First Steps towards Machine-Learning supported Material Parameter Identification

David Koch & André Haufe

DYNAmore GmbH

1 Introduction

Machine learning is becoming more and more part of our world. Even though most people have so far only passively used the possibilities of this technology, e.g. for search queries or product recommendations, many have surely already thought about how these new possibilities could support their work in the future.
In this contribution, it is investigated if machine learning is suitable to support the process of material characterization. Through deep neural networks it is possible to "learn" nonlinear relationships between a set of input values and the corresponding output, also known as labels. As a proof of concept, it is examined whether the shape of the yield curve can be predicted based on force-displacement curves from simulated tensile tests. So, in a first step, a large number of tensile tests are simulated which differ in the shape of the yield curve. Here, for the description of the yield curve an approach according to Hockett-Sherby was used which provides 4 parameters for the definition of the shape. The force-displacement curves of these tests are used as the input and the parameters of the yield curve as labels. By considering the entire realistic range of all four parameters, the trained neural network should be able to provide the best matching set of parameters for a given force-displacement curve. For the prediction, of course, the initial and boundary conditions must be the same when generating the force-displacement curve, whether by simulation or in a real test. Of course, all initial and boundary conditions as well as all other assumptions and simulation settings are also learned from the neural network. Therefore a change of these parameters can for sure worsen the predictions considerably and can make a re-learning process inevitable.
The long-term objective of this method and the vision of this work are to learn the possible spectrum of the whole material model in advance in order to be able to finally predict the material properties based on only a few experiments with minimal effort.

2 Artificial neural networks

“Artificial intelligence" (AI) is a domain of computer science that deals with methods of automating intelligent behavior and machine learning. Although the term AI is currently clearly overused, the availability of large computing capacities has led to a wide range of possible applications of methods that can be assigned to AI.
In the field of machine learning, artificial neural networks are used to map relationships and dependencies between certain input and output variables. In addition to the input and output layers, neural networks can also have additional intermediate layers, so-called hidden layers. As the number of hidden layers and their neurons increases, more complex relationships can be mapped more efficiently. Deep learning means that several of these hidden layers are used, however, there is no universal definition what the minimum number of hidden layers is.
In the present case, the artificial neural network is a so-called feedforward neural network (FFNN), which does not allow loops. The neurons in the input layer represent the individual input values. Subsequently, each neuron of a layer is linked to each neuron of the next layer. Multiplication by weights and addition of biases controls the influence of each neuron on the neurons, represented by real numbers, of the next layer. Learning the network therefore consists in finding the weights and values of the biases in such a way that the values of the output neurons correspond as closely as possible to the labels of the data to be learned.
Since at this point it shall be investigated whether at least parts of the material behavior can be predicted by a neural network in the form of individual material parameters, test results labelled with the corresponding correct material parameters must be used as input data. Depending on the output, classification and regression problems can be distinguished. Since in this case the output is a continuous quantity and not a label of discrete classes, this is a regression problem. A gradient method is used for the training, which is described in more detail below.

