Estimation of spot weld design parameters using deep learning

Akhil Pillai[1], Uwe Reuter[1], Marko Thiele[2]

TU Dresden[1]
SCALE GmbH[2]

1 Introduction

1.1 Motivation

In automotive production, each automobile has approximately 7,000 to 12,000 spot welds along with other kinds of connections. The position of the spot weld with respect to the flange and the distance between the spot welds as well as various other parameters usually vary for each part combination (spot weld design). If these properties are known, they can be used for automatic generation of spot welds during the design phase of the product development which is otherwise a cumbersome manual process. The spot weld design to be determined by the engineer depends on many factors (input parameters) such as loads and forces that might be applied to the structure, material combination, geometry of the parts, connection technology and its process parameters. Some of these parameters such as material combination and geometry of the parts are predefined by the designer or are results of the circumstances such as loads and forces applied at the connection. The remaining parameters such as connection technology, process parameters, spot weld distances and flange distance have to be chosen by the engineer. On the basis of existing designs and with help of machine learning techniques it may be possible to predict the spot weld design parameters like spot weld distance and flange distance. Within this work existing spot-weld designs are extracted from a vast amount of FEM simulation input data available in the Simulation Data Management (SDM) system LoCo of SCALE GmbH and applied as the basis for training and benchmarking new methods for estimating spot weld parameters.

1.2 Objective

In this work, the distances between spot welds are estimated for spot weld design using machine learning approaches. For a machine learning approach, the first step is to collect relevant data required to predict the desired outcome. Within this work, the desired outcome is the spot weld design parameter of distance between the spot welds. In order to predict the desired outcome, we make use of simulation data used for crash analysis. From the simulation data for crash analysis, we extract the geometry of the parts and also the spot weld designs. From the data of already developed car models, we can build a model which should be able to provide a good initial estimate for distance between spot welds. With this model the design engineer has a good initial estimate of distance between spot welds for different part combinations and hence the number of design iterations required to reach the optimum values decreases.

2 Mathematical basics

This chapter presents the mathematical concepts used in this work in a brief manner to provide basic understanding to the reader. The subsections cover the basics of Artificial Neural Networks (ANN's) and deep learning architectures.

2.1 Artificial neural networks

Basic explanations of Artificial Neural Networks (ANNs), on which this subsection is based, can be found in [1,2,3]. The formation of ANNs is an attempt to mathematically model the performance and intuitive capabilities of the human brain. The functionality of the human brain is essentially based on the interaction between the brain's highly cross-linked nerve cells called natural neurons. The communication within a natural neural network takes place via signals. A neuron serves for receiving, processing and passing on incoming signals. The signals received from neighboring nerve cells are summed in the neuron and if this simulation exceeds a particular threshold value, then further signal is activated and transmitted to the adjoining neurons.

This basic structure is imitated by ANNs. The ANNs are then used to realize complex mappings of input variables on output variables. ANNs mainly consist of cross-linked computational nodes or artificial neurons and communication takes place via numerical values. An artificial neuron receives numerical values from neighboring neurons (input neurons), which are combined to form a weighted sum. This determined sum is then compared with a threshold value (bias) and used as the argument for so called activation functions. The activation function yields a value (output signal) which is an input signal for further connected artificial neurons. An important type of artificial neural network is the multilayer perceptron. The artificial neurons are arranged in layers. Starting from an input layer, numerical values are transferred or propagated to an output layer via one or more hidden layers. The output layer provides the result for the corresponding input data. The input and output data are usually real numbers.

