Creating Machine-Learning-Friendly Training Data from Crash Simulation Data

Sarah Zenne[1], Joachim Sprave[1], Markus Stoll[2]

[1]Mercedes-Benz AG
[2]Renumics GmbH

Abstract

A method is presented for generating training data from FEM meshes based on element properties. Learning at the element level is often indicated when larger structures, especially parts, are too inhomogeneous in size or geometry. At the element level, quality metrics and other information are gathered from the neighborhoods of elements, just as convolutional neural networks for image processing gather information from the neighborhoods of pixels. But in general, neighborhoods of elements are not as well structured as pixels in images. Instead, they form irregular graphs which cannot be processed by standard Neural Network (NN) architectures directly. There are two approaches to make irregular graphs Machine-Learning-friendly: specialized NNs such as Graph NNs, and preprocessing of data where the features of training examples are precalculated from a neighborhood graph. While Graph NNs seem to be a natural choice, data preprocessing has still its advantages in overall simplicity and explainability. In this paper, the advantages of precalculated features are highlighted. Together with the method a Python library that allows CAE engineers to extract Machine-Learning-friendly data from LS-DYNA keyfiles as well as simulation results with a minimum amount of coding is presented. This library streamlines the process of preparing mesh data for Machine Learning and facilitates the implementation of various Machine Learning models for mesh quality evaluation. The data generation and subsequent usage by Machine Learning methods is shown by the example of learning mesh quality assessment from labelled training data. Results for this data that already have been published are used as a baseline for a comparison of model accuracy.

1 Machine Learning Representations for CAE

The tasks in CAE pre- and postprocessing are widespread and heterogenous in numerous ways, e.g. the kind of data involved, the level of detail, or the amount of data available. Mostly, researchers focus on simulation and postprocessing, solving tasks such as the prediction of simulation results [7] or grouping simulations from a design of experiment (DOE) [2]. Applications of Machine Learning in CAE preprocessing are rare, but existent, e.g. for the evaluation of mesh quality [14]. Compared to established application of Machine Learning such as image classification or timeseries prediction, CAE data has some challenges, namely a low number of of training examples, inhomogenous data, and non-Euclidean data.

1. **Low Number of Training Examples:**
   CAE data is mostly generated in a product development process. Depending on the development cycle, there are usually only a couple of variants of a single product per year. Even when it is possible to use data from different products, there are not more than a couple of hundreds of evaluated training examples available. In the world of Machine Learning, this still is a very low number. In some cases it is possible to generate training examples automatically by parametric variation and automated evaluation. But even this costs time, as the examples must be simulated.

2. **Inhomogeneous Data:**
   For Machine Learning, training examples must be from the same domain to be comparable. Images can be scaled a common size, timeseries be resampled to have a common length and so on. CAE handles products, assemblies, and parts. Variants of products, even from the same manufacturer and the same generation, differ in size, geometry, material, and function.

©2023 Copyright by DYNAmore GmbH, an Ansys Company

Graph Terminology	FEM Terminology
node/vertex	element
edge	node
hyperedge	set of elements connected to the same node

Table 1: Comparison of graph and FEM terminology.

For example, a passenger car can be a available as sedan, convertible, and station wagon. So perhaps parts should be considered. A lot of parts are common to these variants, even to other series and predecessor products. But again there is a dilemma here: The same kind of part can have a completely different size and material in two variants. Also, different parts from a single product are barely comparable. A crashbox has almost nothing in common with a trunk hood.

3. **Non-Euclidean Data:**
   The next level of detail to make data comparable are CAE meshes. At a first glance, one is tempted to treat them as images. Since there are faces or volumes in a grid, so they have a lot in common with images. But unlike images CAE meshes are not Euclidean. Meaning they are not structured in a Euclidean way (or: They are not in a Euclidean space). Instead, CAE meshes form irregular graphs, which is the point of origin for this work.

The presented framework is not exclusively designed for CAE use cases, but is applicable to all use cases with graph data underlying.

