Fortgeschrittenenpraktikum — Prof. Dr. T. de Wolff
Data Science — Dr. C. Brauer

Industry Project
Neural Networks for Leakage Detection in Vacuum Bagging

With regard to the application at hand, your goal is to learn a parameterized function f_θ that predicts leakage coordinates $y \in [-1,1] \times [-1,1]$ from normalized flow rates

$$x \in \{x \in \mathbb{R}^4 \mid \sum_j x_j = 1,\ x \geq 0\} \ .$$

For that purpose, you have training data $(x^i, y^i)_{i=1}^m$ available. Unfortunately, the fourth sensor MFC4 had a slight malfunction at the time when the training data were produced. As a consequence, the values x_4 are on average higher than their counterparts x_1, x_2 and x_3. Be aware that the same effect is not present in the validation and test data.

You are further equipped with prior knowledge in the form of so-called equivariance conditions that can be incorporated into the machine learning pipeline. To see what that means, let us start with a fixed vector of flow rates $x = (\mu_1, \ldots, \mu_4)$ and associated leakage coordinates y as displayed in Figure 1 (top row, left), and suppose that the position of the leakage is rotated clockwise by 90° (top row, second from left). In this situation, if you neglect potential measurement noise, it is natural to claim that rotated coordinates y_{90} should be associated with accordingly shifted flow rates $x_{90} = (\mu_4, \mu_1, \mu_2, \mu_3)$. By analogous arguments, you can also perform a flip along the vertical axis (bottom row, left) and claim that the resulting coordinates y_{flipped} come along with flow rates $x_{\text{flipped}} = (\mu_2, \mu_1, \mu_4, \mu_3)$. Subsequent rotations and flips result in a total of seven additional virtual configurations per sample.

It is possible to derive a dedicated network architecture that incorporates the above-mentioned equivariance conditions explicitly, *i.e.*, for each fixed example with $f_\theta(x) = y$ your network will satisfy all of the following conditions

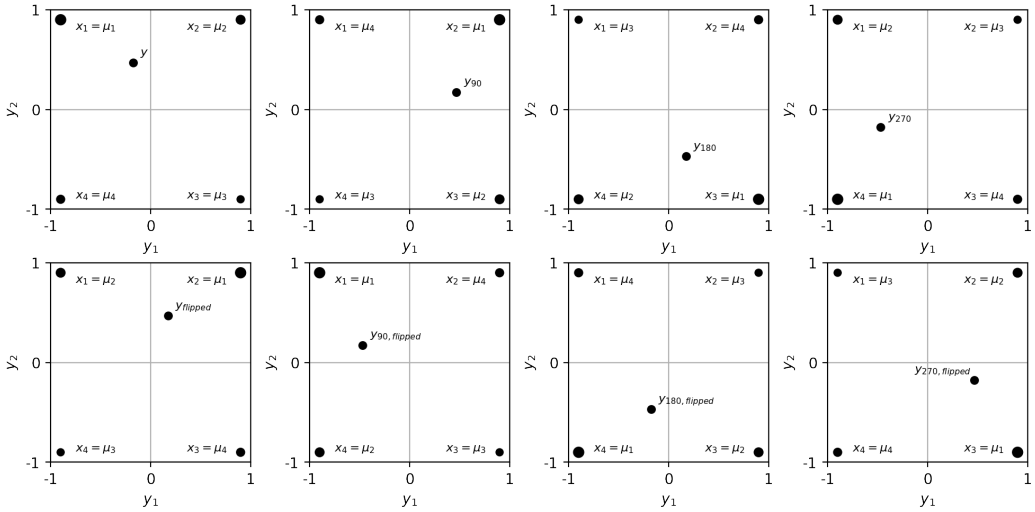

Figure 1: Motivation of equivariance conditions. Markersizes (in corners) indicate flow rates.

by construction:

$$
\begin{aligned}
f_\theta(x_{90}) &= y_{90} \\
f_\theta(x_{180}) &= y_{180} \\
f_\theta(x_{270}) &= y_{270} \\
f_\theta(x_{flipped}) &= y_{flipped} \\
f_\theta(x_{90,flipped}) &= y_{90,flipped} \\
f_\theta(x_{180,flipped}) &= y_{180,flipped} \\
f_\theta(x_{270,flipped}) &= y_{270,flipped}
\end{aligned}
$$

This is not true if you use standard architectures. However, it is possible to create also virtual training examples according to Figure 1 and use these to generate an augmented dataset. By using the augmented dataset for training, you can provide additional information also to standard networks.