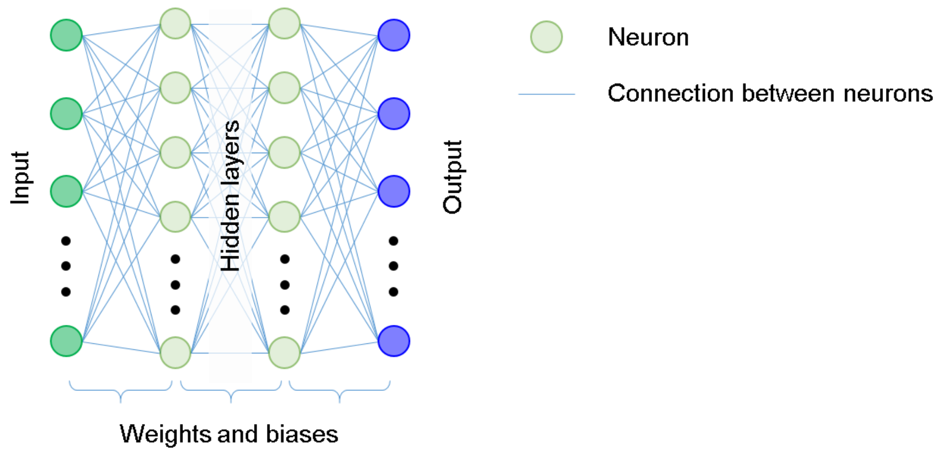

Fig.1: Structure of an artificial neural network

3 Data generation

The required training data was generated by simulations using a simple finite (shell-) element discretization using a standard von Mises plasticity model in LS-DYNA. At this point, the investigated simulation runs differ only in the shape of the yield curve which is represented by a Hockett-Sherby approach:

$$\sigma_y^{true} = \sigma_0 + B\left(1 - e^{-C\left(\varepsilon_{true}^{pl}\right)^N}\right) \tag{1}$$

Therein, σ_0, B, C, and N are the four parameters that define the shape of the yield curve. A range from 0.1 to 1.0 was considered for all parameters. Afterwards, a full-factorial approach was used to obtain parameter sets within the entire range of the parameter space. This means that for each parameter its range was covered by n supporting points. In the case of four parameters, this leads to a total number of n^4 simulations which had have to be run to generate the input data. Apart from the yield curve, a simple *MAT_PIECEWISE_LINEAR_PLASTICITY (*MAT_024) material card was used with standard parameters for steel (rho=7.85e-06 kg/mm^3, e=210 GPa, nu=0.3). Neither strain rate dependence nor failure were considered in this study.
In order to reduce the time required for the simulations, the geometry of the tensile specimen is discretized with 8 elements (see Fig. 2) only. In the context of a feasibility study, this reduction should certainly be permissible.

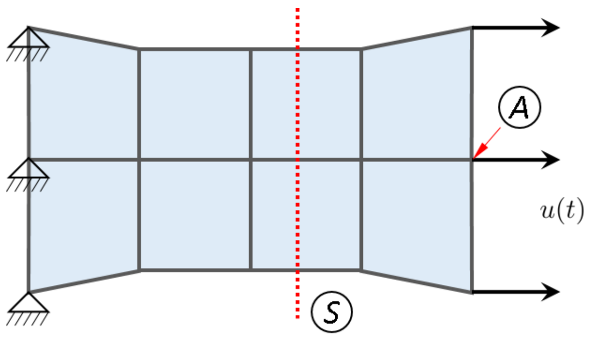

Fig.2: Boundary value problem for data generation

The data that are actually used as input are the evolution of the force at the intersection *S* for certain displacement values at point *A*. The same abscissa positions (displacements) were evaluated for all training sets. Thereby the distance between the evaluated points was reduced in the area with a tendency to higher curvature. This is exemplarily shown in Fig. 3 for one curve. Thus 25 input values were identified for each input data set. Each input set was labelled with the four parameters used for its calculation.

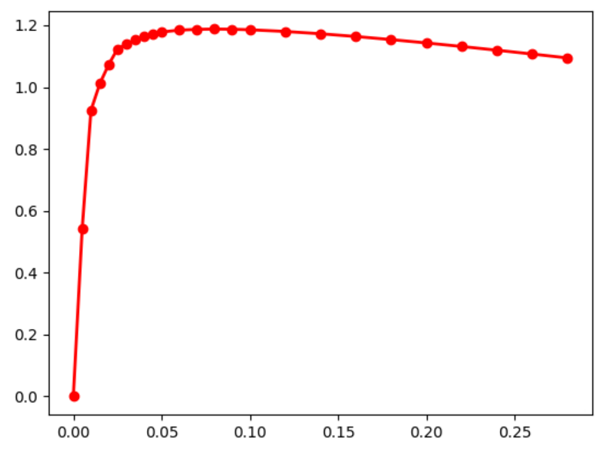

Fig.3: Distribution of data points on the force-displacement curves

With $n = 12$ a total of 20736 tensile tests were simulated. For performance reasons, all tensile tests were calculated in a single simulation run, since for such small examples the initialization time for LS-DYNA is significant larger than the bare computing time. In this case this would have increased the total duration from approx. 12 minutes to more than 12 hours. Unfortunately, however, the high number of "*DEFINE_CURVE"s (the yield curves) means that the demand for memory allocation is growing linearly with the test number as well, so that for the time being only 12 support points per parameter are examined.

