Noname manuscript No.
(will be inserted by the editor)

Backtransformation: A new representation of data processing chains with a scalar decision function

blinded author names

Received: 07 September 2014, revised: 11 July 2015, second revision: 28 September 2015

Abstract Data processing often transforms a complex signal using a set of different preprocessing algorithms to a single value as the outcome of a final decision function. Still, it is challenging to understand and visualize the interplay between the algorithms performing this transformation. Especially when dimensionality reduction is used, the original data structure (e.g., spatio-temporal information) is hidden from subsequent algorithms. To tackle this problem, we introduce the backtransformation concept suggesting to look at the combination of algorithms as *one* transformation which maps the original input signal to a single value. Therefore, it takes the derivative of the final decision function and transforms it back through the previous processing steps via backward iteration and the chain rule. The resulting derivative of the composed decision function in the sample of interest represents the complete decision process. Using it for visualizations might improve the understanding of the process. Often, it is possible to construct a feasible processing chain with affine mappings which simplifies the calculation for the backtransformation and the interpretation of the result a lot. In this case, the affine backtransformation provides the complete parameterization of the processing chain. This article introduces the theory, provides implementation guidelines, and presents three application examples.

Keywords affine transformations · function composition · processing chain interpretation · processing chain visualization

Mathematics Subject Classification (2000) 68T30 · 68N99 · 68W40

blinded author institutes

1 Introduction

The basis of machine learning is understanding the data (Chen et al, 2008), and generating descriptive features (Domingos, 2012). Consequently, for numerous data types and processing algorithms, visualization approaches have been developed (Rieger et al, 2004; Rivet et al, 2009; Le et al, 2012; Haufe et al, 2014; Szegedy et al, 2014) as a better abstraction to enhance the understanding of the behaviour of the applied algorithms and of the data. Here, the visualization of an algorithm is often realized in a similar way as for the input data.

To come up with a representation gets way more complicated when algorithms are combined for a more sophisticated preprocessing before applying a final decision algorithm (Verhoeye and de Wulf, 1999; Rivet et al, 2009; Krell et al, 2013a; Kirchner et al, 2013; Feess et al, 2013), i.e., for processing chains. Under these circumstances, understanding and visualization of single algorithms does only explain single steps in the processing chain that are typically not independent from each other. The order of preprocessing algorithms, e.g., influences single processing visualizations, although the value of the final decision function might be not or only weakly influenced. Hence, one is often interested in knowledge about the whole data transformation in the processing chain but a general approach for solving this problem is missing. This situation gets even worse the more complex the data and the associated processing chains become. If dimensionality reduction algorithms are used for example to reduce the complexity of the data and to get rid of the noise, the structure of the output data is usually very different from the original input after the reduction step. In such a case, it is very difficult to understand the connection between decision algorithm, preprocessing, and original data even if single parts can be visualized. Consequently, a concept for representing the complete processing chain in the domain and format of the original input data is required.

Several approaches are described in the literature to visualize the outcome and transformation of classification algorithms, but again, taking the perspective of a single processing step neglecting the processing history (i.e., the preceding algorithms).

When using classifiers with kernels, a direct visualization of the classifier becomes impossible. Baehrens et al (2010) calculate the derivative of the classification function to give information of the classifier dependent on a chosen sample. Unfortunately, this calculation of the derivative is quite complex, difficult to automatize, computationally expensive, and does not consider any processing before the classification. This makes it hard to apply and to generalize for complete data processing chains and high-dimensional data.

Blankertz et al (2011) discuss the visualization of the linear discriminant analysis (LDA) in the context of an electroencephalogram (EEG) based brain-computer interface (BCI) application with different views on the temporal, spatial and spatio-temporal domain. Here, the classifier is applied on spatial features and visualized as a spatial filter together with an interpretation in

relation to the original data and other spatial filters. For other visualizations, the classifier weights are not directly used. Furthermore, no complex signal processing chain is used, even though spatial filters are very common for the preprocessing of this type of data. The LDA was applied to the raw data and largely improved with a shrinkage criterion. As a side remark, they mention the possibility to visualize the LDA weights directly, when applied to spatio-temporal features (Blankertz et al, 2011, paragraph before section 6, p. 18).

This direct visualization of weights of a linear support vector machine (SVM) has already been suggested by LaConte et al (2005).[1] This approach is intuitive, easy to calculate, and enables a combination with the preprocessing. Furthermore, it can be generalized to other data and other classifiers (Blankertz et al, 2011).

This paper introduces our solution approach denoted as *backtransformation*. It incorporates the aforementioned approaches, but with the fundamental difference that it takes all preprocessing steps in the respective chain into account. With this approach, we are able to extract the complete transformation of the data from the chain, so that, e.g., changes in the order of algorithms or the effect of insertions/deletions of single algorithms become immediately visible. Backtransformation also considers processing chains, where the original (e.g., spatio-temporal) structure of the data is hidden. The data processing chain is identified with a (composed) function, mapping the input data to a scalar. In its core, backtransformation is the derivative of this function, calculated with the chain rule or numerically. The derivative is either calculated locally for each sample of interest (*general backtransformation*) or globally when the processing chain consists of affine transformations only (affine backtransformation). While the general backtransformation gives information on which components in the data have a large (local) influence on the decision process and which components are rather unimportant, the affine backtransformation is independent from the single sample[2].

Numerous established data processing algorithms are affine transformations and it is often possible to combine them to process the data. Hence,we also take a closer look at this type of algorithms and we show that it is possible to retrieve the information on how the data is transformed by the complete decision process, even if a dimensionality reduction algorithm or a temporal filter hide information. The *affine backtransformation* iteratively goes back from the decision algorithm through all processing steps to determine a parameterization of the composed processing function and to enable a semantic interpretation. This results in a helpful representation of the processing chain, where each component in the source domain of the data gets a weight assigned showing its impact in the decision process. In fact, summing up the products of weights and respective data parts is equivalent to applying the single algorithms on the data step-by-step.

[1] Further methods are presented but they are tailored to functional magnetic resonance imaging (fMRI) data.

[2] The respective derivatives are constant for every sample and as such not depending on it.

In Section 2, the backtransformation concept is introduced. First, we introduce the general backtransformation for differentiable processing chains. This is followed by the special variant which is obtained when working with affine transformations. To be even more specific, we discuss the backtransformation at a processing scheme for segmented time series data in Section 2.3. Here, we give examples of algorithms for the affine backtransformation, the generic backtransformation, and also mention cases where it is not applicable. In Section 2.4, we describe how the backtransformation is implemented in a generic way. This is followed by applications of the backtransformation in Section 3 Finally, a conclusion is made in Section 4.

2 Methods

The requirement to apply the proposed backtransformation as outlined in the following is that the data processing is a concatenation of differentiable transformations (e.g., affine mappings) and that the last algorithm in the chain is a (decision) function which maps the data to a *single scalar*. The final mapping to the label in case of a classification task is not relevant, here.

For each processing stage, the key steps of the backtransformation are to first choose a mathematical representation of input and output data and then to determine a parameterization of the algorithm which has to be mapped to fit to the chosen data representations. Finally, the derivatives of the resulting transformations have to be calculated and iteratively combined. In its core it is the application of the chain rule for derivatives (see Section 2.1). For the case of using only affine mappings, it is just the multiplication of the transformation matrices, as shown in Section 2.2. Details on the implementation are given in Section 2.4. For an example of a processing chain of windowed time series data with a two-dimensional representation of the data see Fig. 1 and Section 2.3.

The backward modeling begins with the parametrization of the final decision function and continues by iteratively combining it backwards with the preceding algorithms in a processing chain. With each iteration, weights are calculated, which correspond to the components of the input data of the last observed algorithm.

For the abstract formulation of the backtransformation approach, data with a one-dimensional representation before and after each processing step is used. The output of each processing step is fed into the next processing algorithm.

