Institut für Mechanik
FG Strukturmechanik und Strukturberechnung

Machine Learning in Computational Mechanics

1. Hausaufgabe

Vorgelegt von: Elvis Sattler (404043)
Mert Akdemir (393309)

Abgabe: 06.05.2024

Contents

List of Figures

1 Introduction to PyTorch

In Fig.(1.1) is a reference configuration and a deformed configuration shown via a scatterplot, while the deformed configuration is colored according to the euclidian norm of the displacement field. The displacement field is given as follows:

$$u_1 = 0.1 \cdot \log(0.3 + x) + sinh(y), \tag{1.1}$$

$$u_2 = 0.8x^2 \tag{1.2}$$

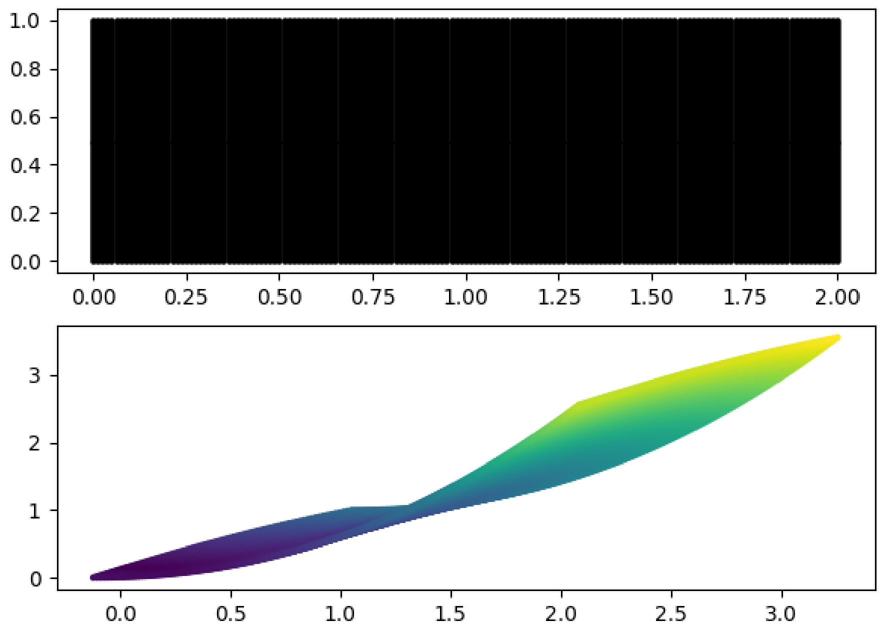

Figure 1.1: Undeformed and deformed configuration with coloring

2 PyTorch for Linear Regression

Linear regression with a single layer neural network is implemented using PyTorch to get the best fitting linear function regarding the nonlinear noisy sample points. The implementation can be broken down to the following key steps:

1. Setup neural network (input (x_n), layer (weights and biases), output (y_n))
2. define optimizer (Stochastic Gradient Descent, SGD (compare Eq.(2.1)))
   with newton raphson (compare Eq.(2.2)) and
   loss function (Mean-Square-Error, MSE compare Eq.(2.4))

$$W^{(t+1)} := W^{(t)} - \eta \nabla_w \mathcal{L}(W^{(t)}) \tag{2.1}$$

$$x_{n+1} := x_n - \frac{f'(x_N)}{f''(x_N)} \tag{2.2}$$

$$l(x, y) = L = \{l_1, ..., l_N\}^T \tag{2.3}$$

$$l_n = (x_n^2 - y_n^2) \tag{2.4}$$

3. create loop with appropriate amount of iterations
4. input data „x“ is fed into the neural network to get predictions
5. the MSE loss ist computed between the predictions and the actual outputs
6. gradients of the loss with respect to the network parameters are computed using backpropagation
7. optimizer updates the weights of the neural network using the computed gradients
8. plotting the original data points alongside the current predictions, while also displaying the loss value on the plot
9. show the final plot with the learned curve

In Fig.(2.1) are the results of the linear regression neural network presented. The result is a trained neural network that approximates the underlying function that generated the training data.
With each iteration, the loss value decreases, indicating that the network is improving its approximation of the underlying function. Over the last three iterations, the loss value stabilizes at 0.0628, suggesting that the model has reached a stable point and further

training may not yield significant improvement. Based on visualization, the trained neural network appears to reasonably approximate the underlying function.