4 Training

The implementation was completely realized in Python with the help of TensorFlow [1] and Keras [2]. These packages make it very convenient to create an artificial neural network and train it on the basis of existing data.
Since the correct output parameter set is known for each input, we speak here of a so-called supervised learning. A backpropagation algorithm is used to optimize weights and biases [3][4]. This algorithm is a special form of a general gradient method based on the mean square error.
For this proof-of-concept study, only a small FFNN was used with a single hidden layer consisting of 10 neurons. Thus, the entire FFNN consisted of 3 layers with 25, 10 and 4 neurons.
In analogy to the biological equivalent, activation functions are usually used in artificial neural networks to represent the activation of a neuron. This function provides the input for each neuron by a corresponding mapping of the sum of the outputs of the neutrons of the previous layer. One of the most frequently used activation functions is the rectifier

$$f(x) = \max(0, x) \,, \tag{2}$$

that was introduced in 2000 by Hahnloser et al. [5]. A unit using a rectifier is also called a rectified linear unit (ReLU). ReLU has meanwhile become the most frequently used activation function for several applications and has replaced the sigmoid functions, e.g.

$$S(x) = \frac{e^{ax}}{e^{ax}+1} \quad \text{with} \quad a > 0 \,, \tag{3}$$

which had previously been widely used. Both activation functions, the ReLU (2) and the sigmoid function (3) for a=2 are shown in Fig. 4.

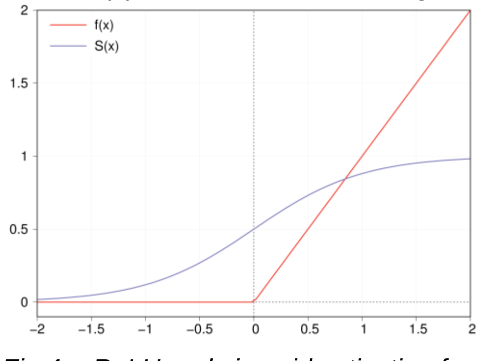

Fig.4: ReLU and sigmoid activation function

The weights were initialized by random tensors following a uniform distribution, while the biases start at a value of 0. For the training, in other words the search for the optimal values for the weights and biases, the optimizer Adam was used. Adam stands for adaptive moment estimation and was first published in 2014 by Kingma & Ba [6]. It allows to use past gradients for the calculation of current gradients by adding fractions of the past gradients to the current ones, which follows the concept of momentum. This optimizer is very popular and widely accepted in practical use for the training of artificial neural networks.
The input data is divided into 90% training and 10% test data. The training itself is only carried out with the use of the training data. The test data is used to assess the generality of the "trained" artificial neural network. Under certain circumstances, the so-called overfitting can occur if the training data is learned too precisely. In this case the error for the prediction of input with slightly different values increases. This can be observed if at a certain point in the training the error evolving from predictions for the test data increases again (although the error for the training data continues to decrease). Fig. 5 shows the evolution of the mean squared error of the trainings and test data during training. It is possible to average the gradients of a batch of several data points to reduce the effort which leads, in some cases, to a more stable convergence. This has turned out to be not very helpful in this particular case. About 100 iterations (epochs) were necessary to converge to an optimum.