2.2 Convolutional neural networks

Convolutional networks [4], also known as convolutional neural networks, or CNN's, are a specialized kind of artificial neural network for processing data that have a known grid like topology. Examples of data on which CNN's can be employed include time series data, which can be thought of as 1-D taking samples at regular time intervals, and image data which can be thought of as a 2-d grid of pixels. The name "convolutional neural networks" indicates that the network employs a mathematical operation called convolution. Convolutional neural networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of its layer. Research into convolutional network architecture proceeds so rapidly that we have a new best architecture for a given benchmark announced every few months. Hence it is impractical to describe the best architecture. However, all the architectures have mainly been composed of the building blocks described in this section. A typical layer of a convolutional network consists of three stages as shown in Figure (1). In the first step, a layer performs several convolutions in parallel to produce a set of linear activation's. In the second step, individual linear activation is run through a nonlinear activation function, mostly the rectified linear activation function. This step is sometimes called the detector layer. In the third step, we use a pooling function to modify the output of the layer further.

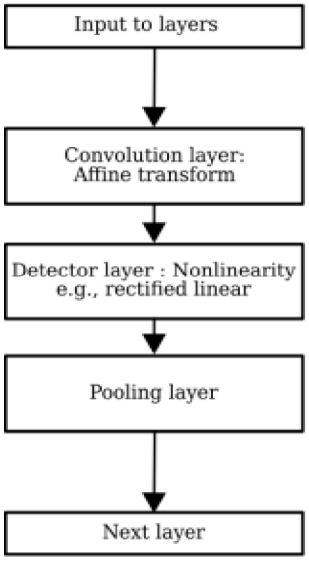

Fig.1: Components of typical convolutional neural network layer

2.3 Barycentric coordinates

In geometry, the barycentric coordinate system is a coordinate system in which the location of a point of a simplex (a triangle, tetrahedron, etc.) is specified as the center of mass, or barycenter, of usually unequal masses placed at its vertices. The system was introduced in 1827 by August Ferdinand Möbius [5]. Consider a set of points $P_0, P_1, \mathrm{K}, P_n$ and consider the set of all affine combinations taken from these points, i.e., all points P can be written as

$$\alpha_0 P_0 + \alpha_1 P_1 + \mathrm{L} + \alpha_n P_n \tag{1}$$

for some

$$\alpha_0 + \alpha_1 + \mathrm{L} \ + \alpha_n = 1 \tag{2}$$

Then this set of points forms an affine space, and the coordinates $(\alpha_0, \alpha_1, \mathrm{K}, \alpha_n)$ are called the barycentric coordinates of the points of the space [6]. These coordinates system is frequently useful and extensively used in working with triangles. This barycentric parameterization is exactly the parameterization that is usually used in our case for generating 3D geometric data.

In context of this work, we will make use of barycentric coordinates to generate points in a triangle. Consider three points P_1, P_2, P_3 in the plane and $\alpha_1, \alpha_2, \alpha_3$ are scalars such that $\alpha_1 + \alpha_2 + \alpha_3 = 1$, then the point P defined by

$$P = \alpha_1 P_1 + \alpha_2 P_2 + \alpha_3 P_3 \tag{3}$$

is a point on the plane of the triangle formed by P_1, P_2, P_3. This point is within the triangle $\Delta P_1 P_2 P_3$ if

$$0 \le \alpha_1, \alpha_2, \alpha_3 \le 1 \tag{4}$$

If any of the α's is less than zero or greater than one, then the point P is outside the triangle. If any of the α's is zero then P is on one of the lines joining the vertices of the triangle.

In this work, we would utilize the concept of Barycentric coordinates to generate points inside elements of a part. In order to achieve this we would need to sample combinations of $\alpha_1, \alpha_2, \alpha_3$ so that we can generate points inside elements that are uniformly distributed. In order to do, we used the concept of Latin Hypercube sampling.