2.1 Backtransformation using the Derivative

This section introduces the general backtransformation. Let the input data be denoted with $x^{(0)} = x^{\text{in}} \in \mathbb{R}^{n_0}$ and let the series of processing algorithms be represented by differentiable mappings

$$F_0 : \mathbb{R}^{n_0} \to \mathbb{R}^{n_1}, \ldots, F_k : \mathbb{R}^{n_k} \to \mathbb{R} \tag{1}$$

which are applied to the data consecutively.[3] Then, the application of the processing chain to obtain the output data $x^{\text{out}} := x^{(k+1)}$ from the input data $x^{(0)}$ can be summarized to:

$$x^{\text{out}} = x^{(k+1)} = F(x^{(0)}) = (F_k \circ \ldots \circ F_0)(x^{(0)}) \; . \tag{2}$$

With this notation, the derivative can be calculated with the chain rule:

$$\frac{\partial F}{\partial y}\left(x^{(0)}\right) = \frac{\partial F_0}{\partial y^{(0)}}\left(x^{(0)}\right) \cdot \frac{\partial F_1}{\partial y^{(1)}}\left(x^{(1)}\right) \cdot \ldots \cdot \frac{\partial F_{k-1}}{\partial y^{(k-1)}}\left(x^{(k-1)}\right) \cdot \frac{\partial F_k}{\partial y^{(k)}}\left(x^{(k)}\right), \tag{3}$$

where $x^{(l)} \in \mathbb{R}^{n_l}$ is the respective input of the l-th algorithm in the processing chain with the mapping F_l and $x^{(l+1)}$ is the output. The terms $\frac{\partial F_l}{\partial y^{(l)}}$ and $\frac{\partial F}{\partial y}$ represent the total differentials of the differentiable mappings and not the partial derivatives. Equation (3) is a matrix product. It can be calculated iteratively using the backtransformation matrices B_l and the derivatives $\frac{\partial F_{l-1}}{\partial y^{(l-1)}}(x^{(l-1)})$:

$$B_k = \frac{\partial F_k}{\partial y^{(k)}}\left(x^{(k)}\right) \text{ and } B_{l-1} = \frac{\partial F_{l-1}}{\partial y^{(l-1)}}\left(x^{(l-1)}\right) \cdot B_l \text{ with } l = 1, \ldots, k \; . \tag{4}$$

Now each matrix $B_l \in \mathbb{R}^{n_l \times 1}$ has the same dimensions[4] as the respective $x^{(l)}$ and tells which change in the components of $x^{(l)}$ will increase (positive entry in B_l), decrease (negative entry), or will have no effect (zero entry) on the decision function. The higher the absolute value of an entry (multiplied with the estimated variance of the respective input), the larger is the influence of the respective data component on the decision function. Consequently, not only the backtransformation of the complete processing chain (B_0) but also the intermediate results (B_l; $l > 0$) might be used for analyzing the processing chain. B_k is the matrix used in the existing approaches, which do not consider the preprocessing (LaConte et al, 2005; Baehrens et al, 2010; Blankertz et al, 2011). Note that the B_l are dependent on the input of the processing chain and are expected to change with changing input. So the information about the influence of certain parts in the data is only a *local* information. A *global* representation is only possible when using affine transformations instead of arbitrary differentiable mappings F_l.

[3] The notation of data and its components differs from the notation in classification tasks. Here, we look at one data sample $x^{(0)}$ with its different processing stages $x^{(l)}$ and the respective changes in each component of the data $\left(x^{(l)}_{gh}\right)$. The double index notation is applied to account for different axes in the data as in time series (different sensors and time points) or images.

[4] With $n_{k+1} := 1$ it holds that $\frac{\partial F_l}{\partial y^{(l)}} \in \mathbb{R}^{n_l \times n_{l+1}}$ and the dimensions of B_l are a consequence of the recursion. Another reason for the dimensions of B_l is that B_l corresponds to the mapping of $x^{(l)}$ to the scalar output x^{out}.

2.2 Affine Backtransformation

For handling affine transformations like translations, the data vectors are augmented by adding a coordinate with value 1 to have homogenous coordinates. Every affine transformation F can be identified with a tuple (A, T), where A is a linear mapping matrix and T a translation vector and the corresponding mapping of the processing algorithm applied on data x^{in} reads as

$$x^{\text{out}} = F(x^{\text{in}}) = Ax^{\text{in}} + T = (A \;\; T) \begin{pmatrix} x^{\text{in}} \\ 1 \end{pmatrix} . \tag{5}$$

So by extending the matrix $(A \;\; T)$ to a Matrix A' with an additional row of zeros with a 1 at the translational component, the mapping on the augmented data $x'^{\text{in}} = \left(\begin{smallmatrix} x^{\text{in}} \\ 1 \end{smallmatrix}\right)$ can be written in the simple notation: $x'^{\text{out}} = A'x'^{\text{in}}$. With a processing chain with corresponding matrices $A'_0, \ldots, A'_k$ the transformation of the input data x'^{in} can be summarized to

$$x'^{\text{out}} = A'_k \cdot \ldots \cdot A'_1 \cdot A'_0 \cdot x'^{\text{in}} . \tag{6}$$

With this notation, the backtransformation concept now boils down to iteratively determine the matrices

$$B_k = A'_k \, , B_{k-1} = A'_k \cdot A'_{k-1} \, , \ldots \, , \text{ and } B_0 = A'_k \cdot A'_{k-1} \cdot \ldots \cdot A'_1 \cdot A'_0 \, . \tag{7}$$

This corresponds to a convolution of affine mappings.[5] Each $B_l \in \mathbb{R}^{(n_l+1)\times 2}$ defines the mapping of the data from the respective point in the processing chain (after l previous processing steps) to the final decision value. So *each product* B_l consists of a weighting vector $w^{(l)}$ and an offset $b^{(l)}$ and the artificial second row with zero entries and 1 in the last column. The term $w^{(l)}$ can now be used for interpretation and understanding the respective sub-processing chain or the complete chain with $w^{(0)}$ (see Section 3). The term is equivalent to the B_l from the backtransformation using the derivative (Section 2.1).

The following section renders possible (and impossible) algorithms which can be used for the affine backtransformation and how the weights from the backtransformation are determined in detail for a data processing chain applied on two-dimensional data.

2.3 Backtransformation Modeling Example

In this section, a more concrete example of applying the backtransformation principle is given for processing time series epochs of fixed length of several sensors with the same sampling frequency. We provide examples for affine transformations to show that there is a large number of available algorithms to construct a good processing chain. Additionally, cases will be highlighted

[5] Note that no matrix inversion is required even though one might expect that, because the goal is to find out what the original mapping was doing with the data which sounds like an inverse approach.

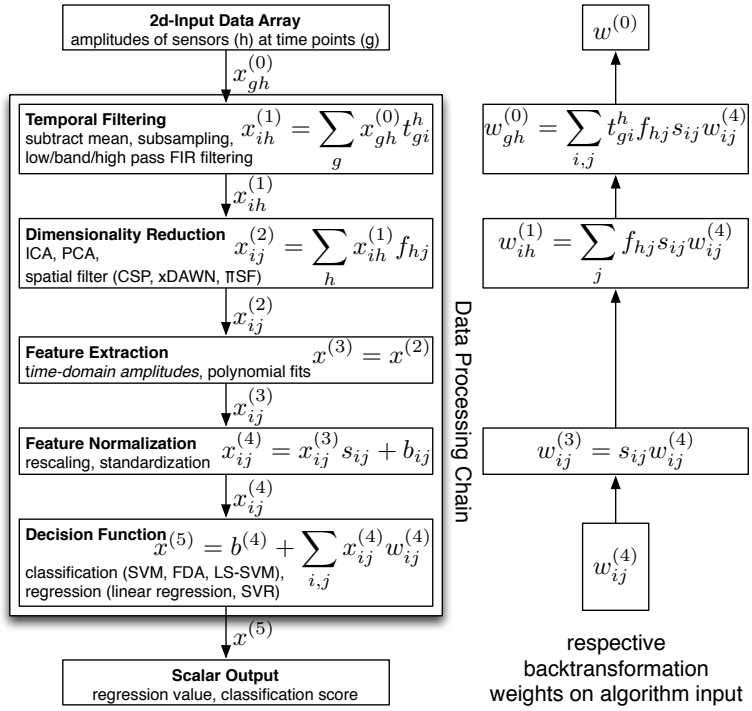

Fig. 1: *Illustrative data processing chain scheme with examples of affine algorithms and the formulas for the backtransformation in short.* Spatio-temporal data $x^{(0)}_{gh}$ are processed from top to bottom ($x^{(5)}$). Every component of the scheme is optional. Backtransformation takes the classifier parametrization $w^{(4)}$ and projects it iteratively back ($w^{(k)}$) through the processing chain and results in a representation $w^{(0)}$ corresponding to the input domain. For more details refer to Section 2.3.

which require the general backtransformation and cases where the backtransformation is not applicable.

A possible processing chain with examples of affine mappings and the respective backtransformation weights is depicted in Fig. 1. Note that all components of this chain are optional and the presented scheme can be applied to an arbitrary data processing chain even if dimensions like time and space are replaced by others or left out (see Sections 2.2 and 3.2).