2 Data Representation for Meshed CAE Parts

Learning from images by means of Machine Learning is well researched [9] and widely used with great success. Thus, a possible course of action to apply Machine Learning to CAE data is image-based. In order to preserve some of the 3-dimensional information, multi-view approaches are common, e.g. [13] where images of objects are captured images from different perspectives. However, this means breaking down information of 3-dimensional parts to 2-dimensional images of the parts. Assume to pass the CAE data in form of 2-dimensional images to a Machine Learning algorithm, the Machine Learning algorithm needs to recover the third dimension from the multi-sview representation. This means that information is re-created that was already available beforehand. The obvious way to retain the third dimension during training is to use a voxel-based representation [8]. A 3-dimensional image is created by using small cubes (voxels) instead of small squares (pixels). This is appropriate for solid modelling where the 3-dimensional space is filled to a good amount. For the design of structures such as car bodies, this results in an almost empty space where most of the voxels are zero. This is especially true for surface-based designs which are modelled as shell meshes. Since these meshes have no thickness the space is, from a mathematical point of view, completely empty. The solution proposed here is to work directly on the finite element mesh that is applied to the parts anyways. The approach makes use of the fact that the mesh can be interpreted as a graph. As terminologies in Finite Elements and graphs use similar terms in different ways, a clarification can be found in Table 1. From now on graph terminology is used.

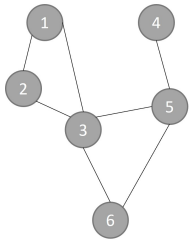
2.1 Graphs and Hypergraphs

In graph theory, a graph is defined as a pair $G = (V, E)$ with V being a set of vertices and E a set of edges where one edge links two vertices. Whereas a hypergraph is defined as a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V}$ being a set of vertices and $\mathcal{E}$ a set of edges where one edge can link any number of vertices [1]. The adjacency matrix is a square matrix representing a graph indicating which vertices are linked by an edge. See Figure 1 for an example.
Each graph is trivially a hypergraph and each hypergraph can be represented as a bipartite graph.

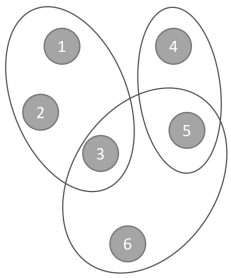
2.2 Previous Work

GNNs are trending in the AI community [16]. For various reasons (see Chapter 3), a sample-and-aggregate (SAGE) approach to extract features from graph neighborhoods is used. There are

©2023 Copyright by DYNAmore GmbH, an Ansys Company



	V1	V2	V3	V4	V5	V6
V1	0	1	1	0	0	0
V2	1	0	1	0	0	0
V3	1	1	0	0	1	1
V4	0	0	0	0	1	0
V5	0	0	1	1	0	1
V6	0	0	1	0	1	0



	V1	V2	V3	V4	V5	V6
V1	0	1	1	0	0	0
V2	1	0	1	0	0	0
V3	1	1	0	0	1	1
V4	0	0	0	0	1	0
V5	0	0	1	1	0	1
V6	0	0	1	0	1	0

Figure 1: Left: a graph and its adjacency matrix. Right: a hypergraph and its adjacency matrix. Note that the adjacency matrices of the graph and the hypergraph are identical.

several different approaches to be found in previous work. GraRep [4] represents graph vertices as low dimensional vectors considering global structural information of the graph. Node2vec [5] on the other hand represents graph nodes in a vector. DeepWalk [12] is based on random walks through the graph to learn a latent representation. LINE [15] preserves local and global information on the graph by proposing an edge-sampling algorithm. Node embeddings for each node of a graph are generated in GraphSAGE [6], where a function is learned that generates embeddings in the local neighborhood of a node by sampling and aggregating features.
In the CAE environment, the users of our approach are highly trained engineers with vast amounts of knowledge in their field, which is to be exploited by giving them the possibility to contribute to the method with their skills. This is why a technique where experts can customize sampling and aggregating and thus enrich data with their knowledge is looked into.

3 SAGE vs. GNN

In the following SAGE approaches (in particular SAGE as proposed here) and GNNs are compared under several different aspects. Note that the section on preprocessing refers to data prepocessing, not CAE preprocessing. For legibility SAGE is abbreviated by S and GNNs by G.

3.1 Preprocessing

S: As SAGE is a preprocessing method, it goes without saying that this part of the Machine Learning process is crucial. By sampling and aggregating graph data, Machine Learning (GNNs put aside) is enabled to work with the given data and domain. To go even further, SAGE enables the user to contribute their knowledge to the model by designing the SAGE algorithms accordingly.

G: GNNs on the other hand don't need much prepocessing. These methods are specifically designed to handle graph data. Thus customizing in preprocessing is not possible.

3.2 Selection of Methods

S: With data made ready for Machine Learning by SAGE, all sorts of Machine Learning methods can be applied. Thus a very well researched foundation of methods is at disposal.