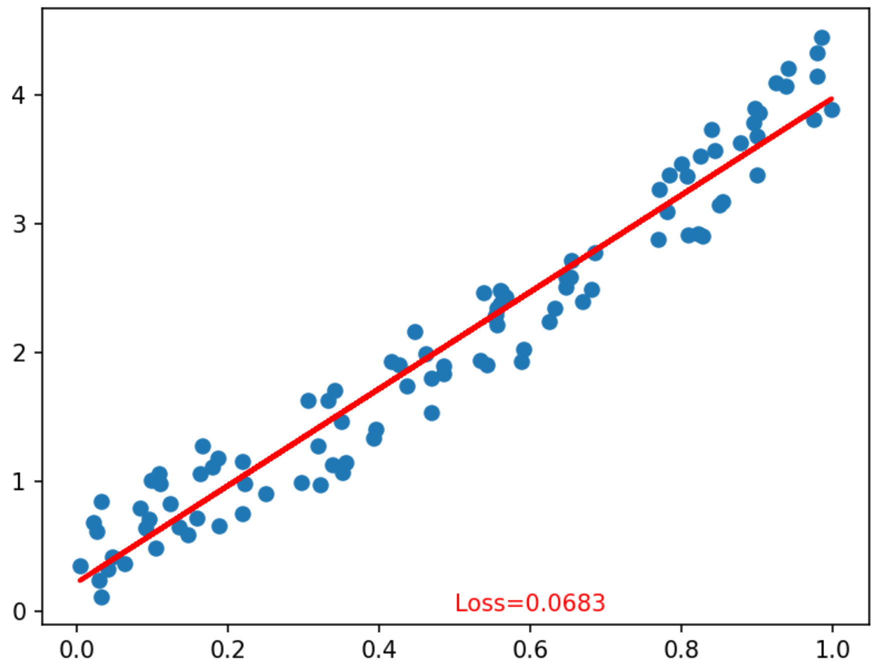

Figure 2.1: Result of linear regression and final loss value

As the next step the code will be runned with a different function y, which is the following:

$$y = np.random.rand(100) * 10 \tag{2.5}$$

The source code with the new function produces the result shown in Fig.(2.2). The loss value remains nearly constant at around 6.253. However, the loss value continues to decrease, albeit very slowly (from 6.2539 to 6.2529). This indicates that the model is making minor adjustments to its parameters to better fit the training data, but the changes are small and incremental. Furthermore the loss value is very high which indicates that the predictions are far form the actual values on average.