An intuitive way of handling such data is to represent it as two-dimensional arrays with the time on one axis and space (e.g., sensors) on the other axis, since important preprocessing steps like temporal and spatial filters just operate on one axis. So this type of representation eases the use and the parameterization of these algorithms compared to the aforementioned mathematically equivalent one-dimensional representation. Furthermore, a two-dimensional

representation of the data helps for its visualization and interpretation. For parametrization of the two-dimensional arrays, the common double index notation is used, where the data $x^{(0)}$ is represented by its components $x^{(0)}_{gh}$ with temporal index g and spatial index h. This index scheme will be kept for all processing stages even if the data could be represented as one-dimensional feature vectors for some stages. The same indexing scheme can be applied for the parametrization of the affine data processing algorithms in the chain as will be shown in the following. The input of the i-th algorithm is denoted with $x^{(i-1)}$ and the output with $x^{(i)}$ respectively. To fit to the concept of backtransformation, first the parametrization of the decision algorithm will be introduced and then the preceding algorithms step-by-step . An overview of the processing chain, the chosen parameterizations, and the resulting weights from the backtransformation is depicted in Fig. 1.

Scalar Decision Function A linear decision function can be parameterized using a decision vector/matrix $w^{(4)}_{ij} \in \mathbb{R}^{m_i \times n_j}$ and an offset $b^{(4)} \in \mathbb{R}$. The transformation of the input $x^{(4)} \in \mathbb{R}^{m_i \times n_j}$ to the decision value $x^{(5)} \in \mathbb{R}$ is then defined as

$$x^{(5)} = b^{(4)} + \sum_{i=1}^{m_i} \sum_{j=1}^{n_j} x^{(4)}_{ij} w^{(4)}_{ij} \ , \tag{8}$$

with m_i time points and n_j sensors. Some examples for machine learning algorithms with linear decision function are SVMs (Vapnik, 1995; Steinwart and Christmann, 2008; Chang and Lin, 2011), balanced relative margin machines (Krell et al, 2014a), regularized fishers discriminant analysis classifiers (Mika et al, 2001), passive-aggressive perceptrons (Crammer et al, 2006), linear regression, support vector regression (Smola and Schölkopf, 2004), ridge regression, and one-class SVMs (Schölkopf et al, 2001; Krell and Wöhrle, 2015) and there are many more.

Depending on the application, data might be not linearly separable or a nonlinear separation provides better results. Here, a common approach is to use nonlinear kernels instead of the linear function. All common kernels are differentiable, so here the general backtransformation can be still applied instead of the affine backtransformation. As long as the decision function is differentiable, the general backtransformation can be used, too. When combining (linear or differentiable) classifiers as an ensemble it depends on the final gating function, if the resulting scalar comes from an affine/differentiable function[6]. The same holds for neural networks where different transition functions could be used. Unfortunately, for neural networks the derivative might not improve the understanding especially when it is showing unexpected local behavior as explained by Szegedy et al (2014). Nevertheless, most often these methods are differentiable. If there is no strict step function used but the function is locally Lipschitz or even locally linear the approximation of a derivative

[6] A weighted sum of classifiers preserves linearity/differentiability. A majority vote will result in a non-differentiable classifier but when the score is the sum of the voters for the selected class, the resulting function will still be locally linear/differentiable.

could be still used even though some information in the critical points might be hidden. Furthermore, in these cases it is better to use a derivative, which considers the left and the right side for each component.

If there is no scalar output or the function is locally constant it is not possible to derive information from the backtransformation. A decision tree usually produces no useful output function. If the output of a classifier is only ± 1, no information can be obtained. Another example for a locally constant function could be obtained from a linear decision function $f(x)$ by limiting its values to the interval $[-1, 1]$ with $\min\{1, \max\{-1, f(x)\}\}$. For every x with $|f(x)| > 1$ the resulting new decision function is locally constant and no interpretation of the derivative is possible.

Feature Normalization With a scaling $s \in \mathbb{R}^{m_i \times n_j}$ and transition $b \in \mathbb{R}^{m_i \times n_j}$ and the same indexes as for the linear decision function, an affine feature normalization can be written as

$$x_{ij}^{(4)} = x_{ij}^{(3)} s_{ij} + b_{ij} \text{ with } i \in \{1, \dots m_i\} \text{ and } j \in \{1, \dots n_j\} \ . \tag{9}$$

This covers most standard feature normalization algorithms like rescaling or standardization (Aksoy and Haralick, 2001). Nonlinear scalings, e.g., using absolute values as in $\min\left\{10, \left|x_{ij}^{(3)}\right|\right\}$, or sample dependent scalings, e.g., division by the Euclidean norm $s_{ij} = \frac{1}{\left\|x^{(3)}\right\|_2}$, are not affine mappings and could not be used for the affine backtransformation.

The general backtransformation could still be used for differentiable normalizations like Euclidean normalization if $x^{(3)} \neq 0$. Using min or max results in locally constant behavior which restricts the applicability of the backtransformation.

For the affine backtransformation, the formula of the feature normalization need to be inserted into the formula of the decision function:

$$x^{(5)} = b^{(4)} + \sum_{i,j} \left(x_{ij}^{(3)} s_{ij} + b_{ij} \right) w_{ij}^{(4)} = b^{(3)} + \sum_{i,j} x_{ij}^{(3)} s_{ij} w_{ij}^{(4)} \ . \tag{10}$$

Here, $b^{(3)} = b^{(4)} + \sum_{i,j} b_{ij}$ summarizes the offset. As denoted in Fig. 1, $s_{ij} w_{ij}^{(4)}$ is the weight to the input data part $x_{ij}^{(3)}$.

Feature Generation For simplicity, the data amplitudes at different sensors have been directly taken as features and nothing needs to be changed in this step $\left(x^{(3)} = x^{(2)}\right)$. Other linear features like polynomial fits would be possible, too (Straube and Feess, 2013). Nonlinear features (e.g., standard deviation, sum of squares, or sum of absolute values of each sensor) would not work for the affine backtransformation but for the general one. Symbolic features, mapped to natural numbers will be even impossible to analyze with the general backtransformation.

Dimensionality Reduction on the Spatial Component A spatial filter transforms real sensors to new pseudo sensors by linear combination of the signal of the original sensors. To use well known dimensionality reduction algorithms like principal component analysis (Lagerlund et al, 1997; Rivet et al, 2009; Abdi and Williams, 2010, PCA), and independent component analysis (Jutten and Herault, 1991; Rivet et al, 2009, ICA) for spatial filtering, the space component of the data is taken as feature component for these algorithms and the time component for the samples. Examples for typical spatial filters are common spatial patterns (Blankertz et al, 2008, CSP), xDAWN (Rivet et al, 2009; Woehrle et al, 2015), and πSF (Ghaderi and Straube, 2013).

The backtransformation with the spatial filtering is the most important part of the concept, because spatial filtering hides the spatial information needed for visualization or getting true spatial information into the classifier.

The number of virtual sensors ranges between the number of real sensors and one. The spatial filter for the j-th virtual sensor is a tuple of weights $f_{1j}, ..., f_{n_h j}$ defining the linear weighting of the n_h real sensors. The transformation for the i-th time point is written as

$$x_{ij}^{(2)} = \sum_{h=1}^{n_h} x_{ih}^{(1)} f_{hj} \ , \tag{11}$$

where the time component could be ignored, because the transformation is independent of time. The transformation formula can be substituted into formula (11):

$$x^{(5)} = b^{(3)} + \sum_{i,j} \sum_{h=1}^{n_h} x_{ih}^{(1)} f_{hj} s_{ij} w_{ij}^{(4)} \tag{12}$$

$$= b^{(3)} + \sum_{i,h} x_{ih}^{(1)} \cdot \left(\sum_{j} f_{hj} s_{ij} w_{ij}^{(4)} \right) \ . \tag{13}$$

Equation (13) shows, that the weight $\sum_j f_{hj} s_{ij} w_{ij}^{(4)}$ is assigned to the input data component $x_{ih}^{(1)}$. If there is no time component, a spatial filter is just a linear dimensionality reduction algorithm. It is also possible to combine different reduction methods or to do a dimensionality reduction after the feature generation.

For spatial filtering, linear transformations are the common choice. But for more general dimensionality reduction algorithms like the PCA, it is also possible to use kernels. Since kernels are usually differentiable, it would be still possible to apply the generic backtransformation, when such an algorithm is used in the processing chain.

Detrending, Temporal Filtering, and Decimation There are numerous discrete-time signal processing algorithms (Oppenheim and Schafer, 2009). Detrending the mean from a time series can be done in several ways. Having a time window, a direct approach would be to subtract the mean of the time window,

or to use some time before the relevant time frame to calculate a guess for the mean (baseline correction). Often, such algorithms can be seen as finite impulse response (FIR) filters, which eliminate very low frequencies. Filtering the variance is a quadratic filter (Krell et al, 2013b) and infinite impulse response (IIR) filters have a feedback part. Both filters are not applicable for the backtransformation, because they have no respective affine transformations and because they rely on the complete signal which makes it impossible to obtain a local derivative.