In this project, your overall tasks are to implement, train and compare equivariant and standard network architectures for the task of leakage detection. To that end, you are going to use datasets of different sizes, also in combination with data augmentation. This will enable you to assess the additional value of prior knowledge that is incorporated directly through a dedicated network architecture and / or through virtual training examples.

For both training set sizes $m \in \{100, 1000\}$, proceed as follows:

1. Load the datasets
   `leakage_dataset_train_[m].csv`
   `leakage_dataset_validation_1000.csv`
   and store the data as NumPy-Arrays
   `X_train`, `Y_train`
   `X_validation`, `Y_validation`.

2. Train a standard fully connected neural network. Hyperparameters like number of epochs, batch size, optimizer, learning rate, depth and width of the network, activation and loss functions – all up to you. You can use a framework like KerasTuner to optimize hyperparameters, or simple `for`-loops if you prefer that. Anyway, you should use the validation data in order to evaluate and compare candidate hyperparameter configurations. Save the model as `model_standard_[m].h5`.

3. Augment the dataset. Each training example (x, y) can be complemented by seven additional virtual examples as illustrated in Figure 1. Clockwise rotations and flips on input data x are obtained as described above. A clockwise rotation on output data y is obtained through $y_{90} = (y_2, -y_1)$ and a flip along the vertical axis on the same data is represented by $y_{flipped} = (-y_1, y_2)$. All other operations (rotation angles and associated flips) can be computed by subsequent application of these two operations.

4. Repeat 2. on the augmented dataset. Save this model as
   `model_standard_[m]_augmented.h5`.

5. Train an equivariant neural network on the original dataset. To that end, you need to implement a custom model using the Keras Subclassing API. All layers except the output layer must have a weight matrix that looks as follows

$$W^\ell := \begin{bmatrix} a & b & c & b \\ b & a & b & c \\ c & b & a & b \\ b & c & b & a \end{bmatrix}$$

where a, b, c are the parameters that shall be learned. Moreover, equivariant layers have no bias vectors, *i.e.*, you have $b^\ell = 0$. The output layer of an equivariant network has a weight matrix

$$W^L \coloneqq \begin{bmatrix} d & -d & -d & d \\ -d & -d & d & d \end{bmatrix} \tag{1}$$

with only one parameter d, and no bias unit as well. Apart from these specifically structured weight matrices and the absence of bias vectors, equivariant networks look all like standard networks. However, they do not come as pre-implemented Keras layers, so you have to use the subclassing API. Once you have this, you can do anything else just as in case of the standard network. Save this model as `model_equivariant_[m].h5`.

6. Train an equivariant network on the augmented dataset. Save this model as `model_equivariant_[m]_augmented.h5`.

If you repeat these steps on both datasets, you finally have 8 winner models that can be compared in terms of their performance, so that you should be able to answer the follwing questions:

- Which model worked best depending on the size of the dataset?
- In which cases did data augmentation enhance the performace?

Finally, compare your models in terms of a visual inspection. This can, *e.g.*, be accomplished by visualizing model predictions on the following sets:

- $\{x \in \mathbb{R}^4 \mid x_1 + x_3 = 0.5,\ x_2 = x_4 = 0.25\}$
- $\{x \in \mathbb{R}^4 \mid x_2 + x_4 = 0.5,\ x_1 = x_3 = 0.25\}$

If your implementation of equivariant networks is correct, all associated predictions should lie on a diagonal of the coordinate plane.

At any time before the submission deadline, you can send models via email (or via your Stud.IP folder and email notification) to `ch.brauer@tu-bs.de`. Models will then be evaluated on the test data in terms of the mean absolute error and added to the leaderboard.