Lecture Machine Learning Algorithms
LVA 389.204

Univ. Prof. DI. Dr.-Ing. Markus Rupp
Institute of Telecommunications
Technische Universität Wien

December 29, 2021

Contents

Chapter 1

Basics Machine Learning

1.1 Nomenclature

The following nomenclature is being applied throughout the script.

$*$	for conjugate complex
T	for transpose
H	for Hermitian, thus transpose and conjugate complex.
$a, b, c...$	is a (deterministic) scalar.
$\mathbf{a}, \mathbf{b}, \mathbf{c}, ...$	is a random variable with density function $f_{\mathbf{a}}(a)$, variance $\sigma^2_{\mathbf{a}}$ and mean $\bar{a}$.
$\underline{a}$,$\underline{b}$,$\underline{c}$...	is a (column-)vector with a number of elements.
$\underline{1}$	is a vector with ones as elements.
$\underline{\mathbf{a}}, \underline{\mathbf{b}}, \underline{\mathbf{c}}$...	is a (column-) vector whose entries are random variables. its joint density is given by $f_{\underline{\mathbf{a}}}(\underline{a})$. its autocorrelation matrix is given by $R_{\underline{\mathbf{aa}}} = \mathrm{E}[\underline{\mathbf{aa}}^H]$.
$A, B, C, ...$	are matrices whose entries are scalars.
$\mathbf{A}, \mathbf{B}, \mathbf{C}, ...$	are matrices whose entries are random variables.
I	is the unit matrix.
$\|\underline{a}\|_p^q$	applied to vector $\underline{a}$ is its $p-$norm of power q. $\|\underline{a}\|_p^q = (\sum_i \|a_i\|^p)^{q/p}$
$\|\underline{a}\|_Q$	applied to vector $\underline{a}$ means weighted norm: $\sqrt{\underline{a}^H Q \underline{a}}$.

Attaching an argument (k) for scalars or an index $_k$ (small letter) on vectors or matrices indicates a further temporal relation: a sequence. Random variables thus become random processes. If a capital letter is attached then it defines a dimension. Positive definite square matrices are denoted by $A > 0$. Consequently, all eigenvalues of A are larger than zero.

Linear filters are denoted by capital letters and often to stress the operation attached

with a further argument q^{-1}. We write for example $B(q^{-1})[v(k)] = B[v(k)]$ for a sequence $v(k)$, filtered by an FIR filter with coefficients $b(0), b(1)...b(M-1)$. Also recursive filter structures (IIR) can be denoted easily by: $\frac{B(q^{-1})}{1-A(q^{-1})}[v(k)]$. This is an IIR filter of the form:

$$y(k) = \sum_{l=1}^{N} a(l)y(k-l) + \sum_{l=0}^{M-1} b(l)v(k-l). \tag{1.1}$$

We prefer this notation to the Z-transform as we do not require any conditions on the input sequence (Dirichlet condition).

As we operate a lot with matrices and vectors we will introduce some short notations, in particular differentiating with respect to real and complex valued vectors. We will use the following rules (for $\underline{w} \in I\!R$ and $\underline{z} \in \mathbb{C}$), see also Appendix A for a more detailed discussion:

$$\begin{aligned}
\frac{\partial R\underline{w}}{\partial \underline{w}} &= R \\
\frac{\partial \underline{w}^T R\underline{w}}{\partial \underline{w}} &= \underline{w}^T[R + R^T] \\
\frac{\partial R\underline{z}}{\partial \underline{z}} &= R \\
\frac{\partial \underline{z}^H R\underline{z}}{\partial \underline{z}} &= \underline{z}^H R.
\end{aligned}$$

1.2 Historical Notes

The following Figure 1.1 depicts the essential milestones in the field of machine learning over the past 80 years. Starting with the first neural models, se co called McCulloch-Pits model, an signal processing equivalent, the perceptron evolved end of the 50ies. Widrow and Hoff derived first adaptive algorithms to train a simple one layer network of such perceptrons. in the 80ies Rummelhard et al. proposed the backpropagation algorithm to train multi layer networks. At this time the field was often referred to as pattern recognition. Essential inputs, the vector support machine, came in the middle of the 90ies when Vapnik moved from Russia to the US and worked at Bell Labs.

The latest innovation step was deep learning, tha tis the substantial increase of the number of hidden layers. The first general, working learning algorithm for supervised, deep, feedforward, multilayer perceptrons was published by Alexey Ivakhnenko and Lapa in 1965. A 1971 paper described a deep network with 8 layers trained by the group method of data handling algorithm. Other deep learning working architectures, specifically those built for computer vision, began with the Neocognitron introduced by Kunihiko Fukushima in 1980.

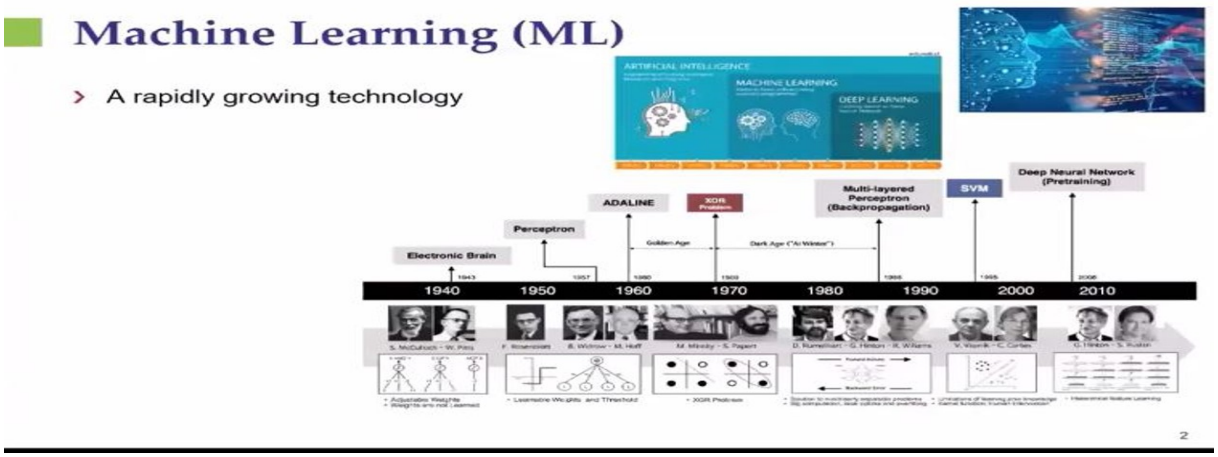

Figure 1.1: *Historical development of the field*

The term Deep Learning was introduced to the machine learning community by Rina Dechter in 1986, and to artificial neural networks by Igor Aizenberg and colleagues in 2000, in the context of Boolean threshold neurons.

In 1989, Yann LeCun et al. applied the standard backpropagation algorithm, which had been around as the reverse mode of automatic differentiation since 1970, to a deep neural network with the purpose of recognizing handwritten ZIP codes on mail. While the algorithm worked, training required 3 days.

In 1995, Brendan Frey demonstrated that it was possible to train (over two days) a network containing six fully connected layers and several hundred hidden units using the wake-sleep algorithm, co-developed with Peter Dayan and Hinton. Many factors contribute to the slow speed, including the vanishing gradient problem analyzed in 1991 by Sepp Hochreiter. In 2006, publications by Geoff Hinton, Ruslan Salakhutdinov, Osindero and Teh showed how a many-layered feedforward neural network could be effectively pre-trained one layer at a time, treating each layer in turn as an unsupervised restricted Boltzmann machine, then fine-tuning it using supervised backpropagation. The papers referred to learning for deep belief nets.

Advances in hardware enabled the renewed interest. In 2009, Nvidia was involved in what was called the big bang of deep learning, as deep-learning neural networks were trained with Nvidia graphics processing units (GPUs). That year, Google Brain used Nvidia GPUs to create capable DNNs. While there, Andrew Ng determined that GPUs could increase the speed of deep-learning systems by about 100 times. In particular, GPUs are well-suited for the matrix/vector math involved in machine learning. GPUs speed up training algorithms by orders of magnitude, reducing running times from weeks to

days. Specialized hardware and algorithm optimizations can be used for efficient processing.

In March 2019, Yoshua Bengio, Geoffrey Hinton and Yann LeCun were awarded the Turing Award for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing. A good read is[1].

1.3 Machine Learning

Machine Learning deals with what we call today "Big Data" that is vast amounts on data as are likely available via the internet. Such data may however be of different origin as Figure 1.2 depicts. Image and video data are often the main interest in order to find a needle in the haystack, for example a missing or criminal person. In the last years financial data have become more and more of interest mostly to detect illegal financial activities but also to predict the behavior of markets. Finally, sensor data often in the context of environment data or medical ones are of interest for example to predict bad weather or ill conditions of persons.

Data Type	Feature Selection Task	Techniques
Sensor data	Extract signal properties from raw sensor data to create higher-level information	**Peak analysis** – perform an fft and identify dominant frequencies **Pulse and transition metrics** – derive signal characteristics such as rise time, fall time, and settling time **Spectral measurements** – plot signal power, bandwidth, mean frequency, and median frequency
Image and video data	Extract features such as edge locations, resolution, and color	**Bag of visual words** – create a histogram of local image features, such as edges, corners, and blobs **Histogram of oriented gradients (HOG)** – create a histogram of local gradient directions **Minimum eigenvalue algorithm** – detect corner locations in images **Edge detection** – identify points where the degree of brightness changes sharply
Transactional data	Calculate derived values that enhance the information in the data	**Timestamp decomposition** – break timestamps down into components such as day and month **Aggregate value calculation** – create higher-level features such as the total number of times a particular event occurred

Figure 1.2: *Typical data for machine learning applications.*

There is also a huge amount of possible ways to treat such data in order to extract desired knowledge. Figure 1.3 shows an overview of the most common areas of machine learning algorithms. We will certainly not cover all of those as this would go beyond the cope of a single lecture. Our focus will be on the underlying algorithms, their design and their properties. In particular we like to learn how we learn. Take for example the learning of a poem by heart. This is a different form of learning then how to drive a car (conceptual learning), and even different to proving a theorem (learning by understanding).

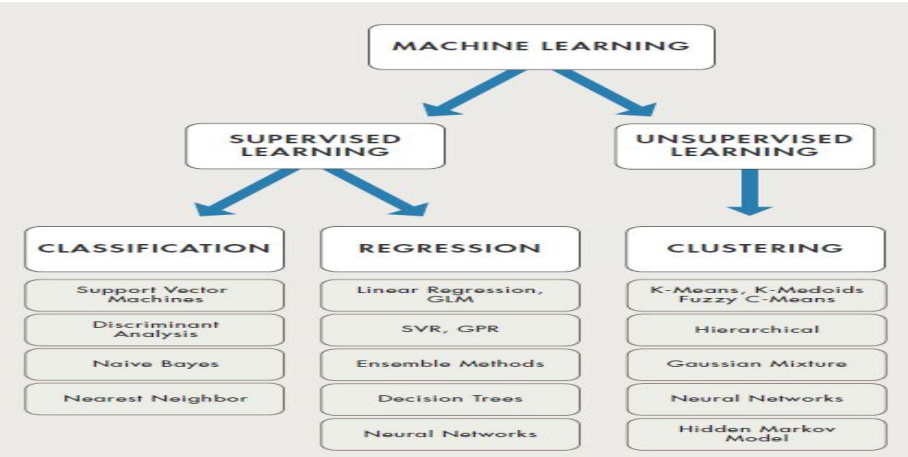

Figure 1.3: *Overview of machine learning algorithms.*

1.4 Applications

The most common application for machine learning is classification, that is to associate some date with a particular attribute which makes them "to belong to class A" while others not. Once we know about the data elements and their association with class A we can use a so called supervised learning, while in case we do not know such dependency and would like to know if there exists any, we talk about unsupervised learning. The concept of belonging to a class or not can easily be extended to separating into multiple classes. As the basic problem is a binary classification, most of the lecture will discuss this problem only. In Figure 1.4 two classical applications of machine learning are shown. In the left part of the figure we see a linear regression in which a line is fitted to pairs of data in order to match their behavior best. This serves to predict the data behavior once only the x-data is provided, we thus can make predictions with this approach. On the right hand side we see tow classes (blue crosses and red diamonds) both crammed together on different corners. Here we like to find a line that separates both groups. We call the x-data: input, regressor or independent variable, while the y-data are: regressand, dependend variable or observation.

To find such a discriminating line is not straightforward, in particular not if the input data is of several dimensions. Let us assume that we have a set of N data with M-

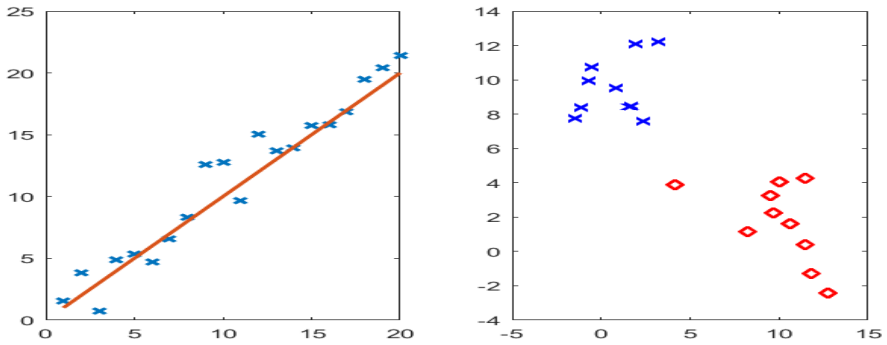

Figure 1.4: *Regression versus classification.*

dimensional input vectors $\underline{x}_i; i = 1, 2, ..., N$. Then we have to find a linear combination

$$y_i = c_0 + c_1 x_{i,1} + c_2 x_{i,2} + ... + c_M x_{i,M} + v_i \tag{1.2}$$

$$= [c_0, c_1, ..., c_M] \begin{bmatrix} 1 \\ x_{i,1} \\ \vdots \\ x_{i,M} \end{bmatrix} + v_i \tag{1.3}$$

$$= \underline{c}^H \hat{\underline{x}}_i + v_i, \tag{1.4}$$

where we introduced now an augmented vector $\hat{\underline{x}}_i$ to indicate that we have added a constant 1 as first element. Such constant is indeed very necessary to shift the line (a hyperplane in general) up or downwards. We refer to this artificial input value as bias. We can now describe the hyperplane by including the observation y_i in normal form:

$$[1, \underline{x}_i, y_i]^H \begin{bmatrix} \underline{c} \\ -1 \end{bmatrix} = 0.$$

We have used here the most general form in which data are complex values. In most applications they however will be real valued and the Hermitian operator needs to be replaced by a transposition.