One can either use uniform temporal filtering, which is similar to spatial filtering with changed axis, or introduce different filters for every sensor. As parametrization, t^h_{gi} is chosen for the weight at sensor h for the source g and the resulting time point i with a number of m_g time points in the source domain:

$$x^{(1)}_{ih} = \sum_{g=1}^{m_g} x^{(0)}_{gh} t^h_{gi} \ . \tag{14}$$

Starting with the more common filter formulation as convolution (filter of length N):

$$x^{(1)}_{ih} = \sum_{l=0}^{N} a_l \cdot x^{(0)}_{(n-l)h} \overset{g:=n-l}{=} \sum_{g=n-N}^{n} a_{(n-g)} \cdot x^{(0)}_{gh} \ , \tag{15}$$

the filter coefficients a_i can be directly mapped to the t^h_{gi} and the other coefficients can be set to zero.

Reducing the sampling frequency of the data by downsampling is a combination of a low-pass filter and systematically leaving out several time points after the filtering (decimation). When using a FIR filter, the given parameterization of a temporal filter can be used here, too. For leaving out samples, the matrix t_{gi} for sensor h can be obtained from an identity matrix by only keeping the rows, where samples are taken from.

The final step is similar to the spatial filtering part:

$$x^{(5)} = b^{(3)} + \sum_{i,h} \left(\sum_{g=1}^{m_g} x^{(0)}_{gh} t^h_{gi} \right) \cdot \left(\sum_j f_{hj} s_{ij} w^{(4)}_{ij} \right) \tag{16}$$

$$= b^{(3)} + \sum_{g,h} x^{(0)}_{gh} \cdot \left(\sum_{i,j} t^h_{gi} f_{hj} s_{ij} w^{(4)}_{ij} \right) \tag{17}$$

$$= b^{(3)} + \sum_{g=1}^{m_g} \sum_{h=1}^{n_h} x^{(0)}_{gh} w^{(0)}_{gh} \ . \tag{18}$$

The input component of the original data $x^{(0)}_{gh}$ finally gets assigned the weight $w^{(0)}_{gh} = \sum_{i,j} t^h_{gi} f_{hj} s_{ij} w^{(4)}_{ij}$. Note that for some applications it is good to work on normalized and filtered data for interpreting data and the behavior of the data processing. In that case, the backtransformation is stopped before the temporal filtering and the respective weights are used.

Others The aforementioned algorithms can be combined and repeated (e.g., concatenations of FIR filters or PCA and xDAWN). Having a different feature generator, multiple filters, decimation, or skipping a filter or normalization the same calculation scheme could be used resulting in different $b^{(3)}$ and $w^{(0)}$. Nevertheless, $w^{(0)}$ has the same indexes as the original data $x^{(0)}$. After the final mapping to a scalar by the decision function, a shift of the decision criterion (e.g., using threshold adaptation as suggested by Metzen and Kirchner (2011)) is possible but has no impact on the backtransformation because it only requires $w^{(0)}$ and not the offset. If a probability fit (Platt, 1999; Lin et al, 2007; Baehrens et al, 2010) was used, this step has to be either ignored or the general approach (Section 2.1) has to be applied. Since the probability fit is a mostly sigmoid function which maps $\mathbb{R} \to [0, 1]$, it is also possible to visualize its derivative separately. For the interpretation concerning a sample, the function value is determined and the respective (positive) derivative is multiplied with the affine transformation part to get the local importance. Hence, the relations between the weights remain the same but the absolute values only change. This approach of mixing the calculations is much easier to implement and interpret.

If nonlinear preprocessing is used to normalize the data (e.g., to have variance of one), the normalized data can be used as input for the backtransformation and the respective processing chain. This might be even advantageous for the interpretation when the visualization of the original data is not helpful due to artifacts and outliers. An example for such a case is to work with normalized image data like the MNIST dataset (LeCun et al, 1998) instead of the original data, where the size of the images and the position of the digits varied a lot (see also Section 3.2 and Section 3.3).

If any of the algorithms in the observed processing chain is not an affine mapping, the affine backtransformation cannot be applied. For getting the real derivatives for the general backtransformation all algorithms need to be differentiable. But if the derivative vanishes at some points due to locally constant behavior, the backtransformation might be meaningless. On the other hand, if a generalized derivative can be determined for non-differentiable algorithms this might still work (Clarke, 1990; Rockafellar and Wets, 2009).

2.4 Generic Implementation of the Backtransformation

This section gives information on how to apply the backtransformation concept in practice especially when the aforementioned calculations are difficult or impossible to perform and a "generic" implementation is required to handle arbitrary processing chains.

The backtransformation has been implemented in a signal processing and classification environment called pySPACE (Krell et al, 2013a) and can be directly used[7]. This modular Python software gives simple access to more than

[7] `http://pyspace.github.io/pyspace/`

200 classification and preprocessing algorithms and so it provides a reasonable interface for a generic implementation. It provides data visualization tools for the different processing stages and largely supports the handling of complex processing chains.

In practice, accessing the single parameterizations for the transformation matrices A_i for the affine backtransformation might be impossible (e.g., because external libraries are used without access to the internal algorithm parameters) or too difficult (e.g., code of numerous algorithms needs to be written to extract these parameters). In this case, the backtransformation approach cannot be applied directly in the way it is described in Section 2.2. Instead, the respective products and weights for the affine backtransformation can be reconstructed with the following trick which only requires the algorithms to be affine. No access to any parameters is needed. First, the offset of the transformation product is obtained by processing a zero data sample with the complete processing chain. The processing function is denoted by F. The resulting scalar output is the offset

$$b^{(0)} = F(0). \tag{19}$$

Second, a basis $\{e_1, \ldots, e_n\}$ of the original space (e.g., the canonical basis) needs to be chosen. In the last step, the weights $w_i^{(0)}$, which directly correspond to the base elements, are determined by also processing the respective base element e_i with the processing chain and subtracting the offset $b^{(0)}$ from the scalar output:

$$w_i^{(0)} = F(e_i) - F(0). \tag{20}$$

The calculation of the derivative for the general backtransformation approach is more complicated. Deriving and implementing the derivative function for each algorithm used in a processing chain and combining the derivatives can be very difficult, especially if the goal is to implement it for a large number of relevant algorithms, e.g., as provided in the pySPACE framework. A generic approach would be to use automatic differentiation tools (Griewank and Walther, 2008). These tools generate a program which calculates the derivative directly from the program code. They can also consider the concatenation of algorithms by applying the chain rule. For most standard implementations, open source automatic differentiation tools could be applied. For existing frameworks, it is required to modify each algorithm implementation such that the existing differentiation tools know all derivatives of used elemental functions used in the code, which might be a lot of work. Furthermore, this approach would be impossible if black box algorithms were used. So for simplicity, a different approach, which is similar to the previous one for the affine case can be chosen. This is the numerical calculation of the derivative of the complete decision function via differential quotients for directional derivatives:

$$\frac{\partial F}{\partial e_i}(x_0) \approx \frac{F(x_0 + he_i) - F(x_0)}{h}\ . \tag{21}$$

Here, e_i is the i-th unit vector, and h is the step size. It is difficult to choose the optimal h for the best approximation, but for the backtransformation

a rough approximation should be sufficient. A good first guess is to choose $h = 1.5 \cdot 10^{-8} \langle x_0, e_i \rangle$ if $\langle x_0, e_i \rangle \neq 0$ and in the other case $h = 1.5 \cdot 10^{-8}$ (Press, 2007). In the backtransformation implementation in pySPACE, the value of $1.5 \cdot 10^{-8}$ can be exchanged easily by the user. It is additionally possible to use more accurate formulas for the differential quotient at the cost of additional function evaluations like

$$\frac{\partial F}{\partial e_i}(x_0) \approx \frac{F(x_0 - he_i) - 8F(x_0 - \frac{h}{2}e_i) + 8F(x_0 + \frac{h}{2}e_i) - F(x_0 - he_i)}{6h} \,. \tag{22}$$

3 Applications

Having a transformation of the decision algorithm back through different data representation spaces to the original data space might help for the understanding and interpretation of processing chains in several applications (e.g., image detection, classification of neuroscientific data, robot sensor regression) as explained in the following. First, some general remarks will be given on visualization techniques. Afterwards, the affine and the general backtransformation will be applied on handwritten digit classification (Section 3.2 and Section 3.3) because it is a relatively simple problem which can be understood without expert knowledge. Finally, a more complex example is given on EEG data classification (Section 3.4) and an outlook for further applications (Section 3.5).