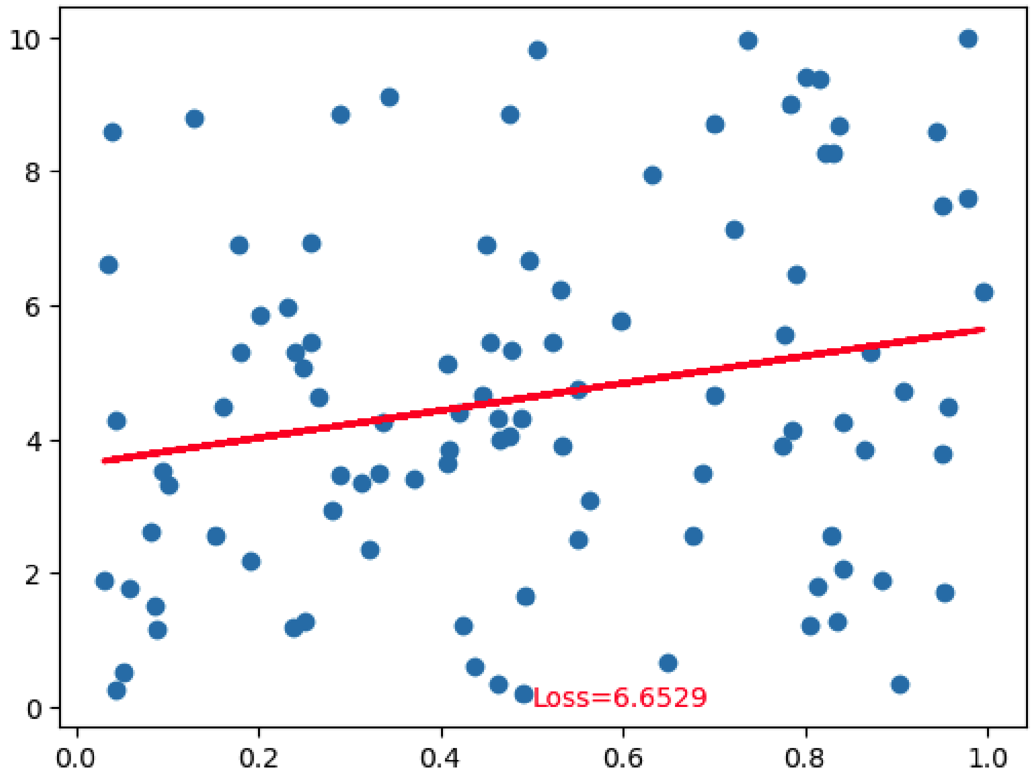

Figure 2.2: Result of linear regression and final loss value for the second function

In Figure 2.3, we depict the results of applying linear regression to a sine function (2.6). The loss value decreases from 0.3412 to 0.2126, indicating an enhancement in the neural network's alignment with the data. However, even with this decrease, the final loss value of 0.2126 remains relatively high. This elevation might be ascribed to the inherent noise inherent in the sine function.

$$y = sin(2 * \pi * x) \tag{2.6}$$

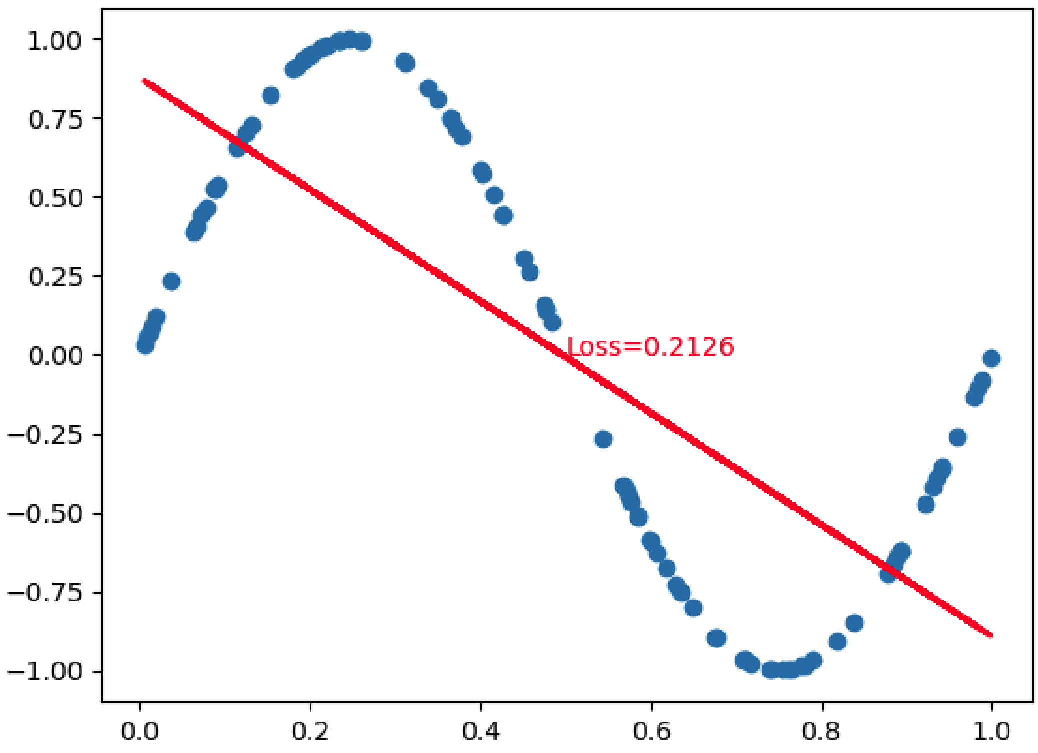

Figure 2.3: Result of linear regression and final loss value for the third function