Regression: Let us formulate the regression problem in terms of Least Squares (LS), i.e.,

$$\min_{\underline{c}} \sum_{i=1}^{N} |v_i|^2 = \min_{\underline{c}} \|\underline{v}\|_2^2$$

$$= \min_{\underline{c}} \sum_{i=1}^{N} |y_i - \hat{\underline{x}}_i^H \underline{c}|^2$$

$$= \min_{\underline{c}} \|\underline{y} - X\underline{c}\|_2^2,$$

where we have combined all input vectors into a matrix X. The LS solution is well known and given by

$$\underline{c}_{LS} = \begin{cases} [X^H X]^{-1} X^H \underline{y} & ; N \geq M+1 \\ X^H [X X^H] \underline{y} & ; \text{else} \end{cases} .$$

In a case a linear solution (hyperplane) does not suffice, the formulation can be extended towards a nonlinear representation by

$$\begin{aligned} y_i &= c_0 + c_1 x_{i,1} + ... + c_M x_{i,M} + \\ &\quad + c_{M+1} x_{i,1}^2 + ... + c_{2M} x_{i,M}^2 + \\ &\quad + ... \\ &\quad + c_{KM} x_{i,1}^{K+1} + .. + c_{(K+1)M} x_{i,M}^{K+1} + v_i \\ &= [1, \underline{\tilde{x}}_i^H] \underline{c} + v_i. \end{aligned}$$

However, note that mixing terms such as $x_i x_k$ have not been included. The entire description explodes quickly in terms of coefficients and is thus rarely a good approach. Nevertheless, strictly speaking the problem remains linear in the coefficients and thus a LS problem. However, the Gramian XX^H can now become poorly conditioned and thus numerically challenging to solve.

Classification: A different application is pattern recognition. Here, (binary) patterns need to be classified. Take for example a digital camera delivering images of an automatic bottling machine. It needs to be recognized whether the bottle is of correct size and whether the onprint is correct. The images from the camera are compared to reference images and then it is decided whether the required conditions are satisfied or not. Well suited for such decisions are neural networks as will be shown later. For now, assume we are given a set of pairs $\{\underline{x}_i, y_i\}; i = 1, 2, ..., N$, very similar as in the regression problem before. Often, however, a preprocessing if the input vectors is required to obtain vectors with the desired features. We thus assume that there exists a unique mapping $\underline{x}_i : f(\underline{x}_i) \rightarrow \underline{f}_i$ which now will serve as input vectors. The values y_i are either "1". indicating that the corresponding vector $\underline{x}_i$ belongs to a class, say A, or not if $y_i = -1$. The problem is depicted in Figure 1.5 The problem of finding a separating line as indicated in the figure is to find two parameters, say the slope a and the offset y_0, resulting in

$$\begin{aligned} y &= ax + y_0 \\ 0 &= [1, x, y] \begin{bmatrix} y_0 \\ a \\ -1 \end{bmatrix} = \left\langle \hat{\underline{f}}, \underline{w} \right\rangle, \end{aligned}$$

where we combined all three coefficients (including -1) in vector $\underline{w}$ and the input (including the bias term) is in $\hat{\underline{f}}$. If the feature vector $\hat{\underline{f}}_i$ belongs to class A, then the corresponding $y_i = 1$. Let us assume for now that the opposite set of feature vectors that belong top

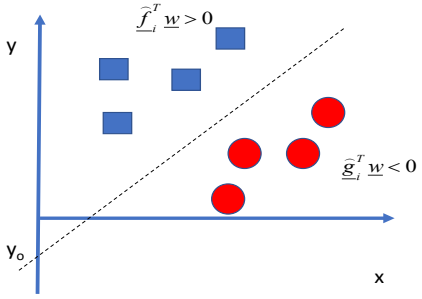

Figure 1.5: *Binary classification.*

$y_i = -1$ (not in class A) are denoted by $\hat{\underline{g}}_i$. Augmenting We can now formulate the problem as LS problem:

$$\underline{w}_{opt} = \arg\min_{\underline{w}} \sum_{i=1}^{N_1} \left(1 - \left\langle \underline{w}, \hat{\underline{f}}_i \right\rangle\right)^2 + \sum_{i=1}^{N_2} \left(-1 - \left\langle \underline{w}, \hat{\underline{g}}_i \right\rangle\right)^2,$$

where we used N_1 occurrences of feature vector $\hat{\underline{f}}_i$ and N_2 for $\hat{\underline{g}}_i$. The solution can be given explicitly assuming the $N_1 + N_2 \geq 3$:

$$\underline{w}_{opt} = \left[\sum_{i=1}^{N_1} \hat{\underline{f}}_i \hat{\underline{f}}_i^T + \sum_{i=1}^{N_2} \hat{\underline{g}}_i \hat{\underline{g}}_i^T\right]^{-1} \left(\sum_{i=1}^{N_1} \hat{\underline{f}}_i - \sum_{i=1}^{N_2} \hat{\underline{g}}_i\right).$$

In the following Figure 1.6 we can recognize the obtained LS solution for such a problem. Furthermore, there is also a solution of an iterative classifier shown which also solves the problem but possibly not as good as LS as the line even touches one of the elements. In Figure 1.7 we see another example in which obviously LS does not provide a good solution. Here, the number N_2 of the blue dots is much larger than N_1 of the red dots and although a perfect discrimination exists, LS does not select it. This shows that our formulation of the problem was a bit too naive and needs to be improved.[1] A potential solution to our formulation is to assign other values than $\{-1, 1\}$ but $\{\alpha, \beta\}$ to the binary classification

[1] The example is from Prof. Aggelos Pikrakis who spend a sabbatical at TU Wien in Vienna giving a course on machine learning.

problem. By proper selecting α and β we can shift the separation line. However, as this is not known beforehand, it is not considered a practical solution.

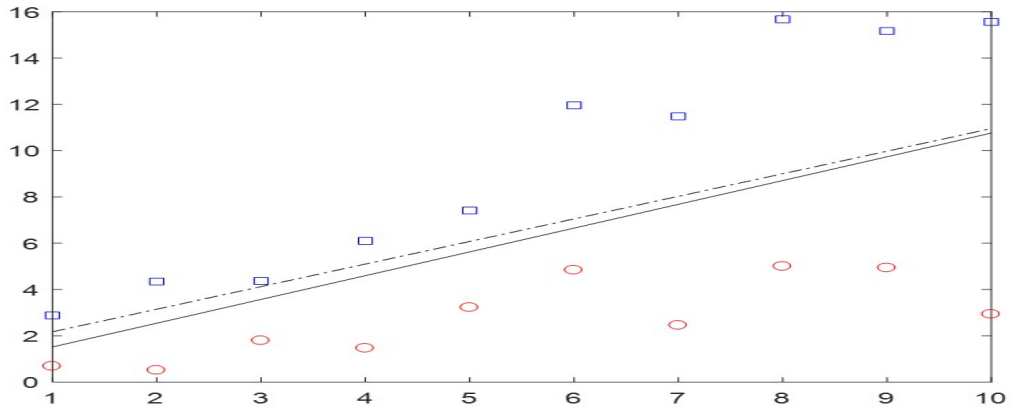

Figure 1.6: *Automatic classification by LS and iterative approach.*

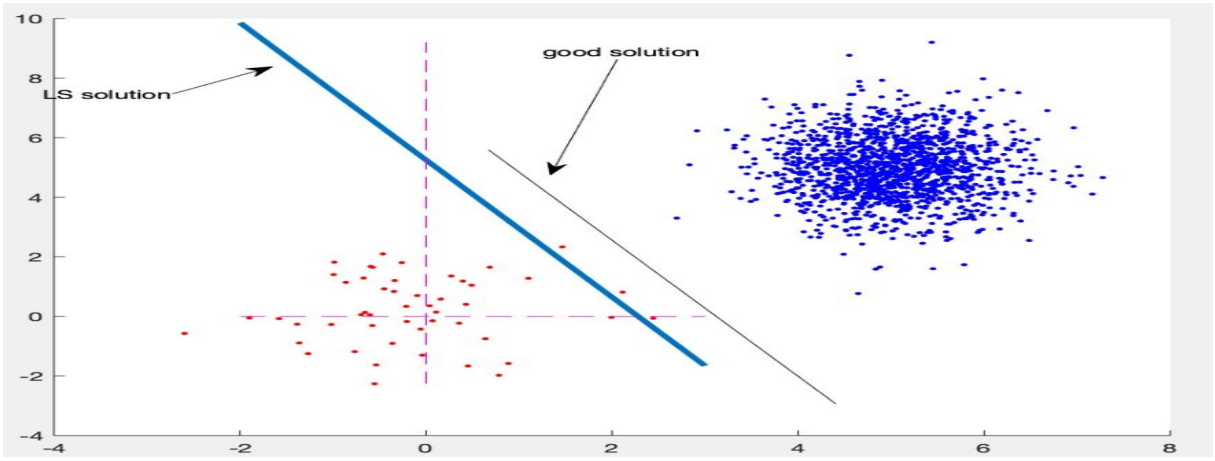

Figure 1.7: *An example in which LS fails to classify.*

Equalization: A further application of adaptive filters can be found in data transmissions over frequency selective (time-dispersive) channels. Figure 1.8 shows the problem of data transmission when employing an adaptive filter for equalization. A digital signal is being transmitted via a linear channel c and additively disturbed by noise $v(n)$. At the receiver a digital filter is to be found that the following nonlinear mapping on the transmission alphabet guarantees the correct mapping onto the transmission symbols. The problem is often less in the choice of a suitable filter structure or the large size of coefficients but to find

the suitable algorithm that allows for **rapid tracking** of channel alternations (frequency dispersive or time-variant channel) and at the same time does not require high demands on **numerical precision**. Different to the previous system identification, we do not have a reference signal now. It can be obtained by introducing a training sequence known at the receiver. However, the concept of data transmission is to transmit **unknown data**, and thus the training sequence needs to be very small compared to the unknown data. A second method is to use decoded signals as reference signals. This only works as long as the errors remain small.

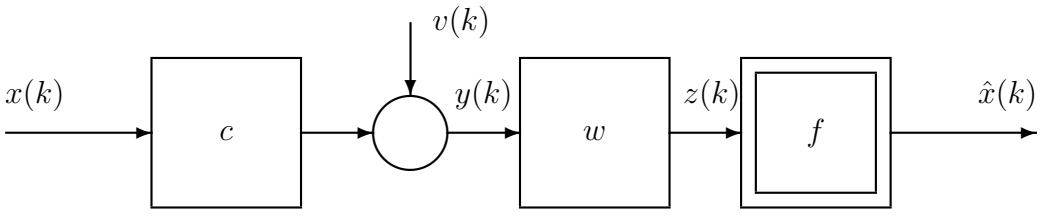

Figure 1.8: *Optimal equalizer and decoder.*

1.5 Architectures

A classical neural network is constructed by several layers of perceptrons, thus linear combiners followed by a memoryless nonlinear function. The fist layer is called the input layer, the last the output layer. All other layers are denominated as hidden layers. Nevertheless, depending on the application such a neural network can have different architectures. We have already mentioned the possibility to convert the observations in to a feature vector, which is nothing else but a preprocessing unit. In image and video applications where it is essential to find specific objects in a scene or even understand an entire scene, preprocessing units to detect edges are of advantage. Similar if specific tumors for medical applications are the the focus, a matched filter maybe useful. In such cases so-called convolutional neural networks (CNN) have proven useful. Figure 1.9 depicts such a CNN. Here convolutional layers and so called pooling layers are alternating, the later simply performs some form of down-sampling. Note that in the neural network community often the wording differs from the signal processing community. Here three terms that are used very often:

- **stride**: means subsampling. Applying a 2D convolutional filter can cause a subsampling, if the filter is not applied to every pixel but every n-th pixel. A conventional convolution thus uses a stride of 1, a subsampling by four is equivalent to a stride of 2 (2 in horizontal and 2 in vertical). Rarely, the stride in horizontal and vertical

differs. In this case the stride is a tuple, e.g., (2,3).

- **padding**: means zero filling. In particular when applying a convolutional (2D) filter on the border, some initial values are missing. In this case the original input is augmented by additional zeros to ensure the resulting outcome is of the same dimension.

- **pooling**: means selecting. Pooling typically follows a convolutional layer to reduce the amount of data. In pooling we replace a set of related data (window size) by a single value. If, for example, in a 3x3 window only the centre value is taking, we have a down-sampling If, for example, in a 3x3 window the maximal value is taken, we have so called max pooling If, for example, in a 3x3 window the average value is taken, we have an average pooling.

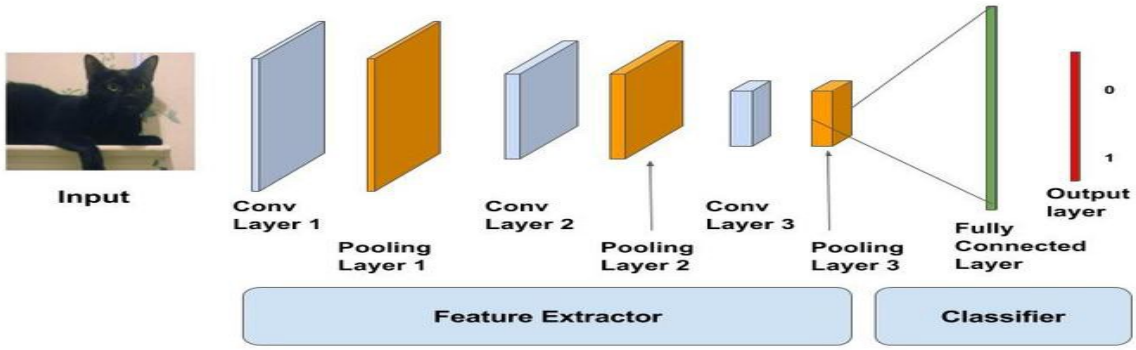

Figure 1.9: *A convolutional neural network architecture.*

In applications where temporal correlations are of importance such as speech processing, so-called recurrent neural networks (RNN) have been proven useful. Figure 1.10 depicts such a simple RNN comprising of a decision operation $f()$ and a feedback in the decision path. At the training phase many patterns x are presented to the input together with decisions d. The neural network has to find the correct set of coefficients a and b to guarantee that similar pattern result in the same decisions. In general such networks may contain several layers. Typically the feedback part only influences the hidden layers.