G: Altough GNN methods are very popular today and become more sophisticated steadily, the topic is still rather new and error-prone, specifically for domains like CAE which are not very popular in the area of AI Research.

3.3 Data Filtering

S: The major argument to use a SAGE ansatz is that features are fully customizable. Features can be crafted and chosen on the basis of domain specific knowledge. In some applications this helps pushing the Machine Learning model into the right direction.

G: When applying GNNs to a problem all information on the graph given to the model is used in training without any domain specific filtering of the data. This can obtain good results, but might need to reconstruct information that is obviously computable for the domain expert

©2023 Copyright by DYNAmore GmbH, an Ansys Company

and thus leads to either bigger models trying to reconstruct the information or models poorly performing.

4 Best of Both Worlds: mesh2vec

The mesh2vec approach, which is open-source, is described in detail below. The source code is available on GitHub at github.com/Renumics/mesh2vec, and the documentation can be found at renumics.github.io/mesh2vec. To install mesh2vec, you can use the pip package manager to install it.

4.1 Automated SAGE on Hypergraphs

mesh2vec is a framework automating SAGE on hypergraphs. Thus for any hypergraph, mesh2vec can be used to make the graph data ready for Machine Learning without using GNNs. Roughly the procedure can be divided in two steps.

1. **Compute neighborhoods in the hypergraph:**
   For each graph node compute all neighbors for the required distance(s).

   Here two graph nodes are neighbors of distance 1 if and only if they are connected by an edge. Two graph nodes are neighbors of distance n if and only if the shortest path connecting the nodes is of length n.

   Let V be an arbitrary node in the hypergraph and $N_1^{V,m}, \ldots, N_n^{V,m}$, $n \in \mathbb{N}$ neighbors in distance $m \in \mathbb{N}$. Then the set of the neighbors $\mathcal{N}^{V,m}$ is the neighborhood of V in distance m.

   $$\mathcal{N}^{V,m} = N_1^{V,m}, \ldots, N_n^{V,m} \tag{1}$$

2. **Aggregate features in neighborhood:**
   Features are aggregated in the neighborhoods computed in step 1.

   $$\mathrm{feat}_{\mathcal{N}^{V,m}} = \mathrm{aggr}\left(\mathrm{feat}_{N_1^{V,m}}, \ldots, \mathrm{feat}_{N_n^{V,m}}\right) \tag{2}$$

   with aggr being an aggregation function and $\mathrm{feat}_{N_1^{V,m}}, \ldots, \mathrm{feat}_{N_n^{V,m}}$ features given on the nodes in the neighborhood $\mathcal{N}^{V,m}$. Potential aggregation functions are mean, max, min, std, median, etc.

This procedure can be applied to

- neighborhoods in various and/or combined distances,
- all kinds of features and information given per node in the hypergraph and
- any desired aggregation function.

The result of applying mesh2vec to graph data is a set of feature vectors of fixed length to which standard Machine Learning models are directly applicable.

Unlike other SAGE-approaches like GraphSAGE [6] the features here are entirely definable and tunable by the user. Which might at first seem like a disadvantage but for some application this actually is an advantage as expert and domain knowledge can be incorporated in a very straightforward way, whereas in other approaches expert and domain knowledge needs to be retrieved by the Machine Learning method itself.

Note that by looking into the dual hypergraph [1] features can analogously be aggregated for information on graph edges.

©2023 Copyright by DYNAmore GmbH, an Ansys Company

4.2 Application of Machine Learning Models to Prepared Data

The feature vector of fixed length resulting of applying steps 1 and 2 can now be used as input data to standard (or even more sophisticated) Machine Learning models. A wide range of Machine Learning models have been designed for application on feature vectors. These models are very well researched and inspected. Thus they provide a sound foundation for further investigation of the data.

4.3 CAE-specific Procedure

The mesh2vec_cae API allows the generation of a hypergraph based on the geometry of a FEM mesh. Each hypergraph node corresponds to an element in the mesh. Features like the number of border nodes, the aspect ratio, the warpage, the area, and the face normals can be extracted from the mesh and added to the hypergraph. Simulation results like stresses and strains values can also be extracted from LS-DYNA d3plot files and added to the hypergraph.
The aggregation of features is performed over adjacent hyper nodes (mesh elements) in the specified distance, meaning features are gathered from elements that are at exactly at the specified distance from the focal element. Aggregation is conducted using methods like calculating the mean or max value. The framework also has the capability to use the angle difference between neighboring elements as a feature.
The result of this procedure is a feature table, with each row corresponding to an element and each column relating to an aggregated feature for a specific distance.