3.1 Visualization in General

As suggested by LaConte et al (2005) for fMRI data, the backtransformation weights could be visualized in the same way as the respective input data is visualized. This works only if there is a possibility to visualize the data and if this visualization displays the "strength" of the values of the input data. Otherwise, additional effort has to be put into the visualization, or the weights have to be analyzed as raw numbers. For interpreting the weights, it is usually required to also have the original data visualized for comparison (as averaged data or single samples) because higher weights in the backtransformation could be rendered meaningless if the corresponding absolute data values are low or even zero. Additionally to the backtransformation visualization of *one* data processing chain, different chains (with different hyperparameters, training data, or algorithms) can be compared (Krell et al, 2014b). Differences in the weights directly correspond to the differences in the processing. Normally, weights with high absolute values correspond to important components for the processing and weights close to zero are less important and might be even omitted. This very general interpretation scheme does not work for all applications. In some cases, the weights have to be set in relation to the values of the respective data components: If data values are close to zero, high weights

might still be irrelevant, and vice versa. To avoid such problems, it is better to take normalized data, which is very often also a good choice for pure data visualization. Another variant to partially compensate for this issue is to also look at the products of weights and the respective data values.

According to Haufe et al (2014), the backtransformation model is a *backward model* of the original data and as such mixes the reduction of noise with the emphasis of the relevant data pattern. To derive the respective *forward model* they suggest to multiply the respective weighting vector with the covariance matrix of the data. From a different perspective, this approach sounds reasonable, too: If backtransformation reveals that a feature gets a very high weight by the processing chain, but this feature is zero for all except one outlier sample a modified backtransformation would reveal this effect. Furthermore, if a feature is highly correlated with other features, a sparse classifier might just use this one feature and skip the other features which might lead to the wrong assumption, that the other features are useless even though they provide the same information. On the other hand, if features are highly correlated as it holds for EEG data this approach might be also disadvantageous. The processing chain might give a very high weight to the feature, where the best distinction is possible, but the covariance transformation will blur this important information over all sensors and time points. Using such a blurred version for feature selection would be a bad choice. Another current drawback of the method by Haufe et al (2014) is that it puts some assumptions on the data which often do not hold: The expectancy values of noise, data, and signal of interest are assumed to be zero "w.l.o.g." (without loss of generality). Hence, more realistic assumptions are necessary for better applicability. The effect of the covariance correction by Haufe et al (2014) will be analyzed in Sections 3.2 and 3.3.

Note that in Fig. 1, Section 2.2, and Section 2.3 it has been shown that every iteration step in the backtransformation results in weightings $w^{(i)}$ which correspond to the data $x^{(i)}$. This data is obtained by applying the first i algorithms of the processing chain on the original input data $x^{(0)}$. So depending on the application, it is even possible to visualize data and weights of intermediate processing steps. This can be used to further improve the overall picture of what happens in the processing chain.

3.2 Processing Chain Visualization: Handwritten Digit Classification: Affine Processing Chain

For a simple application example of the backtransformation approach, the publicly available MNIST dataset is used (LeCun et al, 1998). This dataset contains numerous normalized greyscale images of all digits with a size of 28×28 pixels. They are stored as one-dimensional feature vectors (784 features). For processing, we first applied a PCA on the feature vectors and reduced the dimension of the data to four (or 64). As a second step, the resulting features were normalized to have zero mean and standard deviation of one on the training

data. Finally, a linear SVM (Chang and Lin, 2011) with a fixed regularization parameter of one is trained on the normalized PCA features. Without backtransformation, the filter weights for the 4 (or 64) principal components could be visualized in the domain of the original data and the single weights assigned by the SVM could be given, but the interplay between SVM and PCA would remain unknown, especially if all 784 principal components would be used. This information can only be given with backtransformation and is displayed in Fig. 2 for the distinction of digit pairs (from 0, 1, and 2). The generic implementation of the affine backtransformation was used, since only affine algorithms were used in the processing chain (PCA, feature standardization, linear classifier). The forward model to the backtransformation, obtained by multiplication with the covariance matrix, is also visualized in Fig. 2. Note that the original data is not normalized (zero mean), although this was an assumption on the data for the covariance transformation approach by Haufe et al (2014)[8].

Generally, it can be seen that the classifier focuses on the digit parts, where there is no overlay between the digits on average. For one class there are high positive values and for the other there are high negative weights. For the classification with 64 principal components, the covariance correction smoothes the weight usage and results in a visualization which is similar to the visualization of the backtransformation for the classification with 4 principal components. Hence, the 60 additional components are mainly used for canceling out "noise".

3.3 Processing Chain Visualization: Handwritten Digit Classification: Nonlinear Classifier

To show the effect of the generic backtransformation for a nonlinear processing chain, the evaluation of Section 3.2 is repeated with a radial basis function kernel for the SVM instead of a linear one. The hyperparameter of the kernel, γ, has been determined according to Varewyck and Martens (2011). Everything else remained unchanged. Again the generic implementation was used. Note that every sample requires its own backtransformation. So for the visualization of the backtransformation, only the first four single samples were taken.

It can be clearly seen in Fig. 3 that there is a different backtransformation for each sample. Similar to the results in Section 3.2 (Fig. 2), the backtransformation with covariance correction (when 64 principal components are taken as features) seems to be more useful in contrast to the raw visualization which also contains the noise cancellation part. This is surprising because this approach has been originally developed for linear models and not for nonlinear ones (Haufe et al, 2014). Using a correction with a "local" covariance would be more appropriate in this case but more demanding from the computation and implementation point of view. A large number of principal components seems to be a bad choice for the nonlinear kernel, because it does not seem to

[8] Nevertheless, the resulting graphics look reasonable.

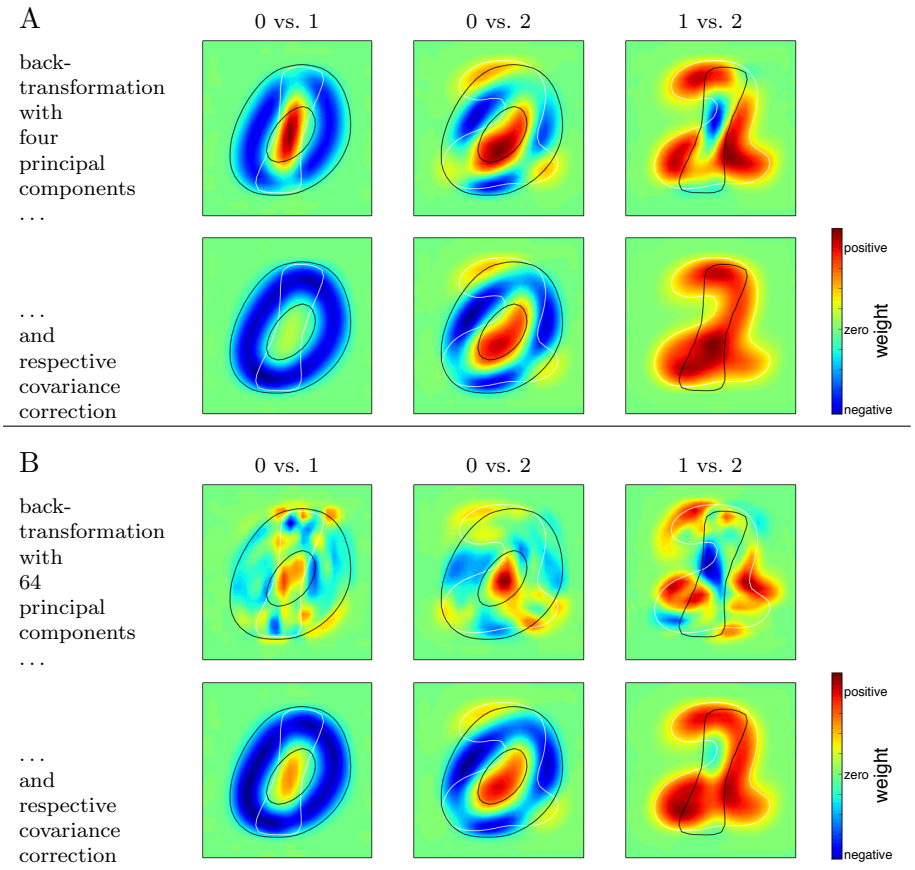

Fig. 2: *Contour plots of backtransformation weights for handwritten digit classification:* The white and black silhouettes display an average contour of the original data (digits 0 vs. 1, 0 vs. 2, and 1 vs. 2). The colored contour plots show the respective weights in the classification process before and after covariance correction with a different number of used principal components (case A and B). Negative weights (blue) are important for the classification of the first class (black silhouette) and positive weights (red) for the second class (white silhouette). Green weights are close to zero and do only contribute weakly to the classification process.

generalize that well and is using a lot of small components instead of focusing on the big shape of the digits.