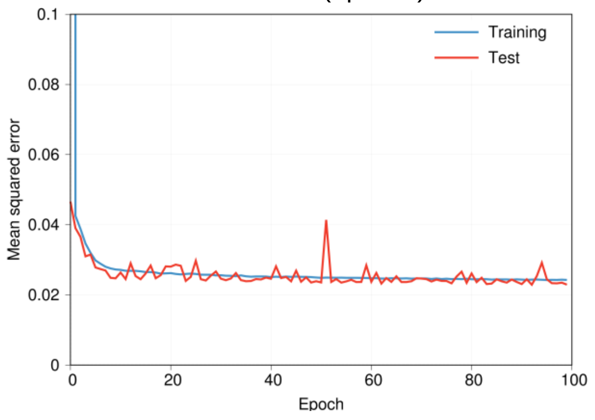

Fig.5: Evolution of the mean squared error

5 Validation

In order to validate the trained network, 81 additional data sets were generated which are not part of the training or test data. For this purpose, all four parameters (σ_0^{inp}, B^{inp}, C^{inp}, N^{inp}) were combined applying values of 0.22, 0.55 and 0.88 in a full-factorial approach. The force-displacement curves were calculated for each of these variants and for each curve the four parameters were predicted with the help of the trained neural network. Since it is conceivable that different yield curves lead to the same force-displacement curves, the predicted parameters (σ_0^{pred}, B^{pred}, C^{pred}, N^{pred}) must not be compared with the input parameters. Instead, the force-displacement curve must be calculated from the predicted parameters again, which can now actually be compared. The entire validation procedure is shown in Fig. 6.

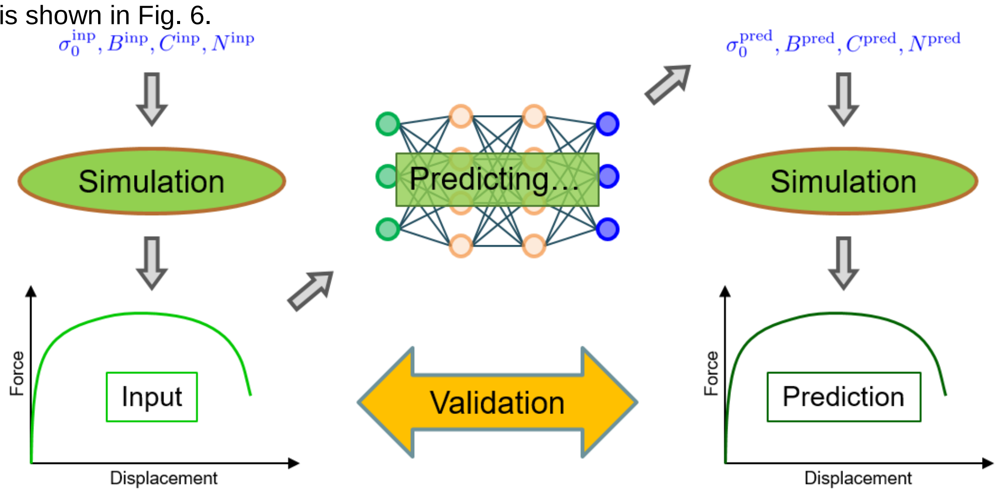
$\sigma_0^{\mathrm{inp}}, B^{\mathrm{inp}}, C^{\mathrm{inp}}, N^{\mathrm{inp}}$
Simulation
Force
Input
Displacement
Predicting...
Validation
$\sigma_0^{\mathrm{pred}}, B^{\mathrm{pred}}, C^{\mathrm{pred}}, N^{\mathrm{pred}}$
Simulation
Force
Prediction
Displacement

Fig.6: Validation procedure

The force-displacement curves of six representative examples out of the 81 validation samples are shown in Fig. 7. The associated input parameters are listed in Table 1.

