of the model during training.

6 Analyse the Result

6.1 Test with Original Function

In Figure 2, we present the results of applying a linear regression neural network to the original function:

$$y = \sin(x) \cdot x^3 + 3x + \text{noise}$$

Over the training, the loss value decreased and eventually stabilized at **0.0738**, indicating that the model has effectively learned to approximate the underlying function. The visualization shows that the trained neural network's predictions align closely with the actual data points, demonstrating a reasonable fit.

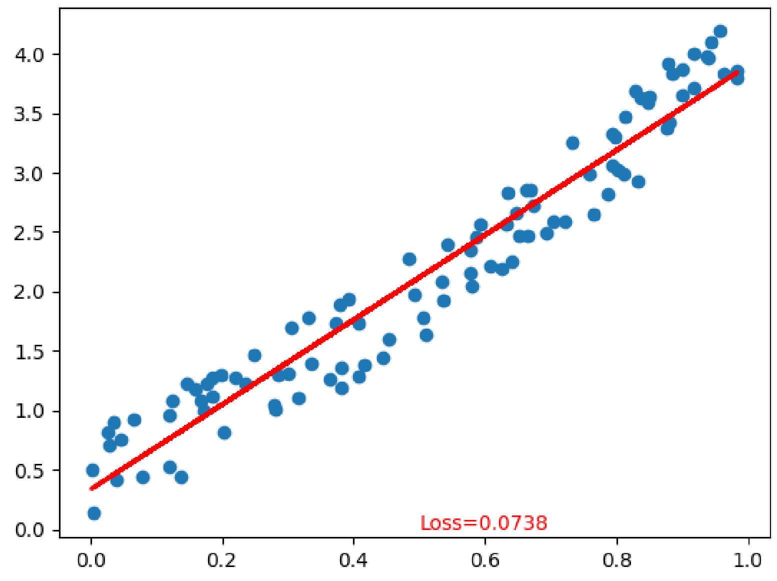

Figure 2: Results of the linear regression on the original function

6.2 Test with second Function

Next, we modified the code with a different function for y:

$$y = \text{np.random.rand}(100) \times 10$$

The results are shown in Figure 3. The loss value remained nearly constant around **7.2972** throughout the training process. This high and stable loss value suggests that the linear model is not well-suited to capture the random nature of the data generated by this function. Since the data is essentially random noise scaled up, the model cannot find a meaningful pattern to learn from, resulting in poor predictions.

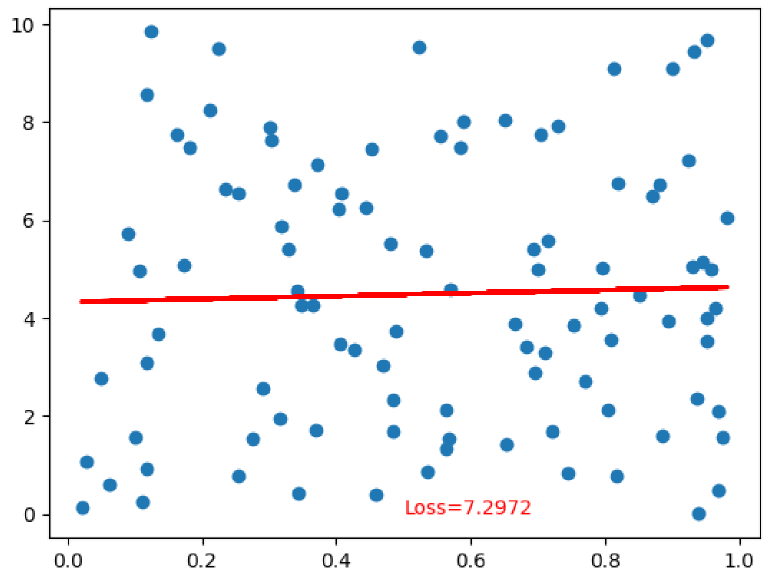

Figure 3: Results of the linear regression on the second function

6.3 Test with third Function

Finally, we tested the code with the function of our choice:

$$y = \sin(x) \cdot \cos(2x)$$

Results are depicted in Figure 4. Although the loss value decreased to a final value of **0.0166**, indicating that the model was optimized in terms of minimizing the loss, the visualization clearly shows that the linear model failed to capture the complex non-linear pattern present in the data.

The red line, representing the model's predictions, approximates the data with a simple linear relationship, which does not accurately fit the wave-like characteristics generated by the product of sine and cosine functions.

This suggests that, despite the relatively low loss value, the model is underfitting the data, and a more complex model would be required to effectively learn the intricate non-linear relationship.

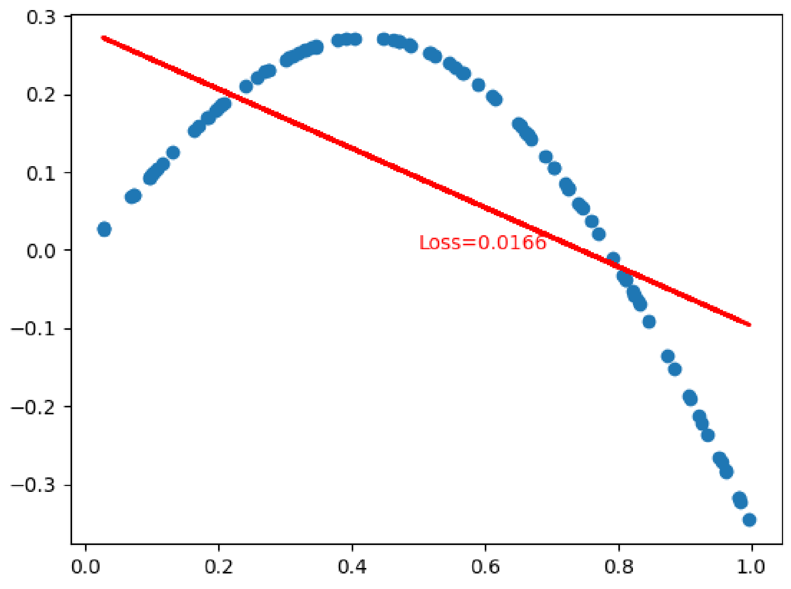

Figure 4: Results of the linear regression on the custom function

6.4 Comparison and Summarize

Although both functions are non-linear, the linear regression model was more successful in fitting the first function

$$y = \sin(x) \cdot x^3 + 3x + \text{noise}$$

compared to the third function

$$y = \sin(x) \cdot \cos(2x)$$

The first function contains a dominant linear term $3x$, which has a much greater influence on the output compared to the non-linear component $\sin(x) \cdot x^3$.

This is because in the interval $(0, 1)$, $3x$ increases steadily and contributes significantly to the overall value, whereas x^3 grows much slower than $3x$, and $\sin(x) \cdot x^3$ is limited by the small values of $\sin(x)$ and x^3, causing the impact of the non-linear term to be minimal.

This makes the overall function appear mostly linear, making it easier for the linear model to fit.

In contrast, the third function is highly non-linear and oscillatory due to the product of sine and cosine, which results in complex wave patterns that a simple linear model cannot capture, leading to poor fitting performance.

Thus, the dominant linear component in the first function allows the model to approximate it, whereas the third function's complexity makes it unsuitable for linear modeling.