5 Example Mesh Quality

The quality of the FEM meshes is important for the reliability of the simulation results. It can be assessed by evaluating the quality of the individual elements. Traditional metrics like aspect ratio, warpage, and skewness can be used, but manual time-consuming review ad rework by experienced engineers is also required to ensure the quality of the mesh.

5.1 Data Preparation

The data used for this example consists of FEM meshes visualized in Figure 2. From each FEM mesh, various features are extracted to characterize the elements. These features include the number of borders (num_border), aspect ratio (aspect), warpage, whether the element is triangular (is_tria), area, and face normals.
For the data preparation process, the mesh2vec_cae module from the mesh2vec library is utilized. First, the mesh is used to create the hypergraph. Then, the features are extracted from the mesh and added to the hypergraph. The feat_list variable specifies the list of features to be extracted.

```
from mesh2vec.mesh2vec_cae import Mesh2VecCae
feat_list = ["num_border", "aspect", "warpage", "is_tria", "area", "skew"]

tmp_json = f"{os.path.basename(keyfile)}_tmp.json"
hg = Mesh2VecCae.from_ansa_shell(
    dist,
    Path(keyfile),
    json_mesh_file=Path(tmp_json),
)
hg.add_features_from_ansa(
    feat_list,
    Path(keyfile),
    json_mesh_file=Path(tmp_json)
)
hg.add_features_from_ansa(
    ["normal"],
    Path(keyfile),
    json_mesh_file=Path(tmp_json)
)
...
```

Listing 1: Extracting and Adding Features to Hypergraph

©2023 Copyright by DYNAmore GmbH, an Ansys Company

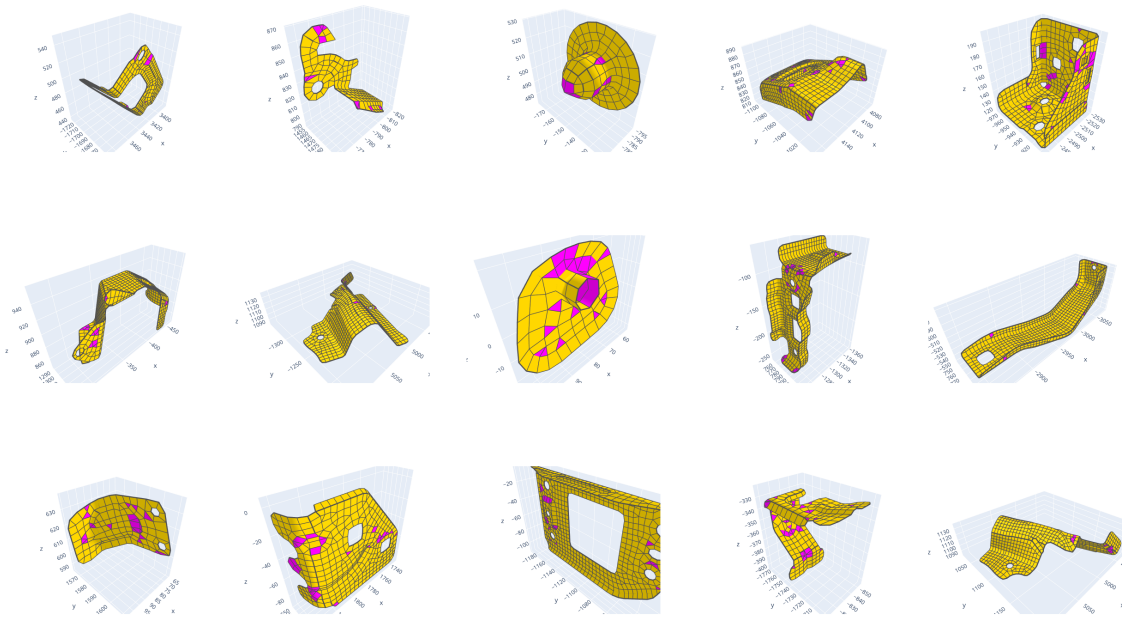


Figure 2: Exemplary meshes of the dataset. In total it consists of 317 element-wise labeled meshes, each labeled either as a 'good' or 'bad' element by an experienced engineer based on predefined quality criteria.