In case of using only 4 principal components, the approach mainly shows the shape of the digit 2 (or 0 for the first column). In contrast, the visualizations without covariance correction clearly indicate with a blue color which parts are relevant for classifying it as the first class and with the red color which parts are important for the second class. An interesting effect occurs for

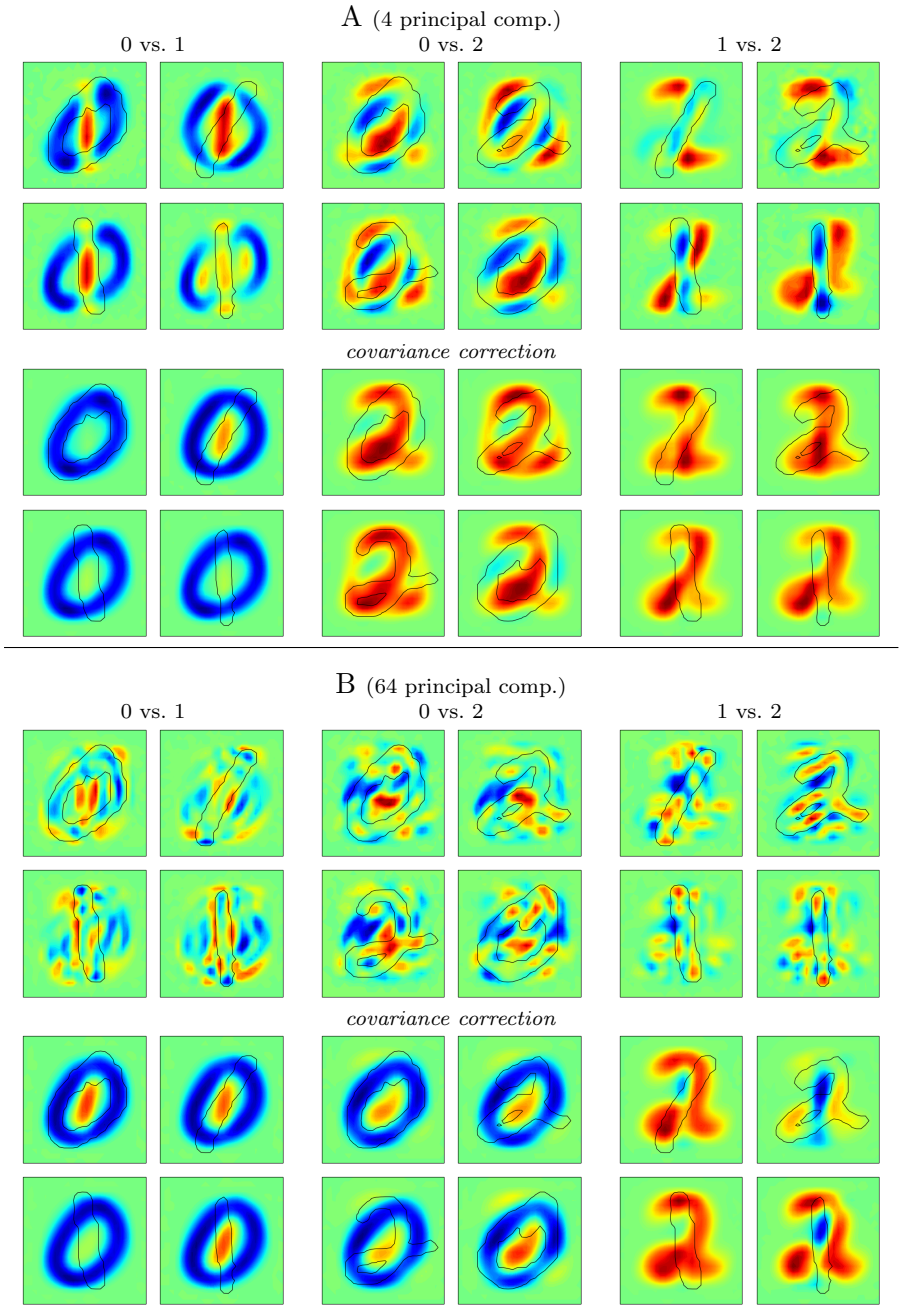

Fig. 3: *Contour plots of backtransformation weights for handwritten digit classification with nonlinear classifier:* The setting is the same as in Fig. 2 except that no average shapes are displayed but the shape of the sample of interest where the backtransformation is calculated for.

the first classifier at the fourth digit (1). Here a closer look could be taken at the classifier and the data to find out why there are yellow weights outside the regular shape of the digit 1. This might be the result of some artifacts in the data (e.g., a sample with very bad handwriting near to the observed sample) or an artifact in the processing.

In the nonlinear and the linear case with 64 principal components the backtransformation reveals that the decision process is not capable of deriving real shape features for the digits. This might be a reason, why a specially tuned deep neural network performs better in this classification task (Schmidhuber, 2012).

3.4 Processing Chain Visualization: Movement Prediction from EEG Data

The electroencephalogram (EEG) is a very complex signal, measuring electrical activity on the scalp with a very high temporal resolution and more than 100 sensors. Several visualization techniques exist for this type of signal, which are used in neuroscience for analysis. When processing EEG data for brain-computer interfaces (BCIs), there is a growing interest in understanding the properties of processing chains and the dynamics of the data, to avoid relying on artifacts and to get information on the original signal back for further interpretation. Here, very often spatial filtering is used for dimensionality reduction to linearly combine the signals from the numerous electrodes to a largely reduced number of new virtual sensors with much less noise (see Section 2.3). These spatial filters and much more importantly the data patterns they are enhancing are visualized with similar methods as used for visualizing data. If the spatial filter is the main part of the processing (e.g., only two filters are used), this approach is sufficient to understand the data processing. However, often more filters and other, additional preprocessing algorithms are used. Hence, the original spatial information cannot be determined for the input of the classifier. This disables a good visualization of the classifier and an understanding of what the classifier learned from the training data. So here, backtransformation can be very helpful.

To illustrate this, a dataset from an EEG experiment was taken (Tabie and Kirchner, 2013). In this experiment, subjects were instructed to move their right arm as fast as possible from a flat board to a buzzer in approximately 30 cm distance. The classification task was to predict upcoming movements by detecting movement-related cortical potentials (Johanshahi and Hallett, 2003) in the EEG single trials. Before applying the backtransformation and visualizing the data as depicted in Fig. 4, the data has been normalized with a standardization, a decimation, and temporal filtering. Only the last part of the signal, which is close to the movement, was visualized. The processing chain was similar to the one in Section 2.3. The details are described by Seeland et al (2013).

The averaged input data in Fig. 4 shows a very strong negative activation at the motor cortex mainly at the left hemisphere around the electrode

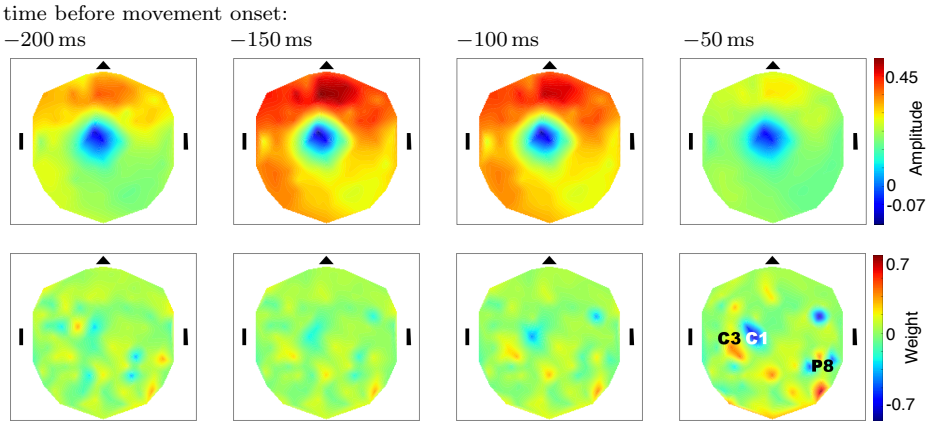

Fig. 4: *Visualization of data for movement prediction and the corresponding processing chain:* In the first row the average of the data before a movement is displayed as topography plots and in the second row the backtransformation weights are displayed, respectively. The data values from the different sensors were mapped to the respective position on the head, displayed as an ellipse with the nose at the top and the ears on the sides.