Another important structure is given in form of so-called autoencoders. One example if provided in Figure 1.11. We recognize that the size of the hidden layers shrinks towards the center and then increases again. The idea here is to reduce complexity preserving information layer by layer towards the middle layer. If all information is still present, then the remaining part can reinstall the original signal representation. Autoencoders are thus

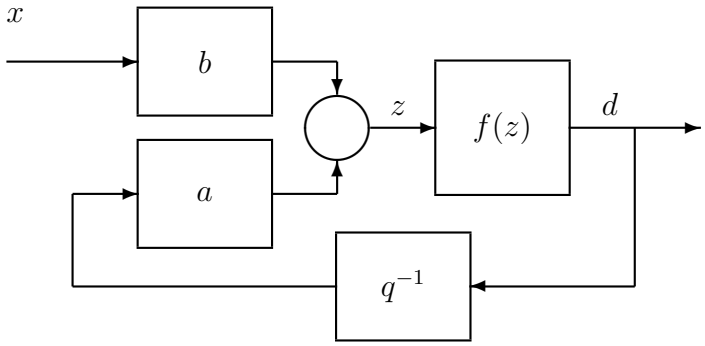

Figure 1.10: *Neuronal network with feedback.*

trained by the same data as input and output. If the error at the output layer is very small the data compression was successful.This feature can alo be used as a classification. Once an autoencoder is trained for a specific class of inputs, the corresponding mean squared error (average difference between input and output) is supposed to be small. If a new input is presented resulting in a large mean squared error, it does not belong to this category.

The second part of an autoencoder is typically the inverse of the first part. It thus makes sense that all dimensions are mirrored from left to right. However, specific problems occur in CNNs as here in between the layers fixed pooling layers are installed and their inverse is far from being straight-forward. A convolutional layer may comprise of simple 3x3 filter operations, its inverse filter is in general of double infinite length. As such convolutional layers are typically used in combination with pooling (downsampling), an inverse effect is obtained by filtering with padding (upsampling), called transposed filtering to match the original size of the filtered input. Transposed filtering is thus not a perfect deconvolution but together with the gradient search algorithm it works as such.

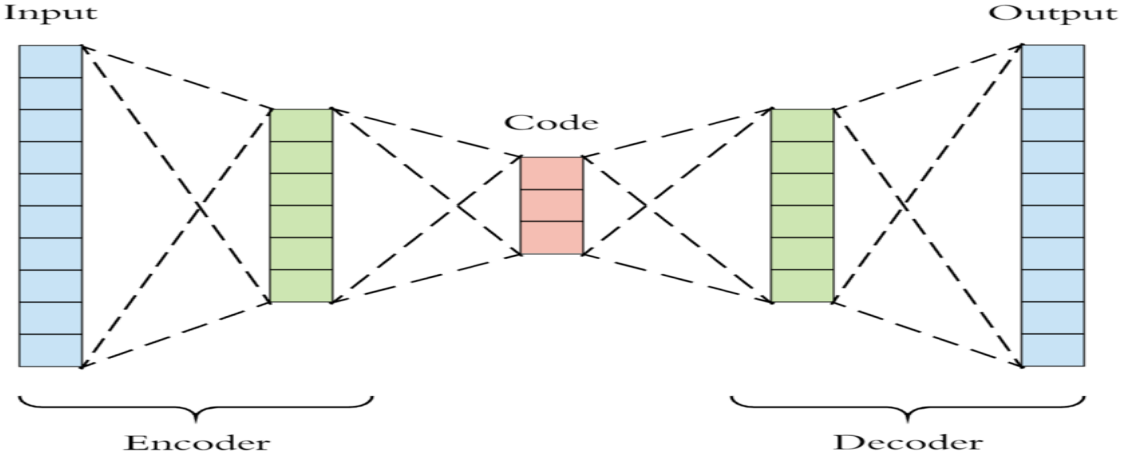

Figure 1.11: *An autoencoding neural netork architecture.*

1.6 Classification Schemes for Adaptive Filters

Adaptive filter algorithms are typically optimizing algorithms. Different to classic optimization problems adaptive filter algorithms try to find optima in a permanently changing environment. The existence of a reference model is of great importance. In a reference model an ideal solution exists that delivers the signal to which the adaptive filter tends to converge. How it is generated plays a crucial role in the structure of the adaptive filter and its properties.

As the formulation of the problems will turn out to be rather complex, a simple solutions such as in LS will not be available. We thus will have to rely on iterative algorithms to find solutions. Those are typically of the form:

$$\underline{\hat{w}}_{k+1} = \underline{\hat{w}}_k + \mu \frac{\partial}{\partial \underline{w}} g(e_k^2),$$

that is a derivative out of a cost function needs to be computed and then the estimated weights that describe the border between the classes are improved. We are thus interested in the conditions that we arrive at the perfect solution. Alternatively to gradient based approaches, there is also so-called *reinforcement learning approaches* which are simpler in nature, utilizing error indicators rather than errors.

Here we distinguish *on- and offline* algorithms. If all required data is available, the algorithm can run a certain number of iterations and then stop resulting in a (sub-) optimal result. This offline mode is often called *iterative algorithm* or *batch processing* as a fixed amount (batch) of data is available. In contrast to this, there is an online mode in which with each iteration new data are presented. This is also-called a *recursive* mode. An adaptive filter is typically in recursive mode as it tries to adapt to an ever changing environment.

Example 1.1 (Iterative Algorithm) *Consider the following update equation, starting with* $w_0 = 0$*;*

$$w_k = w_{k-1} + \frac{1}{k!}a \qquad ; k = 1, 2, ..., N$$

Example 1.2 (Recursive Algorithm) *Consider the following update equation* $w_0 = 0$*;*

$$w_k = w_{k-1} + \frac{1}{k!}a_k \qquad ; k = 1, 2, ...$$

In the second (recursive) algorithm the data a_k are changing while in the first iterative algorithm, a remains unchanged. Question: where does the first algorithm end?[2]

A third difference is given by the choice of *cost functions* (also called *loss functions*). The reference model delivers a desire, the reference signal that is to be approximated by the adaptive filter. The so obtained error signal, that is the difference between reference signal and approximation, can be applied to compute a cost function. Typically, the larger the difference, the larger is the cost function. Depending on which class of signals is considered, we may have expectation values as cost function for stochastic signals or simple norms or metrics for deterministic signals. In one case the term $\mathrm{E}[|e(n)|^p]$ is to minimize while in the other $\sum_{i=0}^{n} |e(n)|^p$ which is similar but not the same. Also minimax formulations are common and will be treated in later chapters.

1.7 Practical Considerations

The following considerations have shown to be of practical relevance although there is no clear theoretical support for them. One open issue is given when the number of training data N_1 for class "A" is distinctly different to N_2 of "not A". A class for which the number of training data is significantly larger than for the other(s), we call the*majority class*. Conversely, if the number is much smaller than for the other(s), we call it the*minority class*. If we use all training data once, we could end up in strong bias towards the majority class. Therefore, we may use a different selection for the two classes which is usually referred to as *re-sampling*, to emphasize the fact that we select data (samples) not only once. If only a fraction of the majority class data is used, we call this an *under-sampling*. This can help to balance the number of training and the two classes. Alternatively, we can reuse the data of the minority class many times to balance the number of training in the majority class. This is called *oversampling*. The later method is related to boot-strapping methods in statistics [89]. If the amount of training data is sufficiently large, a good strategy is it to divide it into two parts of not necessarily equal size and use the first part only for training whereas the second part serves for evaluation of the training quality.

[2]The algorithm converges to $(e-1)a \approx 1.71a$.

	estimated $\hat{y} = 1$	estimated $\hat{y} = 0$
true $y = 1$	TP	TN
true $y = 0$	FN	FP

Table 1.1: Confusion matrix for a binary classifier.

A well established method to measure the quality of a classifier is the so called *confusion matrix* (confusion table) [78]. Table 1.1 depicts such a confusion matrix for a binary classification. The values TP,TN,FN,FP are the numbers of the occurrence of the individual cases. Out of those four values many quality indicators can be derived. The most important ones are

- accuracy: $\frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$
- precision: $\frac{\text{TP}}{\text{TP}+\text{FP}}$
- recall: $\frac{\text{TP}}{\text{TP}+\text{FN}}$
- F1: $\frac{2\text{TP}}{2\text{TP}+\text{FP}+\text{FN}}$

Many more can be found for example under `https://en.wikipedia.org/wiki/Confusion_matrix`.

1.8 Literature

Although there is an abundant amount of literature available, for this lecture the following two books are recommended: [88] provides the essential ideas we will look into, its second edition also obtains many helpful examples for programming. [80] is something like a bible for machine learning and very helpful to read more into various directions.

Chapter 2

Gradient Algorithms in stochastic Settings

In this chapter we present the so-called steepest descent algorithm that is the basis for further gradient type algorithms. Typically such algorithms are applied in the context of system identification, that is by observing input and output of systems, we conclude the input-output relation, thus the impulse response. Applications of this can be found by the millions in electric echo compensation and hands free telephones.

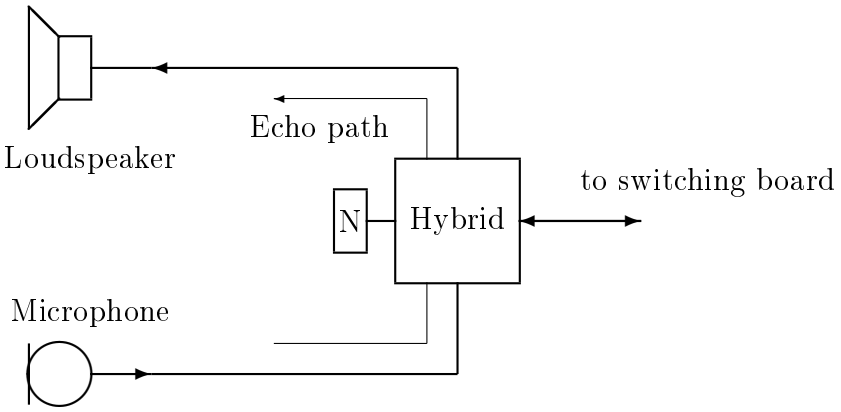

Figure 2.1: *Near echo connection.*

During the past 30 years adaptive filter algorithms have found their way into many electronic products. In most of the cases the user is not aware of their presence which proves their well functioning. Adaptive filters have the property to adapt to a permanently

changing environment and thus to offer optimal performance. In the following we will review various applications of adaptive filters. In Figure 2.1 a schematic for a so-called hybrid is shown (Ger.: Gabelschaltung (Wien-Brücke)) which allows to connect a two-wire line from the switching board to the two wires of a microphone on the one hand and to the two wires of the loudspeaker on the other hand. A perfect balancing is achieved if the user is not hearing himself. Such optimal situation is typically not given as the far-end load (Ger.: Nachbildung N) is in general unknown. Typically a small echo from microphone to loudspeaker is considered as 'natural' as the the user feels that the device is alive. In the context of hands-free telephones (Ger.: Freisprechanlagen) this can cause instabilities and thus needs to be attenuated. An adaptive filter can solve the problem by estimating the impulse response from microphone to loudspeaker, then computes the convolution of input signal of such impulse response, thus replicating the echo and finally subtracting the echo signal.

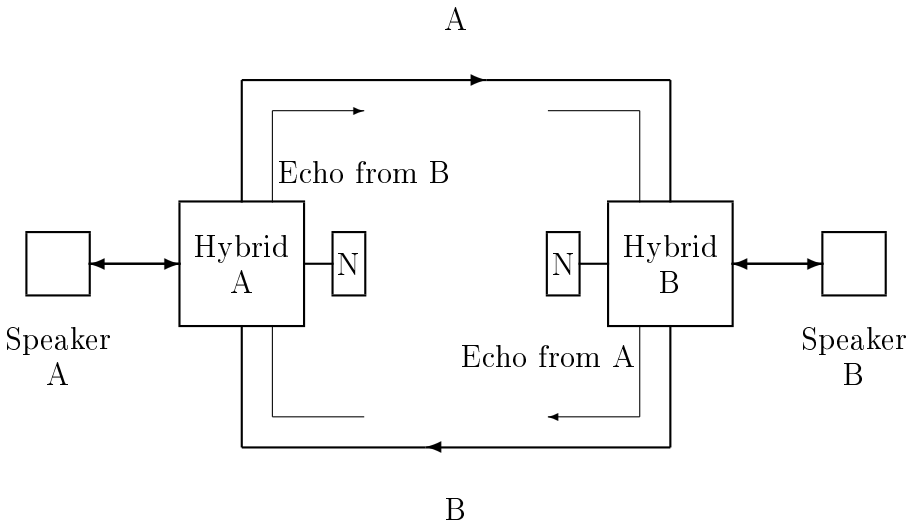

Figure 2.2: *Far echo connection.*

Figure 2.2 exhibits two far-end users schematically, connected via their local hybrids. Everywhere where a switch between two and four wires is required such hybrids are in use. In far-distance connections many of such hybrids may occur. In this case hybrids are not only passing on local signals from microphone to loudspeaker but also far-end signals are returned to their origin. The long delays of 500ms and more in far-distance connections cause disturbances so that the user speaks against himself. If the attenuation of the hybrid is typically 6dB and the echo signal is further amplified

during the far-distance connection, it can happen that the closed loop connection has an attenuation of zero or less and thus the system becomes unstable. Adaptive filters on both ends can compensate for such effects and make the far-distance connection stable. While local echo compensation typically deals with few coefficients of the impulse response (<100), far-end echo compensation may deal with a large number of coefficients (500-4000).

Hands-free telephony (Ger.: Freisprechtelefonanlagen) is by now standard equipment in telephone sets. They are applied in offices as well as in video conferences and even in cars while driving. Figure 2.3 depicts the problem. The far-end speaker signal enters the room via the loudspeaker, is reflected in the room and returns into the microphone signal of the local speaker together with his voice. Both signals are transfered to the far-end speaker where his own signal appears as echo. In case the far-end speaker is also using a hands-free telephone, the loop is closed and a loud sinusoidal sound is audible (Ger.: Rückkopplungspfeifen). An adaptive filter can estimate the loudspeaker-room-impulse response and reconstruct the echo signal in order to subtract it. Important are the very long impulse responses of typically several thousand taps depending on the room size. In this application the local speaker is the disturbance (for the adaptive filter estimation) but as it is the signal of interest to be transmitted, it requires special treatment.

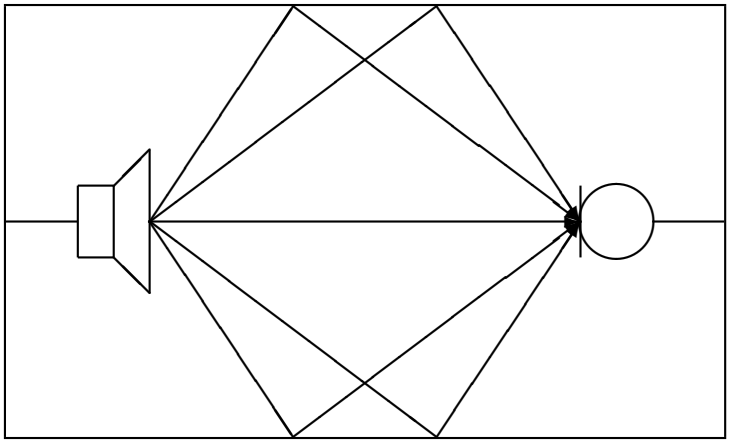

Figure 2.3: *Loudspeaker-Room-Microphone system.*

All applications discussed so far fall into the category of **system identification**. As shown in Figure 2.4 the path of the echo is an unknown linear impulse response. By observing the input and output signal of the system, the adaptive filter learns the impulse response of the system. With the known input signal the filter can reconstruct the echo without the signal of interest and by subtraction obtain a clean, echo-free signal.