5.2 Labeling

In the dataset used here, the part ID (PID) was (mis-)used to mark-up bad elements. Instead of grouping the elements of a part, the PID of each element contains a specific number, which signifies whether the quality of the element is deemed acceptable or unacceptable.

5.3 Feature Aggregation

The features extracted from the FEM meshes are aggregated over the neighboring elements. The aggregation is performed considering a distance range of 0 to 2, meaning features are aggregated for elements at distances 0, 1, and 2 from the target element. The aggregation is done using various aggregation functions, such as taking the mean value. Additionally, the angle difference between neighboring elements is calculated. This process results in a feature table where the features (num_border, aspect, warpage, is_tria, area, skew) are available for the neighbors at distances 0, 1, and 2.

```
    for dist in range(dist):
    for feat in feat_list:
        hg.aggregate(feat, dist, aggr=np.mean)
    hg.aggregate_angle_diff(0, aggr=np.mean)
```

Listing 2: Feature Aggregation

The feature table in Figure 3 shows the features extracted from the FEM meshes. It is suitable for training and evaluation of all kinds of Machine Learning models.

5.4 Training

To train and evaluate the Machine Learning models, the dataset is divided into three subsets: training, validation, and test. The division is based on the original parts from which the FEM meshes were derived. The ExtraTreesClassifier from the sklearn.ensemble module, which is a random forest-based classifier [3], is used for training. The feature vectors are used as input, and the corresponding PID labels serve as the targets. The ExtraTreesClassifier is fitted on the training set to learn the relationships between the features and the mesh quality labels.

©2023 Copyright by DYNAmore GmbH, an Ansys Company

	vtx_id	num_border-mean-0	aspect-mean-0	warpage-mean-0	is_tria-mean-0	area-mean-0	skew-mean-0	normal-mean-0	num_border-mean-1	aspect-mean-1	...
0	17143	0.0	0.000000	0.000000	1.0	7.664105	24.416404	0.000000e+00	0.000000	1.183927	...
1	17150	0.0	1.286823	0.034990	0.0	14.406429	15.116270	0.000000e+00	0.000000	0.823584	...
2	17151	0.0	1.151017	0.480923	0.0	12.052604	18.165319	0.000000e+00	0.000000	0.929593	...
3	17152	0.0	0.000000	0.000000	1.0	8.108811	26.282369	0.000000e+00	0.000000	1.387146	...
4	17157	0.0	1.926077	4.886513	0.0	20.513919	2.436117	0.000000e+00	0.000000	0.898328	...
...	...	...	...	...	...	...	...	...	...	...	...
16356	9424	0.0	1.308234	4.544224	0.0	21.475245	9.624285	0.000000e+00	0.000000	1.386487	...
16357	9430	0.0	1.393880	0.637030	0.0	14.754092	8.439347	0.000000e+00	0.000000	1.379185	...
16358	9432	0.0	1.398841	0.764482	0.0	14.742327	8.459510	2.107342e-08	0.000000	1.461883	...
16359	9433	0.0	1.441166	2.547138	0.0	24.027509	8.362294	1.490116e-08	0.000000	1.457562	...
16360	9464	2.0	1.219348	0.000000	0.0	27.486608	21.243961	0.000000e+00	0.666667	1.124493	...

16361 rows × 21 columns

Figure 3: The feature table consists of 15 columns representing the features and has a total of 1,383,441 rows, with each row corresponding to an element in the FEM mesh. Example: 'aspect-mean_1' is the mean aspect ratio of the elements direct neighbors is calculated with Equation 2.