3 Machine learning approaches for classification of geometric data

3.1 Theoretical aspects of 3D geometric data

The raw 3D data that are captured by different methods come in forms that vary in both, the structure and properties. This section presents comprehensive information on different representations of 3D data by categorizing them into two main families:

Euclidean-structured data

Certain 3D representations have an underlying Euclidean-structure where the properties of the grid-structured are preserved such as having a global parameterization and a common system of coordinates. The main 3D representations that fall under this category are:

- descriptors
- 3D data projections
- RGB-D data
- volumetric data and
- multiview data

Non-Euclidean data

Another type of 3D data representations is the non-Euclidean data. This type of data representations suffers from the absence of global parameterization and the non-existence of a common system of coordinates or vector space structures [7], which makes processing of such data representations a challenging task. Considerable efforts are directed towards learning from such data and applying machine learning techniques on it. The main type of non-Euclidean data is point clouds, 3D meshes and graphs. Usually processing such data types happens on a global scale to learn the whole 3D object's feature which is convenient for complex tasks such as recognition and correspondence.

3.2 Deep learning on 3D data

3D data has multiple popular representations as described briefly in Section (3.1), leading to various approaches for learning. In this section we will briefly discuss some of the significant machine learning approaches applied to them.

- *Volumetric* CNNs: [8,9,10] are the pioneers applying 3D convolutional neural networks on voxelized shapes. Volumetric representation is constrained by its resolution due to data sparsity and computation cost of 3D convolution. FPNN [11] and Vote3D [12] has proposed methods to deal with the sparsity problem; however, it's challenging for them to process very large point clouds.
- *Multi-view* CNNs: [10, 13] have tried to render 3D shapes into 2D images and then apply 2D convolutional networks to classify them. This approach has achieved dominating performance on shape classification and retrieval tasks due to the fact that image CNN's are well engineered for classification tasks [14]. However the question of how many views are required to model the 3D shapes is still open.
- *Spectral* CNNs: Some of the latest approaches [15, 16] use spectral CNNs on meshes. However, these methods are currently constrained on manifold meshes such as organics objects and it's not obvious how to extend them to non-isometric shapes.
- *Feature-based* DNNs: [17, 18] extracts traditional shape features from 3D data and converts it into a vector. Then they use a fully connected network to classify the shape. This approach is constrained by the representation power of the features extracted.

3.3 PointNet

In this work, we explore deep learning architectures capable of reasoning about 3D geometric data of point clouds. Point clouds are simple and unified structures that avoid the combinatorial irregularities and complexities of meshes and are thus easier to learn from. The network named PointNet [19], provides a unified architecture for applications ranging from object classification, part segmentation, to scene parsing. Though simple, PointNet is highly efficient and effective. Empirically, it shows strong performance on par or even better than state of the art. In this section we will discuss about the approach of using this deep learning architecture for geometry classification.

PointNet is a unified architecture that directly takes point clouds as input and outputs either class labels for the entire input or per point segment/part labels for each point of the input. The basic architecture of the network is surprisingly simple as in the initial stages each point is processed identically and independently. A point cloud is represented as a set of 3D points $\{P_i \mid i = 1,\mathrm{K}\ ,n\}$ where each point P_i is a vector of its (x, y, z) coordinate plus extra feature channels such as color, normal etc. For simplicity and clarity, we only use the (x, y, z) coordinates as our input. The network architecture, visualized in Figure (3), provides k scores for all the k candidate classes. The network has three key modules: the max pooling layer as a symmetric function to aggregate information from all the points, a local and global information combination structure, and two joint alignment networks that align both input points and point features.

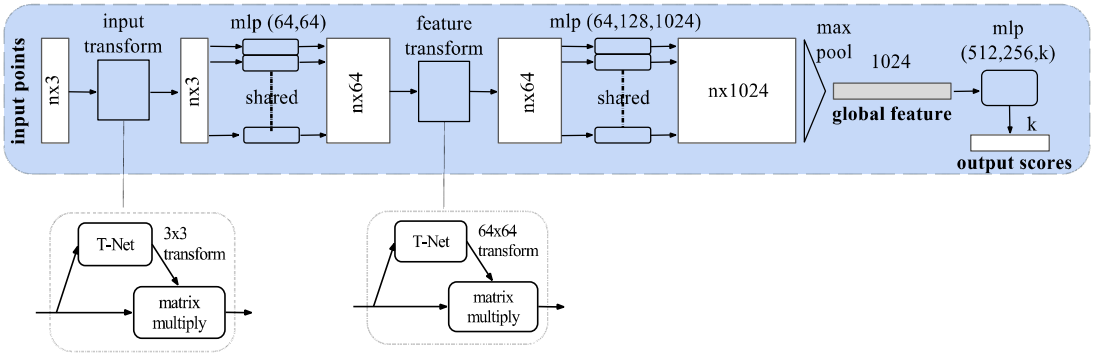

Fig.2: ***PointNet Architecture****. The classification takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes.*