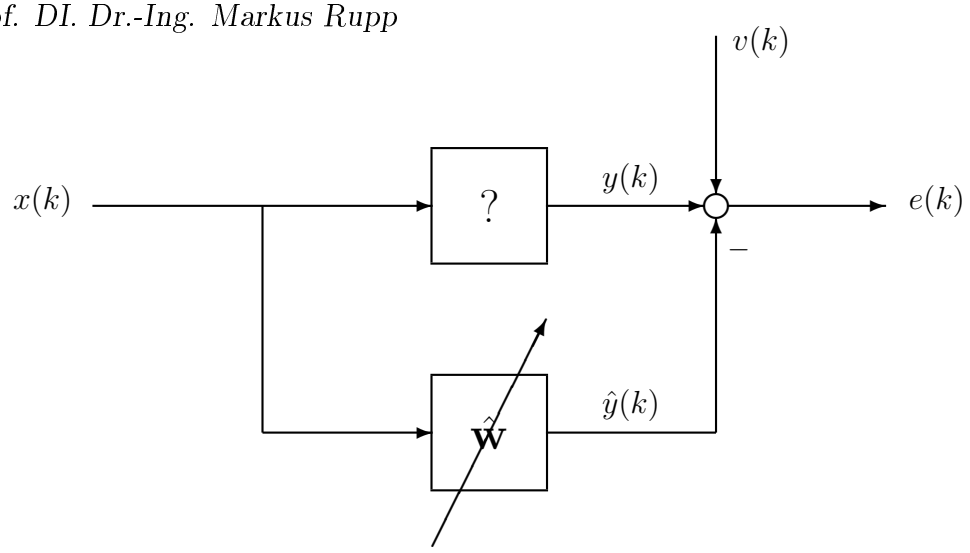

Figure 2.4: *System identification.*

2.1 The various Wiener Solutions

In the literature we find often very different concepts under the names Wiener, Wiener-Hopf, Linear MMSE and Normal equations which may cause some confusion for the reader. Although the common idea is always that we have a linear estimator and the cost function is quadratic, the side constraints may differ tremendously. First of all we have to distinguish between the parameter estimation error and the observation error. Let us reconsider the estimation problem:

$$\mathbf{y} = \underline{w}^T \underline{\mathbf{x}} + \mathbf{v}. \tag{2.1}$$

The term $\underline{w}$ is thus fix while the other terms $\mathbf{y}, \underline{\mathbf{x}}, \mathbf{v}$ are of random nature. Here, $\mathbf{v}$ describes an additive noise, a term that cannot be observed but causes an estimation error, while $\mathbf{y}$ is the term that is observed. For a linear LMSE we assume that the parameter estimation error $\underline{\mathbf{x}} - \hat{\underline{\mathbf{x}}}$ is minimal in the mean square sense. The relation to the observation error $\mathbf{y} - \underline{w}^T \hat{\underline{\mathbf{x}}}$ was never of interest. Alternatively, we could design an estimator so that the observation error becomes minimal in a mean-square sense. In the following Table 2.1 both estimators are listed.

2.2 Complexity of the Exact Wiener Solution

Let us reconsider the following observation equation:

$$\mathbf{d} = \underline{w}^T \underline{\mathbf{x}} + \mathbf{v} = \underline{\mathbf{x}}^T \underline{w} + \mathbf{v}. \tag{2.2}$$

Different to the previous analysis we now assume that the output $\mathbf{d}$ as well as the input $\underline{\mathbf{x}}$ can be observed. Let the autocorrelation matrix $R_{\underline{\mathbf{x}}\underline{\mathbf{x}}} = \mathrm{E}[\underline{\mathbf{x}}\underline{\mathbf{x}}^H] = \left(\mathrm{E}[\underline{\mathbf{x}}^*\underline{\mathbf{x}}^T]\right)^* = \left(\mathrm{E}[\underline{\mathbf{x}}^*\underline{\mathbf{x}}^T]\right)^T$.

Value	Parameter estimation error	Observation error
$\underline{w}$	fix, known	fix, but unknown
$\underline{\mathbf{x}}$	random, $R_{\underline{\mathbf{xx}}}$ known	random, $R_{\underline{\mathbf{xx}}}$ known
$\mathbf{v}$	random, $\sigma^2_{\mathbf{v}}$ known	random, $\sigma^2_{\mathbf{v}}$ known
$\mathbf{y}$	is observed	is observed
is to estimate	$\hat{\underline{\mathbf{x}}} = \mathbf{y}\frac{R_{\underline{\mathbf{xx}}}\underline{w}^*}{\underline{w}^T R_{\underline{\mathbf{xx}}}\underline{w}^* + \sigma^2_{\mathbf{v}}}$	$\hat{\underline{w}} = R^{*(-1)}_{\underline{\mathbf{xx}}}\mathbf{r}^*_{\underline{\mathbf{xy}}}$
observation error	$\sigma^2_{\mathbf{y}}\frac{\sigma^4_{\mathbf{v}}}{[\underline{w}^T R_{\underline{\mathbf{xx}}}\underline{w}^* + \sigma^2_{\mathbf{v}}]^2}$	$\sigma^2_{\mathbf{y}} - \mathbf{r}^H_{\underline{\mathbf{xy}}} R^{*(-1)}_{\underline{\mathbf{xx}}}\mathbf{r}^*_{\underline{\mathbf{xy}}}$

Table 2.1: *Comparison of linear LMS estimator for minimal parameter error and minimal observation error.*

Let the additive noise $\mathbf{v}$ be statistically independent on $\underline{\mathbf{x}}$. The cross correlation $\underline{r}_{\underline{\mathbf{xd}}} = \mathrm{E}[\underline{\mathbf{x}}\mathbf{d}^*]$ delivers:

$$\underline{r}_{\underline{\mathbf{xd}}} = R_{\underline{\mathbf{xx}}}\underline{w}^*.$$

Obviously the unknown weights $\underline{w}$ can be found by minimizing the observation error

$$\min_{\underline{w}} \mathrm{E}\left[\left|\mathbf{d} - \underline{w}^T\underline{\mathbf{x}}\right|^2\right] \tag{2.3}$$

It is thus sufficient to know $R_{\underline{\mathbf{xx}}}$ and $\underline{r}^*_{\underline{\mathbf{xd}}}$. If $R_{\underline{\mathbf{xx}}}$ is regular the solution is uniquely given by:

$$\underline{w}_o = \left(R^*_{\underline{\mathbf{xx}}}\right)^{-1}\underline{r}^*_{\underline{\mathbf{xd}}}.$$

This has already been shown as linear LMS or Wiener solution. The required matrix inversion can be problematic and challenging as it can lead to numerical problems in case the matrix in not well conditioned (this is a common problem for speech signals) as well as a very high complexity. Matrix inversions require in general a complexity order of $O(M^3)$, given a matrix of dimension $M \times M$.

A considerable reduction can be obtained by so-called order recursive structures. They are known under the names Durbin, Levinson and Trench.

2.3 The Steepest Descent Algorithm

An alternative method to find a set of weights that minimizes the observation error that works completely without inverting the matrix is the so-called method of **steepest descent**. Here, iteratively, the estimated weights $\hat{\underline{w}}$ are improved. The algorithm works as follows:

$$\hat{\underline{w}}_k = \hat{\underline{w}}_{k-1} + \mu(k)\underline{z}_k;\ k = 1, 2, ... \tag{2.4}$$

We indicate $\underline{\hat{w}}_k$ that it is an estimate of $\underline{w}$. The iteration starts with an initial value $\underline{\hat{w}}_0$ out of which the improved value $\underline{\hat{w}}_1$ comes out after the first iteration step. The iteration step is indicated by the index k (not a time index!). The improvement in direction $\underline{z}_k$ is controlled by a so-called step-size $\mu(k) > 0$. The question we have to tackle is which direction $\underline{z}_k$ will lead to the desired solution and which is the best step-size.

Let us consider one more time the observation equation (2.2). Let us assume that only $\mathbf{d}$ is given and we like to estimate it by a linear combination of $\underline{\mathbf{x}}$ and $\underline{w}$. If $\underline{\mathbf{x}}, \mathbf{d}$ is given we can write:

$$g_o = \min_{\underline{\hat{w}}} \mathrm{E}\left[\left|\mathbf{d} - \underline{\mathbf{x}}^T \underline{\hat{w}}\right|^2\right]. \tag{2.5}$$

As $\underline{r}_{\underline{\mathbf{x}}\mathbf{d}} = \underline{r}^*_{\mathbf{d}\underline{\mathbf{x}}}$ we can further write

$$g(\underline{\hat{w}}) = \mathrm{E}\left[\left|\mathbf{d} - \underline{\mathbf{x}}^T \underline{\hat{w}}\right|^2\right] = \sigma_d^2 - \underline{\hat{w}}^T \underline{r}_{\underline{\mathbf{x}}\mathbf{d}} - \underline{r}^T_{\mathbf{d}\underline{\mathbf{x}}} \underline{\hat{w}}^* + \underline{\hat{w}}^T R_{\underline{\mathbf{x}}\underline{\mathbf{x}}} \underline{\hat{w}}^* \tag{2.6}$$

as cost function that is to minimize with respect to $\underline{\hat{w}}$. As the function is quadratic in $\underline{\hat{w}}$ (a paraboloid), there is a unique minimum at the Wiener solution $\underline{\hat{w}} = \underline{w}_o$, which we obtain out of the well-known orthogonality relation (**??**):

$$\mathrm{E}[(\mathbf{d} - \underline{w}_o^T \underline{\mathbf{x}})\underline{\mathbf{x}}^H] = \underline{\mathbf{0}}^H,$$

to

$$g_o = \sigma_d^2 - \underline{r}^H_{\underline{\mathbf{x}}\mathbf{d}} R^{-1}_{\underline{\mathbf{x}}\underline{\mathbf{x}}} \underline{r}_{\underline{\mathbf{x}}\mathbf{d}}. \tag{2.7}$$

By substituting of (2.7) in (2.6) we obtain the following description:

$$g(\underline{\hat{w}}) = g_o + (\underline{\hat{w}} - \underline{w}_o)^T R_{\underline{\mathbf{x}}\underline{\mathbf{x}}} (\underline{\hat{w}} - \underline{w}_o)^*. \tag{2.8}$$

Obviously the function $g(\underline{\hat{w}})$ is quadratic. We can thus develop a Taylor series around the point $\underline{\hat{w}}_{k-1}$:

$$g(\underline{\hat{w}}) = g(\underline{\hat{w}}_{k-1}) + \nabla g(\underline{\hat{w}}_{k-1}) \left(\underline{\hat{w}} - \underline{\hat{w}}_{k-1}\right) + \left(\underline{\hat{w}} - \underline{\hat{w}}_{k-1}\right)^H \nabla^2 g(\underline{\hat{w}}_{k-1}) \left(\underline{\hat{w}} - \underline{\hat{w}}_{k-1}\right). \tag{2.9}$$

The derivatives are obtained as follows. The gradient is obtained by differentiating the cost function with respect to $\underline{\hat{w}}$:

$$\nabla g(\underline{\hat{w}}_{k-1}) = \left.\frac{\partial g(\underline{\hat{w}})}{\partial \underline{\hat{w}}}\right|_{\underline{\hat{w}}=\underline{\hat{w}}_{k-1}} = [\underline{\hat{w}} - \underline{w}_o]^H R^*_{\underline{\mathbf{x}}\underline{\mathbf{x}}}. \tag{2.10}$$

Note that the gradient is a row vector. The relation follows immediately from (2.8). The second term in (2.10) is already known from the orthogonality relation as the cross correlation $\underline{r}_{\underline{\mathbf{x}}\mathbf{d}}$ between $\underline{\mathbf{x}}$ and $\mathbf{d}$, thus $\underline{w}_o^H R^*_{\underline{\mathbf{x}}\underline{\mathbf{x}}} = \underline{r}^T_{\underline{\mathbf{x}}\mathbf{d}}$ or equivalently $R_{\underline{\mathbf{x}}\underline{\mathbf{x}}} \underline{w}_o = \underline{r}^*_{\underline{\mathbf{x}}\mathbf{d}}$. We thus obtain:

$$\frac{\partial g}{\partial \underline{\hat{w}}} = \underline{\hat{w}}^H R^*_{\underline{\mathbf{x}}\underline{\mathbf{x}}} - \underline{r}^T_{\underline{\mathbf{x}}\mathbf{d}}. \tag{2.11}$$

The second derivative is obtained by differentiating the gradient:

$$\nabla^2 g(\underline{\hat{w}}_{k-1}) = R^*_{\underline{\mathbf{xx}}}. \tag{2.12}$$

The function $g(\underline{\hat{w}})$ is sufficiently smooth so that we can associate each point with a gradient that points in direction of the steepest ascent. It will thus point away from the minimum. Its negative (and conjugate complex) value on the other hand will point in direction of the steepest descent and thus to the minimum (at least roughly). We thus take the negative (conjugate complex, transpose) gradient as the direction $\underline{z}_k = (\underline{r}^*_{\underline{\mathbf{xd}}} - R^*_{\underline{\mathbf{xx}}}\underline{\hat{w}}_{k-1})$.

The cost (2.8) for only two parameters $\hat{w}_1$ and $\hat{w}_2$ ($\underline{\hat{w}}^T = [\hat{w}_1, \hat{w}_2]$) is shown in Figure 2.5. In the right contour lines we can see negative gradients on various points. We observe that points far away from the minimum do not have a gradient that directly points to the minimum but vaguely into its direction. Only by step-wise iteration we can ensure to approach the minimum.