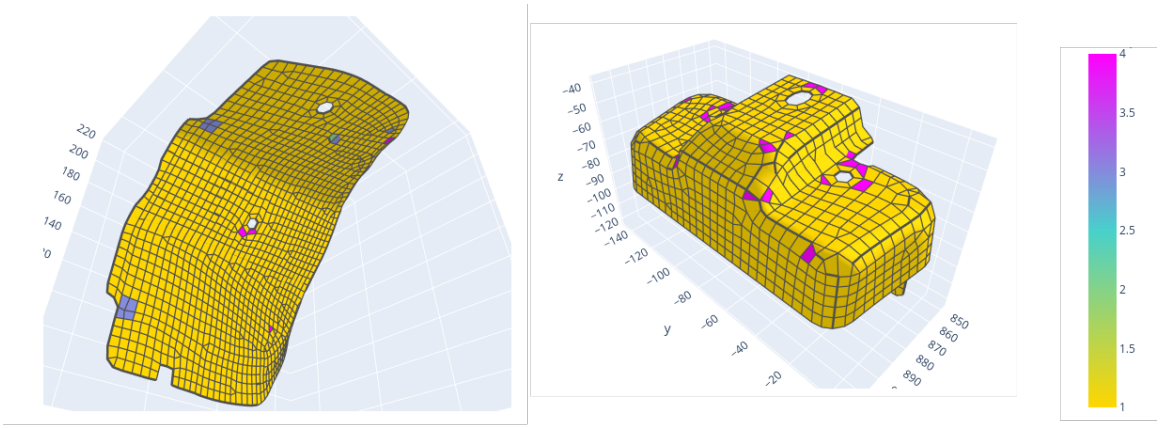


Figure 4: Exemplary predictions in the test set. 0=True Negatives (TN); 1=True Positives (TP); 2=False Negatives (FN); 3=False Positives (FP). Left: TN:2055 TP:4 FN:36 FP:4 Right:TN:784 TP:10 FN:27 FP:28

5.5 Evaluation

The trained model is evaluated on models of the test set to assess its performance. The evaluation metrics include true negatives (TN), true positives (TP), false negatives (FN), and false positives (FP). Further evaluation, such as cross-validation, can also be performed to validate the model's generalization performance. Figure 4 shows two exemplary results.

6 Example d3plot

mesh2vec also supports the generation of element-based feature vectors for LS-DYNA simulation results. The hypergraph can be generated from a d3plot or d3part file. Base features are read from the results using the LASSO Python library [10]. Any scenario where a user wants to extract and aggregate information from element neighborhoods can be implemented with a few lines of code.

Example: Strain Prediction

The existence of effective plastic strain in the neighborhood of an element can be an indicator for a future impact on this element. Figure 5 shows a state of a 3-point bending hatprofile simulation, with a fringe plot for plastic strain. A feature vector built on the elements' state only cannot distinguish between elements which are not likely to see any plastic strain in the next few time steps, and those which have a high probability of developing plastic strain very soon. In order to generate this simple feature vector in mesh2vec, the base feature plstrain must be aggregated for a distance of zero. The aggregation function should return the original value, thus it can be anything that

©2023 Copyright by DYNAmore GmbH, an Ansys Company

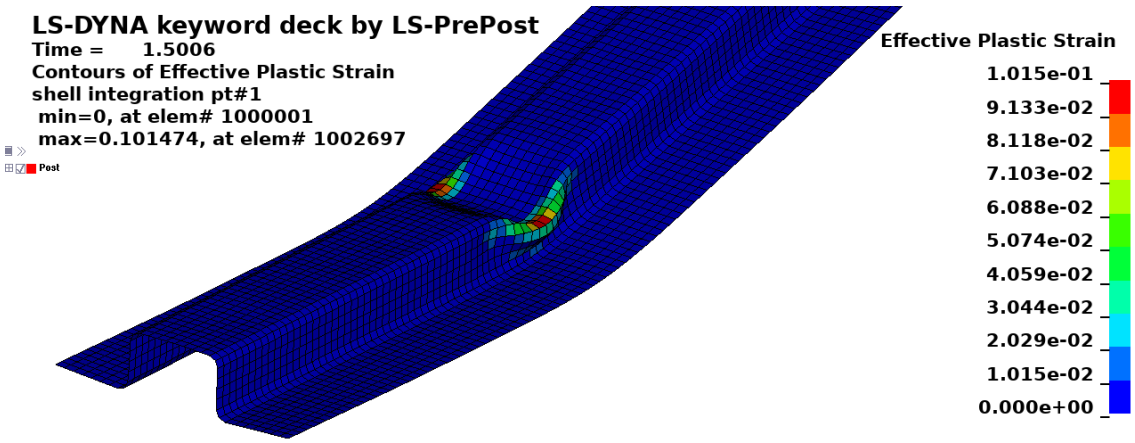


Figure 5: Result from a 3-point bending of a hatprofile in LS-PREPOST. Fringe component is effective plastic strain of integration point number 1.

is equivalent to identity for a single value, e.g. $\min$, $\max$, or mean. Figure 6 shows the same information as in Figure 5 on the undeformed mesh in the mesh2vec's built-in viewer. This viewer is a convenience function of mesh2vec and does not replace a tool like LS-PREPOST by any means.

```
from lasso.dyna import ArrayType
from mesh2vec.mesh2vec_cae import Mesh2VecCae
hg = Mesh2VecCae.from_d3plot_shell(
    5,
    Path("HAT.d3part"),
    partid=1,
)
```