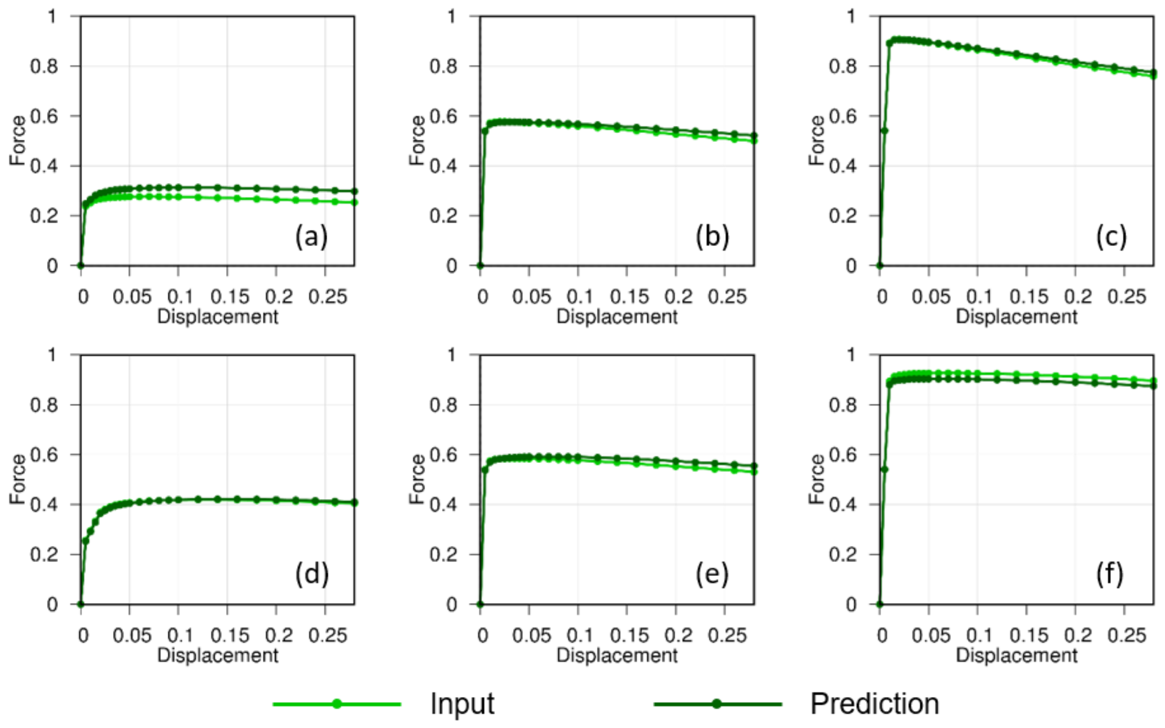

Fig.7: Force-displacement curves for exemplary input and corresponding predicted parameter sets

	σ_0^{inp}	B^{inp}	C^{inp}	N^{inp}
(a)	0.22	0.55	0.22	0.22
(b)	0.55	0.22	0.55	0.55
(c)	0.88	0.22	0.22	0.88
(d)	0.22	0.55	0.88	0.22
(e)	0.55	0.88	0.55	0.55
(f)	0.88	0.88	0.88	0.88

Table 1: Exemplary input parameter sets

The accuracy in the agreement of the curves varies, but is remarkably good in the overall picture. In order to better assess the quality of the predictions, the results were further evaluated. For this purpose, the mean squared error (MSE) was calculated for each curve:

$$\mathrm{MSE} = \frac{1}{25}\sum_{i=1}^{25}(\mathrm{Y}_i^{\mathrm{inp}} - \mathrm{Y}_i^{\mathrm{pred}})^2 \ . \qquad (2)$$

Therein Y_i^{inp} and Y_i^{pred} are the individual force values of the force-displacement curves from the input and output parameter set, respectively. The distribution of the MSEs is shown in Fig. 8, with the examples selected above marked accordingly.