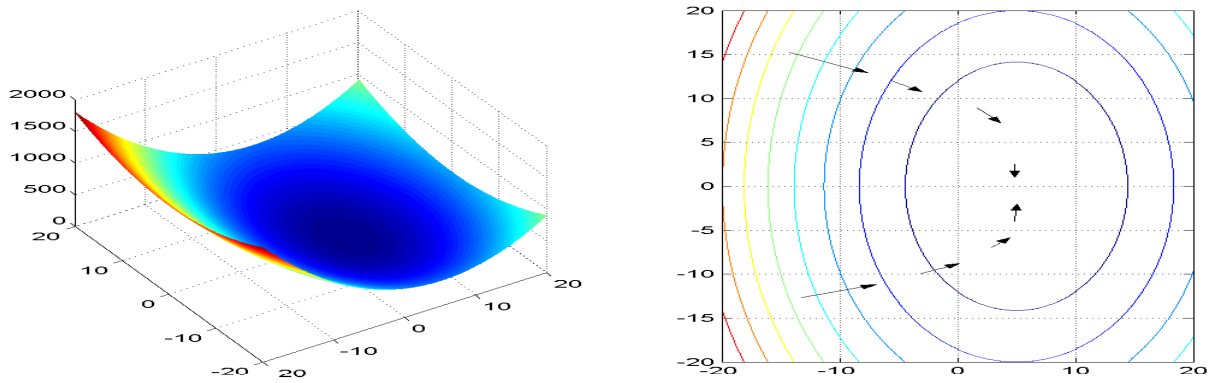

Figure 2.5: *Cost function: left as paraboloid, right as contour plot.*

If we substitute the iterative approach (2.4) into the cost function (2.9) we obtain:

$$g(\underline{\hat{w}}_k) = g(\underline{\hat{w}}_{k-1}) + \nabla g(\underline{\hat{w}}_{k-1})\mu(k)\underline{z}_k + \mu^2(k)\underline{z}_k^H \nabla^2 g(\underline{\hat{w}}_{k-1})\underline{z}_k, \tag{2.13}$$

where we denoted the development point of the Taylor series as $\underline{\hat{w}} = \underline{\hat{w}}_k$. If we require the cost function to decrease

$$g(\underline{\hat{w}}_k) < g(\underline{\hat{w}}_{k-1}) \tag{2.14}$$

then we have to require

$$\mu(k)\nabla g(\underline{\hat{w}}_{k-1})\underline{z}_k < 0. \tag{2.15}$$

Such condition is satisfiable for many directions, in particular for directions of the form:

$$\underline{z}_k = -B\nabla^H g(\underline{\hat{w}}_{k-1}) \tag{2.16}$$

for positive definite matrices B.

Let $B = I$ and we obtain the standard steepest-descent algorithm:

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu(k)[\underline{r}^*_{\mathbf{xd}} - R^*_{\underline{\mathbf{xx}}}\underline{\hat{w}}_{k-1}], \quad k = 1, 2, ... \tag{2.17}$$

This iterative equation has the typical form:

$$\text{New estimate} = \text{old estimate} + \text{correction},$$

a formulation that is present in all adaptive filter algorithms. The choice of the correction term results in conditions for convergence but also speed of adaptation and precision of its steady-state.

Consider the difference

$$\underline{\tilde{w}}_k \stackrel{\Delta}{=} \underline{w}_o - \underline{\hat{w}}_k \tag{2.18}$$

between current estimate $\underline{\hat{w}}_k$ and Wiener solution $\underline{w}_o$, we can formulate the update equation (2.17) in this new variable

$$\begin{aligned} \underline{\tilde{w}}_k &= \underline{\tilde{w}}_{k-1} - \mu(k)R^*_{\underline{\mathbf{xx}}}\underline{\tilde{w}}_{k-1} && (2.19)\\ &= \left(I - \mu(k)R^*_{\underline{\mathbf{xx}}}\right)\underline{\tilde{w}}_{k-1}, \quad k = 1, 2, ... && (2.20) \end{aligned}$$

The difference (2.18) is called *parameter estimation error vector* or error vector. Obviously, Equation (2.20) is a homogeneous difference equation. It can be diagonalized $QR^*_{\underline{\mathbf{xx}}}Q^H = \Lambda$ and with the transformation $\underline{\tilde{u}} = Q\underline{\tilde{w}}$ we obtain:

$$\begin{aligned} \underline{\tilde{u}}_k &= (I - \mu(k)\Lambda)\underline{\tilde{u}}_{k-1}, && (2.21)\\ \tilde{u}_i(k) &= (1 - \mu(k)\lambda_i)\tilde{u}_i(k-1). && (2.22) \end{aligned}$$

The index i denotes here the $i-$th entry of the vector $\underline{u}$ and λ_i are the diagonal terms of the diagonal matrix Λ. From the diagonalized form (2.22) we can read immediately the convergence conditions of the iterative algorithm:

$$|1 - \mu(k)\lambda_i| < 1. \tag{2.23}$$

Equivalently we can formulate this condition in terms of the step-size $\mu(k)$

$$0 < \mu(k) < \frac{2}{\lambda_{\max}} \leq \frac{2}{\lambda_i}. \tag{2.24}$$

We thus conclude that the step-size needs to be positive and is bounded from above by the reciprocal largest eigenvalue of $R_{\underline{\mathbf{xx}}}$. Such a condition may not be very practical as we first have to undertake an eigenvalue analysis of $R_{\underline{\mathbf{xx}}}$. Note however that

trace($R_{\underline{\mathbf{xx}}}$) =trace($\Lambda$) = $\sum \lambda_i > \lambda_{\max}$. We thus have the possibility to upper bound the step-size without finding the explicit values of the eigenvalues.

If this condition for the step-size is satisfied, then the solution of the homogeneous difference equation system can be written in terms of exponential decaying terms $\prod_{l=1}^{k}(1-\mu(l)\lambda_i)$. We can also find the **adaptation rate**, the speed with which the terms decay and thus the filter learns its correct value. The smaller the term $(1-\mu(k)\lambda_i)$, the faster runs the error value towards zero. The choice $\mu(k) = 1/\lambda_i$ may be optimal but it is dependent for each eigenvalue.

The considerations in this section was focused on quadratic cost functions. This however does not mean that the method of steepest descent can only be applied for such cost functions. If an arbitrary cost function is given, we can always derive a Taylor series as in (2.9). We then obtain:

$$g(\underline{\hat{w}}) = g(\underline{\hat{w}}_{k-1}) + \nabla g(\underline{\hat{w}}_{k-1})\left(\underline{\hat{w}} - \underline{\hat{w}}_{k-1}\right) + \left(\underline{\hat{w}} - \underline{\hat{w}}_{k-1}\right)^H \nabla^2 g(\underline{\hat{w}}_{k-1})\left(\underline{\hat{w}} - \underline{\hat{w}}_{k-1}\right) + ..., \quad (2.25)$$

thus further terms that are non-linear and non-quadratic. Here, the condition (2.15) is not suitable any longer to find the global minimum. The quadratic terms on the other hand provide hints that there exist one or more points around which approximate a (local) minimum by parabolas. If the steepest descent method is applied we will find such local minimum. However it may not be the desired global minimum. Only for a quadratic cost function we can ensure this.

Newton Method If we alternatively select $B = R_{\underline{\mathbf{xx}}}^{*-1}$ we obtain the so-called Newton-Iteration

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu_k R_{\underline{\mathbf{xx}}}^{*-1}[\underline{r}_{\underline{\mathbf{xd}}}^{*} - R_{\underline{\mathbf{xx}}}^{*}\underline{\hat{w}}_{k-1}], \quad k = 1, 2, ... \quad (2.26)$$

Applying the same procedure as before, we now obtain the much simpler form:

$$\underline{\tilde{w}}_k = (I - \mu_k)\ \underline{\tilde{w}}_{k-1}, \quad k = 1, 2, ... \quad (2.27)$$

from which we can immediately derive convergence as long as $0 < \mu_k < 2$ and fastest learning for $\mu_k = 1$.

Non-linear function in cost function In neural networks we typically apply a non-linear functions, e.g., a sigmoid function as activation function. The cost function can then read

$$g(\underline{w}) = \frac{1}{1 + e^{-\alpha \underline{x}^h \underline{w}}} \quad (2.28)$$

with some positive value α. Examples are shown in Figure 2.6. The derivative of the cost function turns out to be simply

$$\frac{\partial}{\partial \underline{w}} g(\underline{w}) = \alpha g(\underline{w})(1 - g(\underline{w})).$$

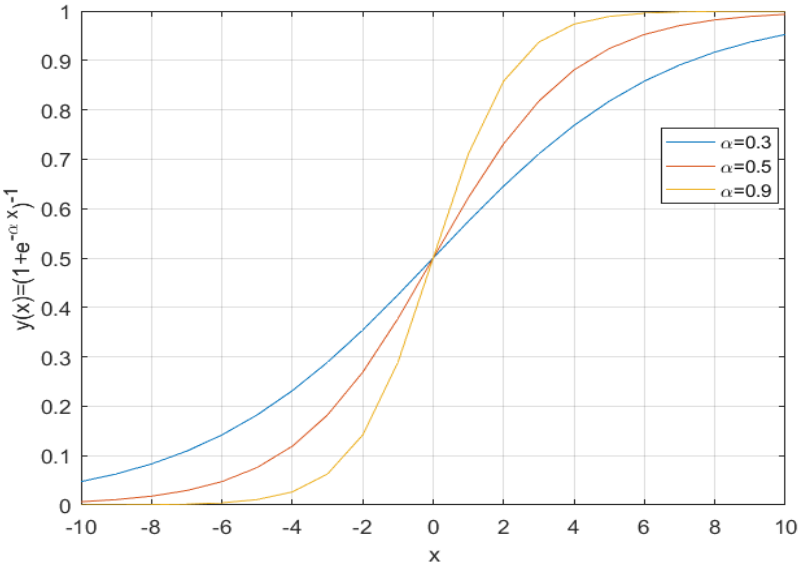

Figure 2.6: *Typical sigmoid functions with various values of* α.

A function that is twice differentiable with

$$\frac{\partial^2}{\partial \underline{w}^2} g(\underline{w}) \geq 0$$

is called convex. Equivalently, the Hessian $\nabla^2 g(\underline{w}) \geq 0$ has non negative eigenvalues. For convex functions an iterative algorithm can guarantee convergence. Some typically known convex functions are $g(x) = x^2, g(x) = e^x, g(x) = -U(x)\log(x)$. Note that the sigmoid function is not convex.

Exercise 2.1 *Assume the eigenvalues* λ_i *of the autocorrelation matrix* $\mathbf{R}_{\underline{\mathbf{xx}}}$ *as given. Compute the constant step-size* μ *such that:*

$$\min_{\mu} \max_{\lambda_i} |1 - \mu\lambda_i| .$$

Hint: Consider the two extrema $1 - \mu\lambda_{\min}$ *and* $1 - \mu\lambda_{\max}$.

Exercise 2.2 *Consider Equation (2.16) and select* $B = \left(R^*_{\underline{\mathbf{xx}}}\right)^{-1}$. *What can we say about the so obtained iteration of Newton-type? For which step-sizes* μ *do we obtain convergence?*

Exercise 2.3 *Derive the steepest-descent algorithm (2.8)-(2.17) for real-valued signals. What will change?*

Exercise 2.4 *Consider the infinite series $(I - \mu R)^k$ for positive definite matrix R. Under which conditions does $\sum_{k=0}^{\infty}(I - \mu R)^k$ converge? Which value does the sum take on in the limit?*

Exercise 2.5 *Show that the optimal step-size is given when the quadratic cost function (2.9) is decreasing maximally fast and thus given by:*

$$\mu_{opt}(k) = \frac{\|\nabla g(\underline{\hat{w}}_{k-1})\|^2}{\nabla g(\underline{\hat{w}}_{k-1}) R_{\underline{\mathbf{xx}}} \nabla^H g(\underline{\hat{w}}_{k-1})}.$$

Matlab Experiment 2.1 For a linear system of length $M = 10$ with white (alternatively also with colored) excitation signal , compute the Wiener solution and display graphically the cost function $g(\underline{\hat{w}}_{k-1})$ for the first 100 values over k. Use a fixed step-size and experiment with an optimal time-variant step-size.

2.4 Literature

A good overview on estimation methods is in [38]. An introduction to the Steepest Descent Algorithm is in [33] and the newer edition [34] and [75].

Chapter 3

The LMS Algorithm

The LMS algorithm is the by far most frequently applied adaptive algorithm. Its advantages are its numerical stability, low complexity and also its robustness. Almost all practically employed adaptive algorithms are LMS algorithms or derivatives thereof. In this chapter we introduce the LMS algorithm starting with its classic interpretation as approximation to the steepest descent algorithm. Its most important properties such as convergence speed and condition will be derived based on stochastic approaches, that is the driving signals are considered random processes.

3.1 Classic Approach: Approximating the Wiener Solution

Let us reconsider the update equation (2.17) of the steepest-descent algorithm with constant step-size μ:

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu(\underline{r}^*_{\underline{\mathbf{x}}\mathbf{d}} - R^*_{\underline{\mathbf{x}}\underline{\mathbf{x}}}\underline{\hat{w}}_{k-1}), \quad k = 1, 2, ...$$

We require a-priori knowledge of the cross correlation $\underline{r}_{\underline{\mathbf{x}}\mathbf{d}}$ as well as the autocorrelation $R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}$ from the driving input process. If these are unknown we can use estimates instead. Starting with the input signal vector

$$\underline{\mathbf{x}}^T_k \triangleq [\mathbf{x}(k), \mathbf{x}(k-1), ..., \mathbf{x}(k-M+1)], \tag{3.1}$$

then the immediate (instantaneous) estimates are:

$$\widehat{R}^*_{\underline{\mathbf{x}}\underline{\mathbf{x}}} = \underline{\mathbf{x}}^*_k \underline{\mathbf{x}}^T_k \tag{3.2}$$

$$\widehat{\underline{r}}_{\underline{\mathbf{x}}\mathbf{d}} = \underline{\mathbf{x}}_k \mathbf{d}^*(k). \tag{3.3}$$

Substituting these estimates we immediately obtain the **LMS algorithm**:

$$\underline{\hat{\mathbf{w}}}_k = \underline{\hat{\mathbf{w}}}_{k-1} + \mu \underline{\mathbf{x}}^*_k \underbrace{(\mathbf{d}(k) - \underline{\mathbf{x}}^T_k \underline{\hat{\mathbf{w}}}_{k-1})}_{\tilde{\mathbf{e}}_a(k)}. \tag{3.4}$$

The name Least-Mean-Square (LMS) is somewhat misleading. Correctly it is a stochastic gradient method as the gradient is not a fixed value given by the present statistic as in case of the steepest descent algorithm but is varying with the instantaneous values of the input signal $\underline{\mathbf{x}}_k$. The name LMS has been used extensively in literature and will thus be used in this text as well. Note however that the LMS estimator is an estimator with a nonlinear function as was shown in the previous chapter. As we will see later in the chapter on robustness, the name stochastic gradient method is also not entirely correct as the algorithm can work perfectly well without random signals.