Listing 3: A hypergraph neighborhood is read from an LS-DYNA result. Here, neighborhoods up to a distance of 5 are extracted from the file `HAT.d3part`. The hypergraph is restricted the part with PID=1.

```
plstrain_feature = hg.add_feature_from_d3plot(
    "element_shell_effective_plastic_strain",
    Path("HAT.d3part"),
    timestep=16, shell_layer=0,
)
thickness_feature = hg.add_feature_from_d3plot(
    "element_shell_thickness",
    Path("HAT.d3part"),
    timestep=15, shell_layer=0,
)
d0plstrain = hg.aggregate(plstrain_feature, 0, np.max)
d3plstrain = hg.aggregate(plstrain_feature, 3, np.mean)
d1thickness = hg.aggregate(thickness_feature, 1, np.max)
df = hg.to_dataframe()
```

Listing 4: Adding effective plastic strain and element thickness as features

In Listing 4, effective plastic strain and element thickness are added as features, using time step 17 and integration point number 1 (mesh2vec starts counting from 0) for plastic strain, and time step 16 for the element thickness. The features are only available for the generation of Machine Learning after aggregation which is done in lines 19 – 21. Plastic strain is aggregated for a distance of 0, providing the original value, and averaged for a distance of 3. Additionally, the maximum thickness in a distance of 1 is added as an aggregated feature. Finally, using the `to_dataframe` method, the features are made available as Pandas DataFrame which can be easily used as training data for Machine Learning methods. Figure 7 shows the resulting feature vectors for a small subset of elements. The resulting hypergraph in Figure 7 contains a fixed-sized vector of real values for each element. This is a format most Machine Learning methods can process directly. For example, one of the simplest forms of unsupervised learning, dimensional reduction by means of principal component analysis (PCA), can be easily applied using the scikit-learn [11] package, see Listing 5 for the code and Figure 9 for the visualization. While this example may not be useful due to its simplicity, one could create a more meaningful result by adding more features and aggregations.

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
kmeans = KMeans(n_clusters=3).fit(df.values[:,1:])
```

©2023 Copyright by DYNAmore GmbH, an Ansys Company

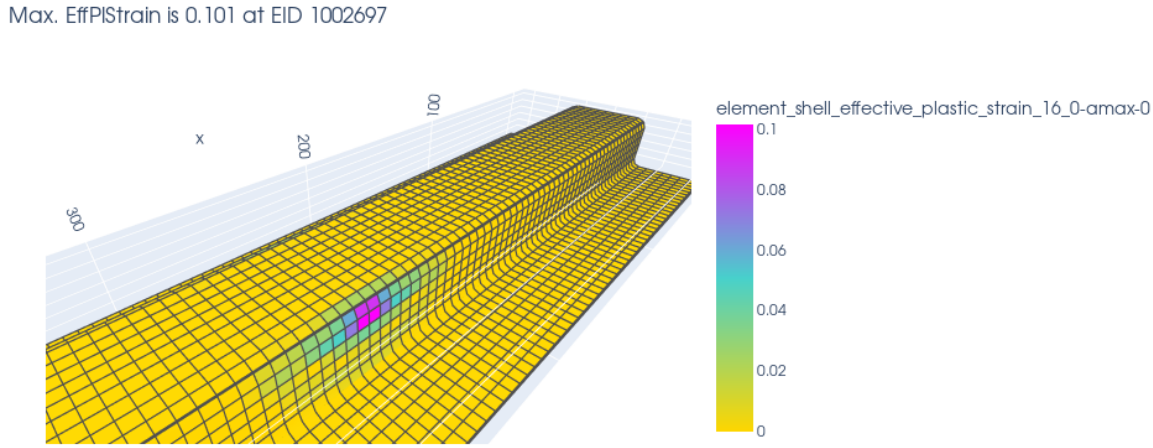


Figure 6: Visualization of the hatprofile from mesh2vec_cae. Here, the effective plastic strain of the hatprofile above has been taken from the d3part of the simulation. It has been maximized for a distance of 0 which means that the unaggregated values are shown and match the fringe plot of LS-PREPOST above.

	vtx_id	element_shell_effective_plastic_strain_16_0-amax-0	element_shell_effective_plastic_strain_16_0-mean-3	element_shell_thickness_15-amax-1
1290	1001294	0.010867	0.017195	1.023168
1291	1001295	0.023877	0.018721	1.034358
1292	1001296	0.023887	0.018725	1.034396
1293	1001297	0.028781	0.017861	1.054509
1294	1001298	0.028843	0.017855	1.054500

Figure 7: Some feature vectors generated from the hatprofile 3-point bending example.

Figure 8: Here, the values have been aggregated by average over all neighboring elements in distance 3.

©2023 Copyright by DYNAmore GmbH, an Ansys Company

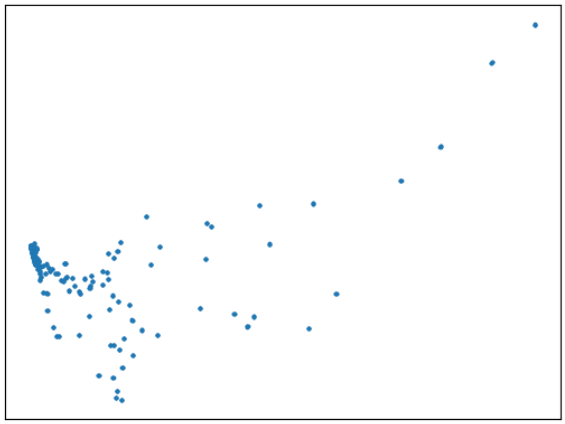


Figure 9: 2-dimensional PCA from the feature table generated above. Each dot stands for an element. Most elements gather in the lower left corner which presumably is the boring region, i.e. elements that did not see much strain in ths simulation. The upper left corner probably contains the elements in the region where the impactor has hit.