C1[9]. This activation is consistent with the occurrence of movement related cortical potentials and is expected from the EEG literature (Johanshahi and Hallett, 2003). The region of the activation (blue circle on the left hemisphere at the motor cortex region) is associated with right arm movements, which the subjects had to perform in the experiment.

The backtransformation weights are much more spread over the head compared to the averaged data. There is a major activation at the left motor cortex at electrodes C1 (negative) and C3 (positive), but also a large spread activation at the back of the head at the right hemisphere (around the electrode P8). On the time scale, the most important weights can be found at the last time point, 50 ms before movement onset.

This is reasonable, because the most important movement related information is expected to be just before the movement starts, although movement intention can be detected above chance level on average 460 ms before the movement onset (Lew et al, 2012). Note that the analysis has been performed on single trials and not on averaged data and that for a good classification the largest difference is of interest and not the minimal one. The high weights at C1 and CP3 clearly fit to the high negative activation found in the averaged data and as such highlight the signal of interest. For interpreting the other weights, two things have to be kept in mind. First, EEG data usually contains numerous artifacts and second, due to the conductivity of the skin

[9] A standard extended $10-20$ electrode layout has been chosen with 128 electrodes: `http://www.brainproducts.com/filedownload.php?path=downloads/actiCAP-128-channel-Standard-2_1201.pdf`.

it is possible to measure every electric signal at a certain electrode also on the other electrodes. Keeping that in mind, the activation around P8 could be interpreted as a noise filter for the more important class related signal at C1 and CP3. This required filtering effect on EEG data is closely related to spatial filtering, which emphasizes a certain spatial pattern (Blankertz et al, 2011, section 4.2). It could be also a relevant signal which cannot be observed in the plot of the averaged data. These observations are now a good starting point for domain experts to take a closer look at the raw data to determine which interpretation fits better.

3.5 Applications Beyond Visualization

In the following, we shortly describe two further applications of the affine backtransformation.

3.5.1 Group Ranking

The formula in Equation (18) has the same structure as a normal linear classifier. Each weight gives an information about the importance of certain signal components of the input. Summing up the absolute values of one sensor can now be done in the way as suggested for SVMs (Feess et al, 2013; Lal et al, 2004) to get a sensor ranking:

$$R_h = \sum_{g=1}^{n_g} \left| w_{gh}^{(0)} \right| \ . \tag{23}$$

Such a ranking can then be used for sensor selection algorithms to reduce the number of used electrodes for a BCI and ease comfort and save costs. It can be used in robotics or other applications too, where the reduction of input sources can be beneficial. A similar ranking could be also applied to the time points. The advantage of this ranking method is that it directly operates on the processing chain and not solely on the input data or feature domain. Note, that the quality of the ranking also highly depends on the quality of the processing chain. If a processing chain is worse than an other, chances are high that also its ranking is worse (Krell, 2015, section 3.4.3).

3.5.2 Reinitialization of Linear Classifier with Affine Preprocessing

There could be several reasons for exchanging the preprocessing in a signal processing chain. For example, first some initial preprocessing is loaded but in parallel a new better fitting data specific processing is trained or tuned on new incoming data (e.g., a new spatial filter (Woehrle et al, 2015)). If dimensionality would not be fitting after changing the preprocessing chain, a new classifier would also be needed. But even if dimensions of old and new preprocessing were the same it might be good to *adapt* the classifier to that

change to have a better initialization. Here, the affine backtransformation can be used as described in the following.

For this application, a processing chain of affine transformations is assumed which ends with a sample weighting online learning algorithm like the passive aggressive algorithm or a perceptron. Since the classification function is a weighted sum of samples, it enables the following calculation:

$$w = \sum_i \alpha_i y_i \hat{x}_i = \sum_i \alpha_i y_i (Ax_i + T) = A \sum_i \alpha_i y_i x_i + T \sum_i \alpha_i y_i \tag{24}$$

$$= Aw^{(0)} + Tb \text{ with } w^{(0)} = \sum_i \alpha_i y_i x_i \text{ and } b = \sum_i \alpha_i y_i \; . \tag{25}$$

Here, x_i is the training data with the training labels y_i and $\hat{x}_i$ is the preprocessed training data given to the classifier. The weights α_i are calculated by update formulas of the classifier. During the update step, $w^{(0)}$ must be calculated additionally but neither x_i, y_i, nor α_i are stored. When changing the preprocessing from (A, T) to (A', T')

$$w' = A' w^{(0)} \tag{26}$$

is a straightforward estimate for the new classifier. The advantage of this formula is, that it just requires additionally calculating and storing $w^{(0)}$. So the resulting classifier can be still used for memory efficient online learning. Especially, even if neither (A', T') nor (A, T) is known, w' can be calculated using the new signal processing function $\hat{F}(x) = A'x + T'$:

$$w' = A' w^{(0)} = \hat{F}(w^{(0)}) - T'b = \hat{F}(w^{(0)}) - 0A' w^{(0)} b - T'b = \hat{F}(w^{(0)}) - \hat{F}(0 w^{(0)}) b \; . \tag{27}$$

So, w' can be computed by processing $w^{(0)}$ and a sample of zero entries in the signal processing chain. This only requires some minor processing time but no additional resources. Usually the processing chain is very fast and so the additional processing time should not be a problem. For giving a proof of concept, this application of the backtransformation was used in a setting, where the preprocessing was randomly changed. With the aforementioned approach the change could be perfectly compensated without any loss in performance (Krell, 2015, section 2.4.6).

4 Conclusion

In this paper, a direct approach is given to look at the complete data processing chain (in contrast to separate handling of its components) and to transform it to a representation in the same format as the data. This could be used to improve the understanding of complex processing chains and might enable several applications in future. It was shown that backtransformation can be used for visualization of the decision process and a direct comparison with a visualization of the data is possible and enables an interpretation of the processing. Our approach extends existing algorithms by also considering the

preprocessing, by putting no restrictions on the decision algorithm, by providing the implementation details, and integrating the backtransformation in the pySPACE framework which already comes with a large number of available algorithms.

Backtransformation can be used for interpreting the behaviour of the decision process, but it remains an open question on how the further analysis is performed, so that additional investigations and expert knowledge might be required. A related problem occurs when using temporal and spatial filters. Here the solution is to visualize the frequency response and the spatial pattern instead of the pure weights of the transformation. The frequency response gives information on how frequencies are filtered out and spatial patterns give information on which signal in space is emphasized by the respective spatial filter. It would be interesting to develop new methods, which improve the interpretability of the decision process, e.g., by extending the method of covariance multiplication with a more sophisticated calculation of the covariance matrix or by deriving a different formula for getting the forward model. This might enable the backtransformation to reveal new signals or connections in the data which can then be used to improve the observed data processing chain.

In future, it would be interesting to further analyze the application of the backtransformation, e.g., by using other data or processing chains, by analyzing regression problems, or by integrating it into other algorithms and analyzing its benefit.

References

Abdi H, Williams LJ (2010) Principal component analysis. Wiley Interdisciplinary Reviews: Computational Statistics 2(4):433–459, DOI 10.1002/wics.101

Aksoy S, Haralick RM (2001) Feature normalization and likelihood-based similarity measures for image retrieval. Pattern Recognition Letters 22(5):563–582, DOI 10.1016/S0167-8655(00)00112-4

Baehrens D, Schroeter T, Harmeling S, Kawanabe M, Hansen K, Müller KR (2010) How to Explain Individual Classification Decisions. Journal of Machine Learning Research 11:1803–1831

Blankertz B, Tomioka R, Lemm S, Kawanabe M, Müller KR (2008) Optimizing spatial filters for robust EEG single-trial analysis. IEEE Signal Processing Magazine 25(1):41–56, DOI 10.1109/MSP.2008.4408441

Blankertz B, Lemm S, Treder M, Haufe S, Müller KR (2011) Single-trial analysis and classification of ERP components–a tutorial. NeuroImage 56(2):814–825, DOI 10.1016/j.neuroimage.2010.06.048

Chang CC, Lin CJ (2011) LIBSVM. ACM Transactions on Intelligent Systems and Technology 2(3):1–27, DOI 10.1145/1961189.1961199

Chen Ch, Härdle W, Unwin A (2008) Handbook of Data Visualization. Springer Handbooks of Computational Statistics, Springer