Note further that the estimates are also random values such as $\mathbf{d}(k)$ and $\underline{\mathbf{x}}_k$. The error term $(\mathbf{d}(k) - \underline{\mathbf{x}}_k^T \underline{\hat{\mathbf{w}}}_{k-1})$ is called *disturbed a-priori error*, as it is constructed by a-priori estimates $\underline{\hat{\mathbf{w}}}_{k-1}$. Analogue to this there is also a *disturbed a-posteriori error* constructed by the a-posteriori estimates $\underline{\hat{\mathbf{w}}}_k$: $\tilde{\mathbf{e}}_p = \mathbf{d}(k) - \underline{\mathbf{x}}_k^T \underline{\hat{\mathbf{w}}}_k$.

We obtain first variants of this algorithm by selecting different step-sizes. Time variant step-sizes $\mu(k)$ appear practical in particular when coupled to the energy of the input process. Following algorithms are common:

- general time variant step-size $\mu(k)$: stochastic gradient type algorithm.
- $\mu(k) = \alpha / \|\underline{\mathbf{x}}_k\|^2$: normalized LMS or NLMS algorithm.
- $\mu(k) = \alpha / [\epsilon + \|\underline{\mathbf{x}}_k\|^2]$ with $\epsilon > 0$, ϵ-NLMS algorithm[5].
- $\mu(k) = \alpha / [1 + \alpha \|\underline{\mathbf{x}}_k\|^2]$ a-posteriori form of the LMS algorithm.

Next to its low complexity and simplicity there is another point that makes the LMS algorithm so popular: new adaptive algorithms are readily derived. All we need is an error term that is converted into a cost function, for example:

$$\min \mathrm{E}\left[f[\tilde{\mathbf{e}}_a(k)]\right].$$

Differentiate with respect to the parameters $\underline{\hat{w}}_{k-1}$ and write down a gradient methods following the idea: *New estimate is old estimate plus negative gradient.* In most cases this method is successful. Correctly, the algorithm needs to be analyzed first. With statistical methods this is often not feasible. Finally we will show a typical example. We minimize $\mathrm{E}[|\tilde{\mathbf{e}}_a(k)|^K]$. Differentiating with respect to $\underline{\hat{w}}_{k-1}$ leads to the gradient: $\mathrm{E}[-\frac{K}{2}|\tilde{e}_a(k)|^{K-2}\underline{x}_k \tilde{e}_a(k)]$. The expectation is substituted by its instantaneous values and we obtain:

The Least-Mean-K Algorithm:

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu(k)|\tilde{e}_a(k)|^{K-2}\underline{x}_k^* \tilde{e}_a(k). \tag{3.5}$$

Exercise 3.1 *Derive an adaptive algorithm to minimize the cost function* $E[|\tilde{e}_a(k)|]$ *and distinguish here complex-valued as well as real-valued signals.*

Exercise 3.2 *Consider an undisturbed, nonlinear system:* $\mathbf{y}(k) = \underline{\mathbf{x}}_k^T \underline{w}_1 + \underline{\mathbf{xx}}_k^T \underline{w}_2$ *with* $\mathbf{xx}(k-i) = \mathbf{x}(k)\mathbf{x}(k-i), i = 0, 1, ..M_2 - 1$. *The parameter vectors* $\underline{w}_1$ *and* $\underline{w}_2$ *have the dimensions* $M_1 \times 1$ *and* $M_2 \times 1$. *Derive an adaptive algorithm to minimize the additively disturbed squared error signal. What is the autocorrelation matrix of the input process if* $\mathbf{x}(k)$ *is a white Gaussian process?*

3.2 Stationary Behavior

The learning properties such as learning rate in a stationary environment that is a constant impulse response given by the Wiener solution $\underline{w}_o$ can be computed analytically under certain assumptions. Such assumptions are strongly simplifying the situation and are only correct in case of a linear combiners. We require for this derivation the knowledge of statistical convergence as well as some properties of Gaussian and spherically invariant processes as pre-requisite. More details can be found in the Appendices B, C and D. For the more interested reader, there is an even more general derivation with much looser conditions in Appendix **??**.

3.2.1 Assumptions

Assumptions: Independence Assumption (Ger.: Unabhängigkeitsannahme)

- The observed desired signal $\mathbf{d}(k)$ originates from a reference model $\mathbf{d}(k) = \underline{w}_o^T \underline{\mathbf{x}}_k + \mathbf{v}(k)$, with zero-mean processes $x(k)$ and $v(k)$.
- The vectors $\underline{\mathbf{x}}_k$ of the input process are statistically independent to each other that is $f_{\underline{\mathbf{xx}}}(\underline{x}_k, \underline{x}_l) = f_{\underline{\mathbf{x}}}(\underline{x}_k) f_{\underline{\mathbf{x}}}(\underline{x}_l)$ for $k \neq l$.
- The driving input process $\underline{\mathbf{x}}_k$ is of zero-mean and circular (spherically invariant[8]) Gaussian distributed.
- The additive noise $\mathbf{v}(k)$ is statistically independent of the input process $\underline{\mathbf{x}}_k$.

Note that by such conditions the vectors $\underline{\hat{\mathbf{w}}}_k$ are statistically independent of $\underline{\mathbf{x}}_l$, $l > k$.

3.2.2 The Mean Error Vector

We consider the parameter error vector (also weight error vector, tap error vector)

$$\underline{\tilde{\mathbf{w}}}_k \stackrel{\Delta}{=} \underline{w}_o - \underline{\hat{\mathbf{w}}}_k \tag{3.6}$$

in the mean, that is

$$\begin{aligned} \underline{\tilde{\mathbf{w}}}_k &= (I - \mu \underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T)\underline{\tilde{\mathbf{w}}}_{k-1} - \mu \underline{\mathbf{x}}_k^* v(k);\ k = 1, 2, ... && (3.7) \\ \mathrm{E}[\underline{\tilde{\mathbf{w}}}_k] &= \mathrm{E}[(I - \mu \underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T)\underline{\tilde{\mathbf{w}}}_{k-1}];\ k = 1, 2, ... && (3.8) \end{aligned}$$

As the vectors $\underline{\hat{\mathbf{w}}}_{k-1}$ are statistically independent of $\underline{\mathbf{x}}_k$, we have:

$$\mathrm{E}[\underline{\tilde{\mathbf{w}}}_k] = \mathrm{E}[I - \mu \underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T]\mathrm{E}[\underline{\tilde{\mathbf{w}}}_{k-1}] = (I - \mu R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}^*)\mathrm{E}[\underline{\tilde{\mathbf{w}}}_{k-1}]. \tag{3.9}$$

From this equation we recognize that the error vectors **in the mean** behave exactly as the error vectors of the steepest descent algorithm. We thus know the condition for convergence in the mean:

$$0 < \mu < \frac{2}{\lambda_{\max}}. \tag{3.10}$$

With the results of Exercise 2.1 we recognize that the eigenvalue ratio is responsible for the maximum convergence speed in the mean. We further can interpret Equation (3.9), that we have an asymptotically bias-free estimator.

3.2.3 The Mean Square Error Vector

The just found conditions in the mean are relatively weak conditions. A much stronger condition is given if we consider the error vector in the mean square sense. For this we consider the error vector covariance matrix P_k at time instant k:

$$P_k = \mathrm{E}[(\underline{w}_o - \underline{\hat{\mathbf{w}}}_k)(\underline{w}_o - \underline{\hat{\mathbf{w}}}_k)^H] = \mathrm{E}[\underline{\tilde{\mathbf{w}}}_k \underline{\tilde{\mathbf{w}}}_k^H]. \tag{3.11}$$

Substituting the LMS equation (3.4) into the definition above we obtain a recursive equation for the error vector covariance matrix:

$$\begin{aligned} P_k &= \mathrm{E}[(I - \mu \underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T)P_{k-1}(I - \mu \underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T)] + \mu^2 \mathrm{E}[\underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T |\mathbf{v}(k)|^2] && (3.12) \\ &= P_{k-1} - \mu \mathrm{E}[P_{k-1} \underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T] - \mu \mathrm{E}[\underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T P_{k-1}] + \mu^2 \mathrm{E}[\underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T P_{k-1} \underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T] \\ &\quad + \mu^2 \mathrm{E}[\underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T |\mathbf{v}(k)|^2]. && (3.13) \end{aligned}$$

Note that due to the independence assumption we can write $\mathrm{E}[P_{k-1} \underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T] = P_{k-1} R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}^*$ and $\mathrm{E}[\underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T |\mathbf{v}(k)|^2] = R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}^* \sigma_{\mathbf{v}}^2$. Equation (3.13) can this be reformulated to:

$$P_k = P_{k-1} - \mu P_{k-1} R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}^* - \mu R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}^* P_{k-1} + \mu^2 \mathrm{E}[\underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T P_{k-1} \underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T] + \mu^2 R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}^* \sigma_{\mathbf{v}}^2. \tag{3.14}$$

Furthermore we have for complex-valued spherically invariant Gaussian processes (see Appendix D):

$$\begin{aligned}\mathrm{E}[\underline{\mathbf{x}}_k^*\underline{\mathbf{x}}_k^T P_{k-1}\underline{\mathbf{x}}_k^*\underline{\mathbf{x}}_k^T] &= \mathrm{E}[\underline{\mathbf{x}}_k^*\underline{\mathbf{x}}_k^T]P_{k-1}\mathrm{E}[\underline{\mathbf{x}}_k^*\underline{\mathbf{x}}_k^T] + \mathrm{trace}[P_{k-1}\mathrm{E}[\underline{\mathbf{x}}_k^*\underline{\mathbf{x}}_k^T]]\mathrm{E}[\underline{\mathbf{x}}_k^*\underline{\mathbf{x}}_k^T] \\ &= R_{\underline{\mathbf{xx}}}^* P_{k-1} R_{\underline{\mathbf{xx}}}^* + \mathrm{trace}[P_{k-1}R_{\underline{\mathbf{xx}}}^*]R_{\underline{\mathbf{xx}}}^*.\end{aligned}$$

Hint 1: for real-valued spherically invariant Gaussian processes we have:

$$\begin{aligned}\mathrm{E}[\underline{\mathbf{x}}_k\underline{\mathbf{x}}_k^T P_{k-1}\underline{\mathbf{x}}_k\underline{\mathbf{x}}_k^T] &= 2\mathrm{E}[\underline{\mathbf{x}}_k\underline{\mathbf{x}}_k^T]P_{k-1}\mathrm{E}[\underline{\mathbf{x}}_k\underline{\mathbf{x}}_k^T] + \mathrm{trace}[P_{k-1}\mathrm{E}[\underline{\mathbf{x}}_k\underline{\mathbf{x}}_k^T]]\mathrm{E}[\underline{\mathbf{x}}_k\underline{\mathbf{x}}_k^T] \\ &= 2R_{\underline{\mathbf{xx}}} P_{k-1} R_{\underline{\mathbf{xx}}} + \mathrm{trace}[P_{k-1}R_{\underline{\mathbf{xx}}}]R_{\underline{\mathbf{xx}}}.\end{aligned}$$

Hint 2: The same statements are also true for the larger class of spherically invariant complex-valued processes.

We can thus reformulate Equation (3.14) into:

$$P_k = P_{k-1} - \mu P_{k-1}R_{\underline{\mathbf{xx}}}^* - \mu R_{\underline{\mathbf{xx}}}^* P_{k-1} + \mu^2(2R_{\underline{\mathbf{xx}}}^* P_{k-1} R_{\underline{\mathbf{xx}}}^* + \mathrm{trace}[P_{k-1}R_{\underline{\mathbf{xx}}}^*]R_{\underline{\mathbf{xx}}}^*) + \mu^2 R_{\underline{\mathbf{xx}}}^*\sigma_{\mathbf{v}}^2. \tag{3.15}$$

Diagonalizing the acf matrix $Q^H R_{\underline{\mathbf{xx}}}^* Q = \Lambda$, and we obtain:

$$\begin{aligned}Q^H P_k Q &= Q^H P_{k-1}Q - \mu Q^H P_{k-1}Q\Lambda - \mu\Lambda Q^H P_{k-1}Q \\ &\quad +\mu^2(2\Lambda Q^H P_{k-1}Q\Lambda + \mathrm{trace}[Q^H P_{k-1}Q\Lambda]\Lambda) + \mu^2\Lambda\sigma_{\mathbf{v}}^2.\end{aligned} \tag{3.16}$$

Unfortunately, we cannot expect that the same diagonalization of $R_{\underline{\mathbf{xx}}}$ also diagonalizes the error vector covariance matrices. We can only expect that the unitary transformation changes the matrix into $Q^H P_k Q = C_k$:

$$C_k = C_{k-1} - \mu C_{k-1}\Lambda - \mu\Lambda C_{k-1} + \mu^2(2\Lambda C_{k-1}\Lambda + \mathrm{trace}[C_{k-1}\Lambda]\Lambda) + \mu^2\Lambda\sigma_{\mathbf{v}}^2. \tag{3.17}$$

Since $\mathrm{trace}[C_{k-1}\Lambda]$ affects only the elements on the main diagonal, we can further focus on the main diagonal elements of C_k and neglect the others. Concentrating all diagonal elements of C_k into a vector $\underline{c}_k$ we obtain

$$\underline{c}_k = B\underline{c}_{k-1} + \mu^2\underline{\lambda}\sigma_{\mathbf{v}}^2. \tag{3.18}$$

Here, we introduced two new terms: the vector $\underline{\lambda}$ contains the eigenvalues of the acf matrix $R_{\underline{\mathbf{xx}}}$ and the matrix B with the following entries:

$$B = I - 2\mu\Lambda + \mu^2(2\Lambda^2 + \underline{\lambda}\underline{\lambda}^T) \tag{3.19}$$

$$= \begin{cases} 1 - 2\mu\lambda_i + 3\mu^2\lambda_i^2 & \text{main diagonal} \\ \mu^2\lambda_i\lambda_j & \text{else} \end{cases} \tag{3.20}$$

Knowing the matrix B we find sufficient conditions for convergence in the mean square sense.