```
pca = PCA(n_components=2)
elem_pca = pca.fit(df.values[:,1:]).transform(df.values[:,1:])
plt.xticks([])
plt.yticks([])
plt.scatter(elem_pca[:,0], elem_pca[:,1], s=3)
```

Listing 5: Code to generate and plot a 2-dimensional PCA from the element-feature-table shown in Figure 7

7 Summary & Outlook

In this publication a tool to generate Machine-Learning-friendly data from graph data with little code and effort is presented. The underlying method is analyzed substantially and its advantages and disadvantages are disclosed. The potential of the library can only be hinted at by the two examples presented. The authors assume that other applications come to the experienced CAE engineers mind.
The proposed method and the associated tool are not limited to CAE applications. The authors plan to evaluate method and tool within other graph-related domains such as social networks.

Acknowledgements

The contribution of Joachim Sprave and Markus Stoll was supported by the Federal Ministry for Economic Affairs and Climate Action of Germany for project AIMM (Artificial Intelligence for Materials Modelling; identification 19I20024).

©2023 Copyright by DYNAmore GmbH, an Ansys Company

References

[1] C. Berge. *Graphs and Hypergraphs*. Graphs and Hypergraphs. North-Holland Publishing Company, 1973.

[2] Bastian Bohn, Jochen Garcke, Rodrigo Iza-Teran, Alexander Paprotny, Benjamin Peherstorfer, Ulf Schepsmeier, and Clemens-August Thole. Analysis of car crash simulation data with nonlinear machine learning methods. *Procedia Computer Science*, 18:621–630, 2013. 2013 International Conference on Computational Science.

[3] L Breiman. Random forests. *Machine Learning*, 45:5–32, 10 2001.

[4] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 891900, New York, NY, USA, 2015. Association for Computing Machinery.

[5] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 855864, New York, NY, USA, 2016. Association for Computing Machinery.

[6] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[7] Christopher P. Kohar, Lars Greve, Tom K. Eller, Daniel S. Connolly, and Kaan Inal. A machine learning framework for accelerating the design process using cae simulations: An application to finite element analysis in structural crashworthiness. *Computer Methods in Applied Mechanics and Engineering*, 385:114008, 2021.

[8] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning, 2019.

[9] Oscar Lorente, Ian Riera, and Aditya Rana. Image classification with classic and deep learning techniques, 2021.

[10] open-lasso python. https://github.com/open-lasso-python/lasso-python. https://github.com/open-lasso-python/lasso-python, 2023.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[13] Marco Seeland and Patrick Mäder. Multi-view classification with convolutional neural networks. *Plos one*, 16(1):e0245230, 2021.

[14] Joachim Sprave and Christian Drescher. Evaluating the quality of finite element meshes with machine learning, 2021.

[15] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

[16] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, jan 2021.

©2023 Copyright by DYNAmore GmbH, an Ansys Company