611 Clarke F (1990) Optimization and Nonsmooth Analysis. Society for Industrial
612 and Applied Mathematics, DOI 10.1137/1.9781611971309
613 Crammer K, Dekel O, Keshet J, Shalev-Shwartz S, Singer Y (2006) Online
614 Passive-Aggressive Algorithms. Journal of Machine Learning Research 7:551
615 – 585
616 Domingos P (2012) A few useful things to know about machine learning. Com-
617 munications of the ACM 55(10):78–87, DOI 10.1145/2347736.2347755
618 Feess D, Krell MM, Metzen JH (2013) Comparison of sensor selection mecha-
619 nisms for an ERP-based brain-computer interface. PLoS ONE 8(7):e67,543,
620 DOI 10.1371/journal.pone.0067543
621 Ghaderi F, Straube S (2013) An adaptive and efficient spatial filter for event-
622 related potentials. In: Proceedings of the 21st European Signal Processing
623 Conference, (EUSIPCO)
624 Griewank A, Walther A (2008) Evaluating Derivatives: Principles and Tech-
625 niques of Algorithmic Differentiation. Society for Industrial and Applied
626 Mathematics
627 Haufe S, Meinecke F, Görgen K, Dähne S, Haynes JD, Blankertz B, Bießmann
628 F (2014) On the interpretation of weight vectors of linear models in mul-
629 tivariate neuroimaging. NeuroImage 87:96–110, DOI 10.1016/j.neuroimage.
630 2013.10.067
631 Johanshahi M, Hallett M (eds) (2003) The Bereitschaftspotential: movement-
632 related cortical potentials. Kluwer Academic/Plenum Publishers, New York,
633 USA
634 Jutten C, Herault J (1991) Blind separation of sources, part I: An adaptive
635 algorithm based on neuromimetic architecture. Signal Processing 24(1):1–
636 10, DOI 10.1016/0165-1684(91)90079-X
637 Kirchner EA, Kim SK, Straube S, Seeland A, Wöhrle H, Krell MM, Tabie
638 M, Fahle M (2013) On the applicability of brain reading for predictive
639 human-machine interfaces in robotics. PLoS ONE 8(12):e81,732, DOI
640 10.1371/journal.pone.0081732
641 Krell MM (2015) Generalizing, Decoding, and Optimizing Support Vector
642 Machine Classification. PhD thesis, University of Bremen, Bremen, URL
643 http://nbn-resolving.de/urn:nbn:de:gbv:46-00104380-12
644 Krell MM, Wöhrle H (2015) New one-class classifiers based on the origin
645 separation approach. Pattern Recognition Letters 53:93–99, DOI 10.1016/j.
646 patrec.2014.11.008
647 Krell MM, Straube S, Seeland A, Wöhrle H, Teiwes J, Metzen JH, Kirchner
648 EA, Kirchner F (2013a) pySPACE - a signal processing and classification
649 environment in Python. Frontiers in Neuroinformatics 7(40), DOI 10.3389/
650 fninf.2013.00040, https://github.com/pyspace
651 Krell MM, Tabie M, Wöhrle H, Kirchner EA (2013b) Memory and Processing
652 Efficient Formula for Moving Variance Calculation in EEG and EMG Sig-
653 nal Processing. In: Proc. International Congress on Neurotechnology, Elec-
654 tronics and Informatics (NEUROTECHNIX 2013), ScitePress, Vilamoura,
655 Portugal, pp 41–45, DOI 10.5220/0004633800410045

Krell MM, Feess D, Straube S (2014a) Balanced Relative Margin Machine The missing piece between FDA and SVM classification. Pattern Recognition Letters 41:43–52, DOI 10.1016/j.patrec.2013.09.018

Krell MM, Straube S, Wöhrle H, Kirchner F (2014b) Generalizing, Optimizing, and Decoding Support Vector Machine Classification. In: ECML/PKDD 2014 PhD Session Proceedings, Nancy

LaConte S, Strother S, Cherkassky V, Anderson J, Hu X (2005) Support vector machines for temporal classification of block design fMRI data. NeuroImage 26(2):317–329, DOI 10.1016/j.neuroimage.2005.01.048

Lagerlund TD, Sharbrough FW, Busacker NE (1997) Spatial filtering of multichannel electroencephalographic recordings through principal component analysis by singular value decomposition. Journal of Clinical Neurophysiology 14(1):73–82

Lal TN, Schröder M, Hinterberger T, Weston J, Bogdan M, Birbaumer N, Schölkopf B (2004) Support vector channel selection in BCI. IEEE Engineering in Medicine and Biology Society 51(6):1003–1010, DOI 10.1109/TBME.2004.827827

Le QV, Ranzato M, Monga R, Devin M, Chen K, Corrado GS, Dean J, Ng AY (2012) Building high-level features using large scale unsupervised learning. In: International Conference on Machine Learning

LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11):2278–2324, DOI 10.1109/5.726791

Lew E, Chavarriaga R, Zhang H, Seeck M, del Millan J (2012) Self-paced movement intention detection from human brain signals: Invasive and non-invasive EEG. In: 2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), pp 3280–3283

Lin HT, Lin CJ, Weng RC (2007) A note on Platts probabilistic outputs for support vector machines. Machine Learning 68(3):267–276, DOI 10.1007/s10994-007-5018-6

Metzen JH, Kirchner EA (2011) Rapid adaptation of brain reading interfaces based on threshold adjustment. In: Proceedings of the 2011 Conference of the German Classification Society, (GfKl-2011), Frankfurt, Germany, p 138

Mika S, Rätsch G, Müller KR (2001) A mathematical programming approach to the kernel fisher algorithm. In: Advances in Neural Information Processing Systems 13 (NIPS 2000), MIT Press, pp 591–597

Oppenheim AV, Schafer RW (2009) Discrete-Time Signal Processing, 3rd edn. Prentice Hall Press

Platt JC (1999) Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods

Press W (2007) Numerical recipes: the art of scientific computing, 3rd edn. Cambridge University Press

Rieger J, Kosar K, Lhotska L, Krajca V (2004) Eeg data and data analysis visualization. In: Barreiro J, Martn-Snchez F, Maojo V, Sanz F (eds) Biological and Medical Data Analysis, Lecture Notes in Computer Science, vol 3337, Springer Berlin Heidelberg, pp 39–48, DOI 10.1007/978-3-540-30547-7_5

Rivet B, Souloumiac A, Attina V, Gibert G (2009) xDAWN Algorithm to Enhance Evoked Potentials: Application to Brain-Computer Interface. IEEE Transactions on Biomedical Engineering 56(8):2035–2043, DOI 10.1109/TBME.2009.2012869

Rockafellar RT, Wets RJB (2009) Variational analysis, vol 317. Springer Science & Business Media

Schmidhuber J (2012) Multi-column deep neural networks for image classification. In: Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE Computer Society, pp 3642–3649

Schölkopf B, Platt JC, Shawe-Taylor J, Smola AJ, Williamson RC (2001) Estimating the support of a high-dimensional distribution. Neural computation 13(7):1443–1471, DOI 10.1162/089976601750264965

Seeland A, Wöhrle H, Straube S, Kirchner EA (2013) Online movement prediction in a robotic application scenario. In: 6th International IEEE EMBS Conference on Neural Engineering (NER), San Diego, USA, pp 41–44, DOI 10.1109/NER.2013.6695866

Smola AJ, Schölkopf B (2004) A tutorial on support vector regression. Statistics and Computing 14(3):199–222, DOI 10.1023/B:STCO.0000035301.49549.88

Steinwart I, Christmann A (2008) Support Vector Machines. Springer

Straube S, Feess D (2013) Looking at ERPs from Another Perspective: Polynomial Feature Analysis. Perception 42 ECVP Abstract Supplement:220

Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R (2014) Intriguing properties of neural networks. In: International Conference on Learning Representations

Tabie M, Kirchner EA (2013) EMG Onset Detection - Comparison of different methods for a movement prediction task based on EMG. In: Alvarez S, Solé-Casals J, Fred A, Gamboa H (eds) In Proceedings of the 6th International Conference on Bio-inspired Systems and Signal Processing (BIOSIGNALS-13), SciTePress, Barcelona, Spain, pp 242–247, DOI 10.5220/0004250102420247

Vapnik VN (1995) The nature of statistical learning theory. Springer

Varewyck M, Martens JP (2011) A practical approach to model selection for support vector machines with a Gaussian kernel. IEEE transactions on systems, man, and cybernetics Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society 41(2):330–340, DOI 10.1109/TSMCB.2010.2053026

Verhoeye J, de Wulf R (1999) An Image Processing Chain for Land-Cover Classification Using Multitemporal ERS-1 Data. Photogrammetric Engineering & Remote Sensing 65(10):1179–1186

Woehrle H, Krell MM, Straube S, Kim SK, Kirchner EA, Kirchner F (2015) An Adaptive Spatial Filter for User-Independent Single Trial Detection of Event-Related Potentials. IEEE Transactions on Biomedical Engineering DOI 10.1109/TBME.2015.2402252