Theorem 3.1 *The LMS algorithm is convergent in the mean square sense if it satisfies the given assumptions and the condition:*

$$0 < \mu < \frac{2}{2\lambda_{\max} + trace[\Lambda]}. \tag{3.21}$$

Proof: Convergence of Equation (3.19) is given if the eigenvalues of matrix B are smaller than one in magnitude. Note that B is positive definite, that is all eigenvalues are positive. A sufficient condition for convergence is thus that the largest eigenvalue is smaller than one. The largest eigenvalue is given by the 2-induced norm. It can further be bounded by the 1-induced norm that is

$$\lambda_{\max} = \|B\|_{2,ind} \leq \|B\|_{1,ind}.$$

We can take an arbitrary row of B:

$$(1 - \mu\lambda_i)^2 + \mu^2\lambda_i \left(\lambda_i + \text{trace}[\Lambda]\right) < 1.$$

Reordering with respect to μ results in:

$$0 < \mu < \frac{2}{2\lambda_i + \text{trace}[\Lambda]}.$$

As this must be true for each eigenvalue, we find eventually the condition above. □

3.2.4 Describing Parameters

The convergence speed of the algorithm is given by the eigenvalues of matrix B in Equation (3.18). In general we expect a relation of the form

$$\text{tr}\{P_k\} = \underline{1}^T \underline{c}_k = \sum_{l=1} \gamma_l \left(\lambda_l^{(B)}\right)^k,$$

in which the eigenvalues $\lambda_l^{(B)}$ of matrix B are weighted. In this form we assume that all eigenvalues are different. The weighting factors depend also on the initial values of the parameter error vector. It can thus happen that particular eigenvalues have no appearance. If we consider the worst case then the largest eigenvalue of B will dominate the convergence speed. By Equation (3.18) we can describe the temporal movement of the adaptation process. If we on the other hand consider a single realization the process can look very different. The reason for this is that (3.18) describes the learning in the mean. Only if we ensemble average many realizations we will find a good agreement with the theoretical prediction. The ensemble averaged adaptation curves are called **learning curves**. Figure 3.1 displays a learning curve of the relative system mismatch for various

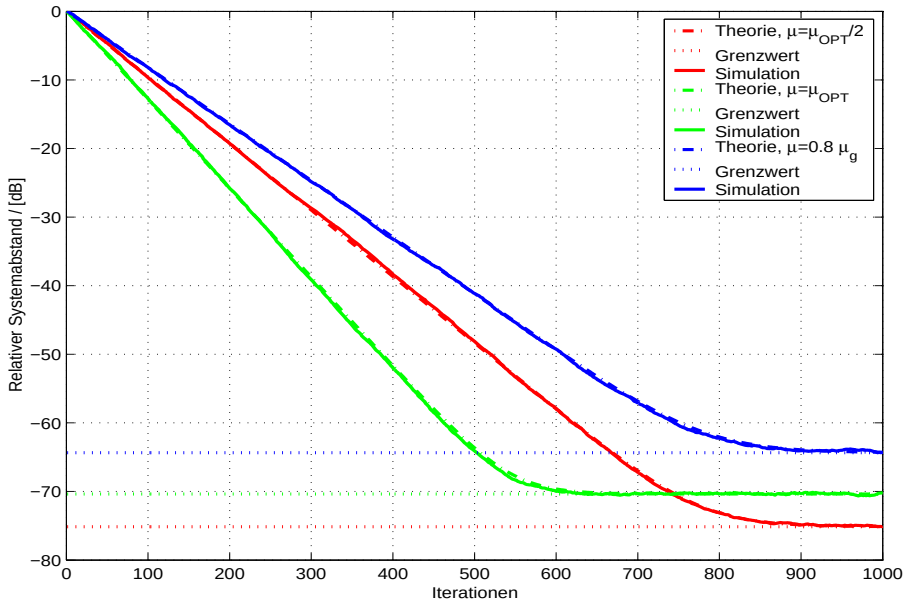

Figure 3.1: *Learning curves: relative system distance for various step-sizes.*

step-sizes.

Next to the convergence speed also the remaining parameter error vector also-called *mismatch* is of interest. This steady-state value is theoretically achieved for $k \to \infty$. Let us consider again Equation (3.18). For $k \to \infty$, the mismatch is given by

$$\begin{aligned} \lim_{k\to\infty} \underline{c}_k &= \underline{c}_\infty = B\underline{c}_\infty + \mu^2 \underline{\lambda}\sigma_{\mathbf{v}}^2 && (3.22)\\ &= [I-B]^{-1}\mu^2\underline{\lambda}\sigma_{\mathbf{v}}^2 && (3.23)\\ &= [2\Lambda - \mu 2\Lambda^2 - \mu\underline{\lambda}\underline{\lambda}^T]^{-1}\mu\underline{\lambda}\sigma_{\mathbf{v}}^2. && (3.24) \end{aligned}$$

Lemma 3.1 (Matrix-Inversion-Lemma) *For nonsingular matrices A and C the following is true:*

$$(A+BCD)^{-1} = A^{-1} - A^{-1}B[DA^{-1}B + C^{-1}]^{-1}DA^{-1}.$$

Proof: The proof is straightforwardly found by substitution and verification. □
We can further simplify the term by applying the matrix-inversion-lemma

$$\underline{1}^T\underline{c}_\infty = \mu\sigma_{\mathbf{v}}^2 \frac{\sum_{l=1}^{M} \frac{1}{2-2\mu\lambda_l}}{1-\mu\sum_{l=1}^{M}\frac{\lambda_l}{2-2\mu\lambda_l}}. \qquad (3.25)$$

For small step-sizes μ the term can further be simplified to

$$\underline{1}^T \underline{c}_\infty \approx \frac{\mu M \sigma_{\mathbf{v}}^2}{2}. \tag{3.26}$$

Often instead of the mismatch the **relative mismatch** is provided:

$$\frac{\underline{1}^T \underline{c}_\infty}{\|\underline{w}_o\|_2^2}.$$

This term is independent of the particular optimal value.

Even more interesting than the mismatch is the distorted a-priori error

$$\tilde{\mathbf{e}}_a(k) \triangleq \mathbf{d}(k) - \underline{\hat{\mathbf{w}}}_{k-1}^T \underline{\mathbf{x}}_k. \tag{3.27}$$

for $k \to \infty$. We obtain:

$$\begin{aligned}
\mathrm{E}[|\tilde{\mathbf{e}}_a(k)|^2] &= \mathrm{E}[|\mathbf{d}(k) - \underline{\hat{\mathbf{w}}}_{k-1}^T \underline{\mathbf{x}}_k|^2] && (3.28)\\
&= \mathrm{E}[|\mathbf{v}(k) + \underline{\tilde{\mathbf{w}}}_{k-1}^T \underline{\mathbf{x}}_k|^2] && (3.29)\\
&= \sigma_{\mathbf{v}}^2 + \mathrm{E}[|\underline{\tilde{\mathbf{w}}}_{k-1}^T \underline{\mathbf{x}}_k|^2] && (3.30)\\
&= \sigma_{\mathbf{v}}^2 + \mathrm{E}[\underline{\tilde{\mathbf{w}}}_{k-1}^T \underline{\mathbf{x}}_k \underline{\mathbf{x}}_k^H \underline{\tilde{\mathbf{w}}}_{k-1}^*] && (3.31)\\
&= \sigma_{\mathbf{v}}^2 + \mathrm{E}[\underline{\tilde{\mathbf{w}}}_{k-1}^T R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}^* \underline{\tilde{\mathbf{w}}}_{k-1}^*] && (3.32)\\
&= \sigma_{\mathbf{v}}^2 + \mathrm{trace}\{P_{k-1} R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}^*\} && (3.33)\\
&= \sigma_{\mathbf{v}}^2 + \underline{\lambda}^T \underline{c}_{k-1}. && (3.34)
\end{aligned}$$

Here, we can recognize the part of the LMS approximation. The Wiener solution only has a noise part $\sigma_{\mathbf{v}}^2$ while the LMS algorithm produces an additional error term g_{ex} called the *excess mean square error*. By applying the matrix inversion lemma we can also give a simple expression for this error:

$$g_{ex} = \underline{\lambda}^T \underline{c}_\infty = \mu \sigma_{\mathbf{v}}^2 \frac{\sum_{l=1}^M \frac{\lambda_l}{2-2\mu\lambda_l}}{1 - \mu \sum_{l=1}^M \frac{\lambda_l}{2-2\mu\lambda_l}}. \tag{3.35}$$

A further parameter that is often used is the so-called *misadjustment* (Ger.: Fehlanpassung). It is the excess mean square error relative to the Wiener solution:

$$m_{LMS} = \frac{g_{ex}}{g_o} = \mu \frac{\sum_{l=1}^M \frac{\lambda_l}{2-2\mu\lambda_l}}{1 - \mu \sum_{l=1}^M \frac{\lambda_l}{2-2\mu\lambda_l}}. \tag{3.36}$$

By this operation the misadjustment is independent of the noise variance. It is also approximately proportional to the step-size μ. By selecting an arbitrarily small step-size

we can thus make the misadjustment as small as desired. However, we loose convergence speed by this. Figure 3.2 displays learning curves for the mean square error energy. Different to the system distance the curves are rather rocky. The curves exhibited here are obtained by averaging over 50 ensemble values. By averaging over more curves, the learning curves become smoother and smoother.

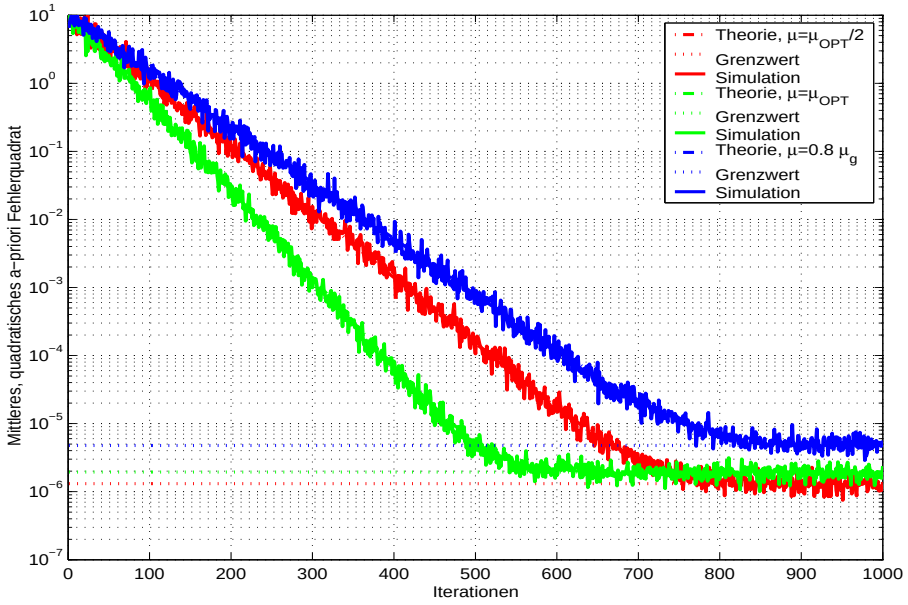

Figure 3.2: *Learning curves: mean squared a-priori error for various step-sizes.*

3.2.5 Convergence with probability one

Let us reconsider Equation (3.7), shown here again for the noise free case:

$$\underline{\tilde{\mathbf{w}}}_k = (I - \mu \underline{\mathbf{x}}_k^* \underline{\mathbf{x}}_k^T) \underline{\tilde{\mathbf{w}}}_{k-1};\ k = 1, 2, ... \tag{3.37}$$

$$= \prod_{l=1}^{k} (I - \mu \underline{\mathbf{x}}_l^* \underline{\mathbf{x}}_l^T) \underline{\tilde{\mathbf{w}}}_0. \tag{3.38}$$

Due to the independence assumption we can interpret the convergence in the mean squared sense as if we have isolated terms $(I - \mu \underline{\mathbf{x}}_l^* \underline{\mathbf{x}}_l^T)$ although in reality the entire product in (3.38) is of importance.

Example 3.1 *A parameter vector of length $M = 1$ is to be estimated by the LMS algorithm. Let the additive noise to be zero. We thus obtain:*

$$\tilde{\mathbf{w}}(k) = \left(1 - \mu|\mathbf{x}(k)|^2\right) \tilde{\mathbf{w}}(k-1);\ k = 1, 2, ... \tag{3.39}$$

$$= \prod_{l=1}^{k} \left(1 - \mu|\mathbf{x}(l)|^2\right) \tilde{\mathbf{w}}(0). \tag{3.40}$$

Squaring on both sides, we obtain the energy relation:

$$|\tilde{\mathbf{w}}(k)|^2 = \prod_{l=1}^{k} \left(1 - \mu|\mathbf{x}(l)|^2\right)^2 |\tilde{\mathbf{w}}(0)|^2. \tag{3.41}$$

Due to the independence assumption we find the convergence condition in the mean square sense by applying the expectation on both sides:

$$E\left[|\tilde{\mathbf{w}}(k)|^2\right] = E\left[\prod_{l=1}^{k} \left(1 - \mu|\mathbf{x}(l)|^2\right)^2\right] E\left[|\tilde{\mathbf{w}}(0)|^2\right] \tag{3.42}$$

$$= \prod_{l=1}^{k} E\left[\left(1 - \mu|\mathbf{x}(l)|^2\right)^2\right] E\left[|\tilde{\mathbf{w}}(0)|^2\right]. \tag{3.43}$$

It is thus sufficient to consider the term $E\left[\left(1 - \mu|\mathbf{x}(l)|^2\right)^2\right]$ to guarantee convergence in the mean square sense. For this the following needs to be true:

$$E\left[\left(1 - \mu|\mathbf{x}(l)|^2\right)^2\right] < 1 \tag{3.44}$$

or, equivalently for the step-size μ :

$$0 < \mu < \frac{2\sigma_{\mathbf{x}}^2}{m_{\mathbf{x}}^{(4)}}. \tag{3.45}$$

On the other hand, we can also require that convergence holds for the entire product. This is a much looser requirement than the condition that convergence holds for each term of the product. We can thus expect that this requirement leads to a much larger stability bound. To find this we apply the logarithm on both sides of (3.41) and obtain:

$$\ln\left(|\tilde{\mathbf{w}}(k)|^2\right) = \sum_{l=1}^{k} \ln\left(\left(1 - \mu|\mathbf{x}(l)|^2\right)^2\right) \ln\left(|\tilde{\mathbf{w}}(0)|^2\right). \tag{3.46}$$

Dividing this term by the number of iterations k and letting this number grow, we obtain for ergodic random processes $\mathbf{x}(k)$:

$$\lim_{k\to\infty} \frac{\ln\left(|\tilde{\mathbf{w}}(k)|^2\right)}{k} = \mathrm{E}\left[\ln\left(\left(1 - \mu|\mathbf{x}(l)|^2\right)^2\right)\right]. \tag{3.47}$$

Remark: This we can also argue by the law of large numbers as the elements $\mathbf{x}(l)$ are i.i.d. with bounded variance.

The condition for convergence is now, after we made use of the logarithm:

$$\mathrm{E}\left[\ln\left(\left(1-\mu|\mathbf{x}(l)|^2\right)^2\right)\right] < 0. \tag{3.48}$$

If we compare this with our previous condition on the convergence in the mean square sense, we can formulate this equivalently as:

$$\ln\left(\mathrm{E}\left[\left(1-\mu|\mathbf{x}(l)|^2\right)^2\right]\right) < 0. \tag{3.49}$$

In Figure 3.3 both functions are plotted for the case of a uniform distribution of $\mathbf{x}(k)$ in the range $[-1,+1]$. The convergence condition in the mean square sense delivers a stability bound for $\mu_g = 10/3 = 3.33$, while our new condition only requires $\mu_g = 6.1$. As the new condition was found by a stochastic limit, we call it **almost sure convergence** or **convergence with probability one**[1].