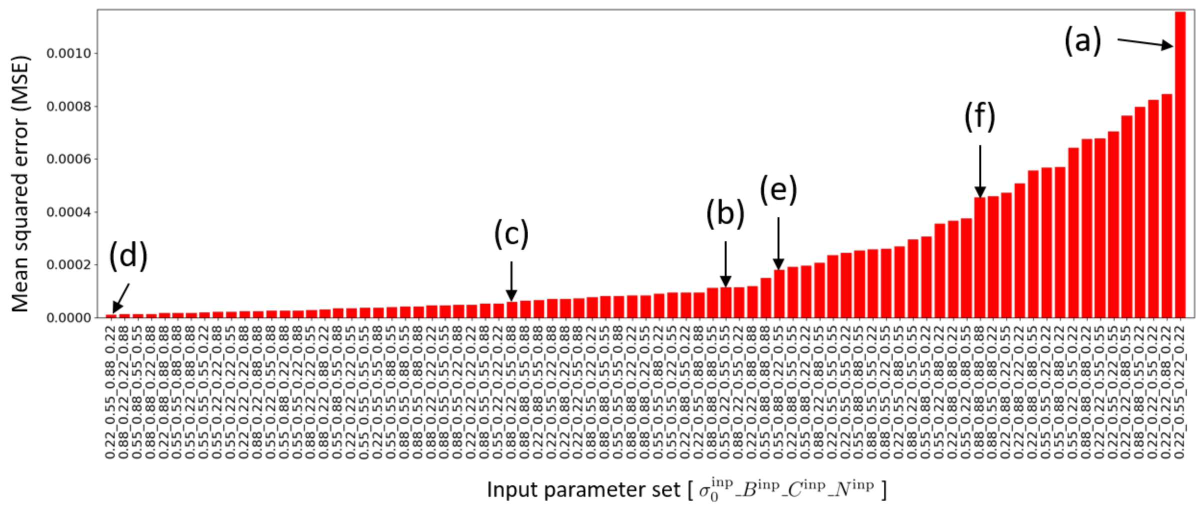

Fig.8: Distribution of the MSE of the validation sets

6 Summary and Outlook

Obviously, it is possible to reproduce the relationships between force-displacement curves and a subset of the material parameters by means of an appropriately trained artificial neural network.
Although no actual test results were used in this contribution, at least the required parameters for the simulatively generated curves could be predicted very well. Ultimately, all possibilities of force-displacement curves that can be generated by the variation of the parameters in the considered ranges are learned during the training. If one can justifiably assume from a material that it can be represented by this material model by an appropriate choice of the parameters, then good parameter combinations should also be found for real experiments although the artificial neural network was trained only on the basis of computer simulations. This fact makes the method seem much more interesting. Once a material model has been trained, the best possible parameter combination can be predicted immediately by means of real experiments. An important criterion here is, of course, compliance with the boundary conditions. In real experiments, these must correspond to the trained boundary conditions of the simulation runs.
In the following only a few of the many possibilities for further development of the method are given:

- **Consider more parameters**
  Other/more parameters of the material model could be considered, such as the modulus of elasticity, yield locus definition, strain rate dependence, damage, etc.
- **Examination of hyperparameters**
  Hyperparameters are parameters that are set before the training starts, such as the number of hidden layers and their number of neurons, layer properties (initial values, activation function), and so on. By varying the hyperparameters, their influence on the predictive quality of the trained network can be investigated.
- **More detailed tensile tests**
  By a more realistic sample geometry and its corresponding discretization, more realistic force-displacement or stress-strain curves can be generated. This would make it possible to predict parameters for real material based on actual experimental data.

In the example presented, no deep neural network was necessary; rather a quite small hidden layer was sufficient to adequately map the existing relationships. However, this could change significantly with increasing number or deviating selection of parameters and most importantly with an increasing model complexity.
It will be interesting to see how this approach will evolve and how it will take the next steps towards the vision of a neural network based material model.

7 Acknowledgement

The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the framework of AIAx (project number 01IS18048 F).

8 Literature

[1] Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; Kudlur, M.; Levenberg, J.; Monga, R.; Moore, S.; Murray, D. G.; Steiner, B.; Tucker, P.; Vasudevan, V.; Warden, P.; Wicke, M.; Yu, Y.; Zheng, X.: "TensorFlow: A system for large-scale machine learning", 12th USENIX Symposium on Operating Systems Design and Implementation (2016): 265-283.

[2] Chollet, F.: Deep learning with python, Manning Publications (2018).

[3] Goodfellow, I; Bengio, Y.; Courville, A: "Deep learning", MIT press (2016): p. 196.

[4] Rumelhart, D. E.; Hinton, G. E.; Williams, R. J.: "Learning representations by back-propagating errors", Cognitive modeling 5 (3) (1988): 1.

[5] Hahnloser, R. H. R.; Sarpeshkar, R.; Mahowald, M. A.; Douglas, R. J.; Seung, H. S.: "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit", Nature 405 (2000): 947-951.

[6] Kingma, D. P., Ba, J. L.: "Adam: A method for stochastic optimization", arXiv preprint (2014): arXiv:1412.6980.