The PointNet Architecture is implemented in TensorFlow, which is a open source machine learning frame work. In this work we used Python 2.7 to generate training data and Python 3.5 to train the PointNet architecture [20]. In order to speed up the training cycle we used a GPU (Graphics Processor Unit). TensorFlow allows us to use GPU's to train machine learning algorithms with the aid of CUDA and cuDNN. CUDA is parallel computing platform and programming model invented by NVIDIA. It enables dramatic increase in computing performance by harnessing the power of GPU [21]. The NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU- accelerated library of primitives for

deep neural networks [22]. cuDNN provides highly tuned implementations for standard routines such as forward and back ward convolution, pooling, normalization, and activation layers. In this work we utilized CUDA 9.0 and cuDNN 7.0 for training purpose.

The training data for PointNet was stored in HDF5 binary data format. Python package "h5py" was used to achieve this. HDF5 is a high-performance data management and storage suite. It supports n-dimensional datasets and each element in the data set can itself be complex object. It comes with a set of integrated performance features that allow for access time and storage space optimizations. Also there is no limit on the number or size of data objects in the collection, giving great flexibility for big data. In this work we would be working with storing huge amount of geometric data as point clouds and hence HDF5 was chosen.

3.4 Generation of 3D geometric data from FEM data

In the previous section, a deep learning architecture is explained briefly which consumes point clouds and can be used to classify 3D geometry. In this section ideas and algorithms used to generate point clouds of 3D geometry for parts of car. Firstly we will have a look at how part geometry is expressed in FEM data. Then we will discuss methods to extract part geometry for individual parts and then using this part geometry data we will generate point clouds. The approaches presented were developed using libraries developed by SCALE GmbH.

3.4.1 Extraction of geometry from FEM data

In this subsection, we will discuss the ideas used to extract geometry data from FEM data. Nodes are the base for expressing geometric data in FEM data. Elements are formed with nodes and using elements we express part geometry. So when we want to extract geometric data, it indicates that we are extracting the coordinates of nodes for elements in a part. Now one can argue that we can just extract data of nodes present in a part and use it as point cloud of a part. This would be a simple process but we know that for feeding this point cloud into PointNet we would require the number of points in point cloud for each part to be the same. In Body in white (BIW) of a car not all parts are of same size and hence the number of nodes used to express the geometries of different parts will be obviously different. Hence it becomes necessary to extract data associated with elements that make up the parts and then use this data to generate points on surface of the parts according to our need. In this work we will only encounter with planar shell elements and thus the further discussions on elements will consider only the properties of shell elements.

SCALE GmbH has developed a python library called **femparser** with which one could parse the include decks of FEM data and extract geometric data. The output of **femparser** is a python object which can be used to extract the geometric data of individual parts using combination of part numbers, elements ID's and node ID's. The extraction algorithm is presented in Algorithm (1). The output is stored in JSON format files for the ease of reading the outputs and for further processing steps (i.e. generation of point clouds)

Algorithm 1 Part geometry extraction

Input: FEM include file of car
Apply femparser → mesh (python object)
procedure 1. Extract all part id's
    for part in mesh **do**
        extract part id's
    end for
end procedure
procedure 2. Extract geometry of each part
    for id in parts id's **do**
        Extract element id from mesh
        for element id **do**
            extract nodal coordinates from mesh
        end for
    end for
end procedure
Output: Part JSON files