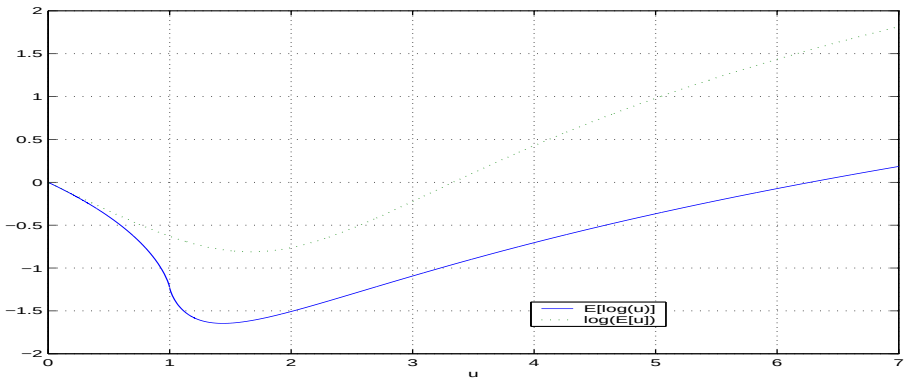

Figure 3.3: *Comparison of convergence conditions.*

Exercise 3.3 *Show that the adaptation*

$$\underline{\hat{\mathbf{w}}}_k = \underline{\hat{\mathbf{w}}}_{k-1} + \mu[\underline{\mathbf{x}}_k^*\underline{\mathbf{x}}_k^T + \epsilon\mathbf{I}]^{-1}\underline{\mathbf{x}}_k^*\tilde{e}_a(k)$$

leads to the $\epsilon-$NLMS algorithm for $\epsilon > 0$.

Exercise 3.4 *Show that the matrix B is positive definite.*

[1]See also Appendix B for more details.

Exercise 3.5 *Compute the step-size so that for a white random process the largest eigenvalue of B becomes minimal. How fast does the algorithm converge in dependence to the filter length M?*

Exercise 3.6 *Compute the misadjustment for the following cases:*

- $\lambda_{\min} = \lambda_{\max}$, *thus a white driving process,*
- $\lambda_{\min} = \lambda_1, \lambda_2 = \lambda_3 = ... = \lambda_M = \lambda_{\max}$,
- $\lambda_{\min} = \lambda_1 = \lambda_2 = ... = \lambda_{M-1}, \lambda_M = \lambda_{\max}$.

Exercise 3.7 *Provide the adaptation for a parameter error vector* $\underline{\tilde{\mathbf{w}}}_k$ *of length* $M = 1$ *and draw a signal flow graph for it. Which conditions for the step-size* μ *and the pdf of the driving process are required to obtain stability?*

Exercise 3.8 *Provide the adaptation for the parameter error vector* $\underline{\tilde{\mathbf{w}}}_k$ *of length* $M = 1$. *Assume the driving process to be bipolar noise of zero-mean with* $\sigma_{\mathbf{x}}^2 = 1$. *Compute the pdf of the error vector. Now compute mean and variance matrix of the parameter error vector for arbitrary length M.*

Exercise 3.9 *Substitute the a-priori error* $\tilde{\mathbf{e}}_a(k) = \mathbf{d}(k) - \underline{\mathbf{x}}_k^T \underline{\hat{\mathbf{w}}}_{k-1}$ *by the a-posteriori error* $\tilde{\mathbf{e}}_p(k) = \mathbf{d}(k) - \underline{\mathbf{x}}_k^T \underline{\hat{\mathbf{w}}}_k$. *Reformulate the gradient method so that only a-priori error terms occur.*

Exercise 3.10 *Consider the matrix Riccati equation*

$$\mathbf{K} = \mathbf{XKX} + \mathbf{R}$$

with $\mathbf{K}, \mathbf{X}, \mathbf{R}$ *Hermitian matrices. Given* $\mathbf{X}$ *and* $\mathbf{R}$, *solve the equation by using Kronecker properties and vectorization of* $\mathbf{X}$.
Apply the same method to solve for the recursive equation of the parameter covariance matrix $\mathbf{K}_k$ *in the LMS algorithm. What step-size condition for stability in the mean square sense can be derived?*

Exercise 3.11 *Show that the stability limit for convergence with probability one after Example 3.1 is indeed* $\mu_g = 6.1$.

Matlab Experiment 3.1 Write a Matlab Program for parameter identification. The driving process is a real-valued, zero-mean Gaussian process with $\sigma_{\mathbf{x}}^2 = 1$. Let the unknown system have $M = 32$ coefficients all different from zero. Run the LMS adaptation of a transversal filter for various step-sizes and plot the relative system mismatch as well as the a-priori error energy over time. Compare with theoretical results on fastest convergence and stability limit.
In a second experiment realize for each of the step-sizes 50 independent runs and plot the ensemble averaged value. Discuss the differences.

Matlab Experiment 3.2 Rerun the previous experiment however with a colored driving process, obtained by filtering the white process with the filter

$$F(z) = \frac{\sqrt{1-b^2}}{1-bz^{-1}}, \quad b = -0.7.$$

Compute the autocorrelation function and provide the acf matrix for $M = 32$. Repeat the previous experiment and compare to the theoretical values. Discuss the results.
Repeat the experiments with an NLMS algorithm. What is different now?

3.3 Application Specific Variants

In the following we will consider some implementation specific particularities of the LMS algorithm. In hardware architectures comprising of a Multiply-Accumulator unit (MAC) a complexity of $2M$ per iteration step is considered for the LMS algorithm. The first half is required for the error signal $\tilde{e}_a(k) = d(k) - \underline{\hat{w}}_{k-1}^T \underline{x}_k$, the second half for the update equation $\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu \tilde{e}_a(k) \underline{x}_k^*$. The operation $\mu \tilde{e}_a(k)$ is required only once and can be neglected for large M. If saving even this operation, the step-size can be realized as a factor of two, thus: $\mu = 2^l$. As this is only a shift operation, in an ASIC design this would not cost anything.

If scaling with respect to input energy is desired, it can be realized with little effort. For the NLMS algorithm for example, the computation:

$$\|\underline{x}_k\|_2^2 = \|\underline{x}_{k-1}\|_2^2 + |x(k)|^2 - |x(k-M)|^2 \tag{3.50}$$

can be achieved recursively. Note that this only works in fixed-point arithmetic. In floating point arithmetic such recursion can lead to cut-off and rounding errors. In this case a block operation is useful. The recursion is implemented over a length of M but at the same time a new block is computed in parallel and the correct results are taken at block boundaries

to avoid an increasing round-off effect:

$$P_{k|k} = P_{k-1} + |x(k)|^2 \tag{3.51}$$

$$\|\underline{x}_k\|_2^2 = \begin{cases} \|\underline{x}_{k-1}\|_2^2 + |x(k)|^2 - |x(k-M)|^2 & k \neq lM \\ P_{k|k} & \text{else} \end{cases} \tag{3.52}$$

$$P_k = \begin{cases} P_{k|k} & k \neq lM \\ 0 & \text{else} \end{cases} \tag{3.53}$$

At very high processing speeds the low complexity of $2M$ MAC operations may still be too high. Three variants are in use with reduced complexity with the drawback of less precision:

- **Sign-Regressor-LMS**: $\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu\ \tilde{e}_a(k)\ \text{sign}[\underline{x}_k^*]$.
- **Sign-Error-LMS**: $\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu\ \text{sign}[\tilde{e}_a(k)]\ \underline{x}_k^*$.
- **Sign-Sign-LMS**: $\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu\ \text{sign}[\tilde{e}_a(k)]\ \text{sign}[\underline{x}_k^*]$.

The sign operation is applied to every element in the vector individually. For complex-valued numbers it works independently on real and imaginary part. For sure the algorithms change their behavior due to such brute force changes.

Note that next to complexity also the data rate can become a problem. If the LMS algorithm is operated in two steps, (first the error computation, then the updates), then for each x a value from $\hat{w}$ needs to be loaded to compute the error. Then after the error is computed, again all values for x and $\hat{w}$ need to be loaded and finally $\hat{w}$ stored. If two parallel data buses are available, one for x and one for $\hat{w}$, the LMS algorithm could not be computed in $2M$ steps. In order to achieve the complexity in $2M$ steps, we have to include some further tricks. As in transversal filters the values in $\underline{x}_{k+1}$ are obtained by shifting all elements of $\underline{x}_k$ by one position and a single new value is introduced. Taking advantage of such property we can start computing the update error for $k+1$ already at k. With this we only require two load and one store operation per step.

Most research went into developing algorithms that are learning faster. As we have already seen that a correlated input process causes a slow learning, a natural way to speed up algorithms is to decorrelate them first. For example, it is possible to either know the correlation matrix $R_{\underline{\mathbf{xx}}}$ of the input process beforehand or to estimate it and then apply the following matrix step-size:

Newton-LMS: $\underline{\hat{\mathbf{w}}}_k = \underline{\hat{\mathbf{w}}}_{k-1} + \mu\tilde{e}_a(k)\left(R_{\underline{\mathbf{xx}}}^*\right)^{-1}\underline{\mathbf{x}}_k^*$.

This procedure unfortunately is very costly in terms of complexity. On the one hand we have to estimate the acf matrix, and then a matrix inversion is required, which can be

achieved with the order of M^2 when applying Levinson algorithm (see Section **??**).

If block processing is not allowed (due to delay constraints) other means are required. One possibility is to extend the scalar step-size to a matrix similar to the Newton LMS. It does not need to be the inverse of the acf matrix, a diagonal matrix with well chosen diagonal elements can also be of advantage. The diagonal elements can be chosen to be proportional to the expected weights, or adaptively selected depending on the estimated weights:

$$M_{k|k} = \begin{pmatrix} \left|(\underline{\hat{w}}_{k-1})_1\right| & & & \\ & \left|(\underline{\hat{w}}_{k-1})_2\right| & & \\ & & \ddots & \\ & & & \left|(\underline{\hat{w}}_{k-1})_M\right| \end{pmatrix} \tag{3.54}$$

$$M_k = \frac{\mu M_{k|k}}{\mathrm{trace}[M_{k|k}]} \tag{3.55}$$

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + M_k \tilde{e}_a(k) \underline{x}_k^*. \tag{3.56}$$

The algorithm is known under the name **Proportionate-weight NLMS** (=PNLMS).

Exercise 3.12 *Formulate the LMS algorithm for a transversal filter so that it requires only two load and one store operation.*

Exercise 3.13 *Show that the Newton LMS algorithm under correlated excitation behaves like an LMS under white excitation*

Exercise 3.14 *How does the filter structure in Figure* **??** *alter the distortion? Is it still a system identification?*

Matlab Experiment 3.3 Extend the algorithm from Experiment 3.2 by adding a prefiltering after Schultheiss and compare the results. Is the learning rate the same as if a white excitation was applied? Also implement the Newton LMS algorithm and compare.

3.4 Literature

Good overviews on adaptive filters and LMS algorithm can be found in [33, 86, 76, 46]. The original idea of the LMS algorithm goes back to Widrow and Hoff [84], although

gradient type algorithms in similar form can be found in older literature. The independence assumptions were introduced by [50]; in [13] a very complex method is derived to make exact predictions without the independence assumption, however the results are not in analytical form. The here shown derivation of the parameter error vector is based on [35, 20], although older work [81] was already moving along such paths. Extensions to the NLMS algorithm can be found in [4, 55] and spherically invariant processes in [57]. A first analysis of the Sign-Error algorithm can be found in [11]. A good explanation for convergence with probability one is in [74]. A deeper understanding is provided in [76]. Polyphase filter banks for hands-free telephony were introduced by Kellermann [40] and for equalizers in [51]. The excitation with sinusoidal signals was introduced by [24, 12], the feedback structure in this form the first time in [62]. The PNLMS can be found in [16, 23, 69].

Chapter 4

Classification Algorithms

4.1 Basic Neuron Models

To understand how the human (and animal) brain is able to discriminate different objects, it may be useful to understand its basic elements. As depicted in Figure 4.1, the units of interest are called neurons as they appear in 100 billions in the human brain (and 100 millions in the intestines!). They are particular cells with a nucleus and an axon that can be relatively long (several meters), comprising of small so-called nodes of Ranvier that transport information in form of moving ions. At the end of such an axon sit the axon terminals that make the connection to other cells via the dendrites at the cell nuclei. The neuron function in a very simplified form is abstracted by a linear combiner with weights c_l that describe the dendrite behavior. They can have positive inforcing weights or negative inhibiting weights.

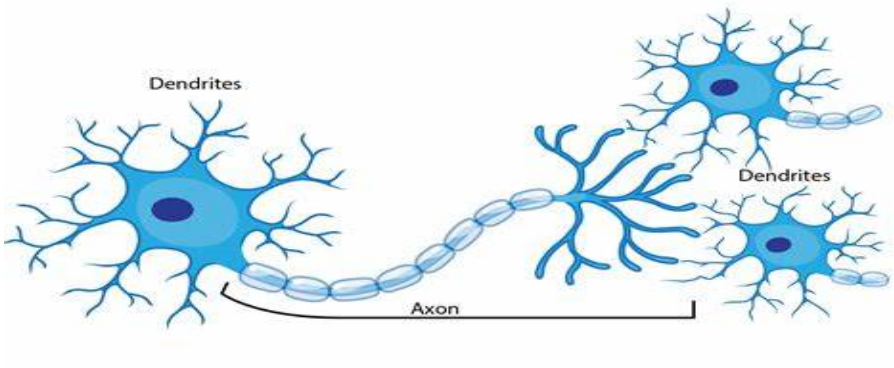

Figure 4.1: *Neuron with axon and dendrites.*

However, the neuron only sends an electric pulse over the axon when a specific threshold is reached. This is modelled with a nonlinear so-called activation function $f(x)$. We have

already discussed the sigmoid function $\sigma(x)$ as well as the tanh() function[1] which have very favourable properties such as low complexity derivatives (see right part of Figure 4.2). However, a closer inspection to measured activation functions as shown in the right part of Figure 4.2 reveals that also another property can be of interest. Such rectifying property as emphasized in the red box of the figure can be modelled as so-called Rectifier-Linear-Unit (ReLu), also referred to as "the hinge":

$$f(x) = \max(0, x).$$

We thus end with a relative simple model

$$y_i = f(c_o + c_1 x_{i,1} + ... + c_M x_{i,M}) = f(\underline{\hat{x}}^H \underline{\hat{c}}),$$

which is often referred to as the (single layer) perceptron as it is an equivalent of a neuron but certainly not the same.

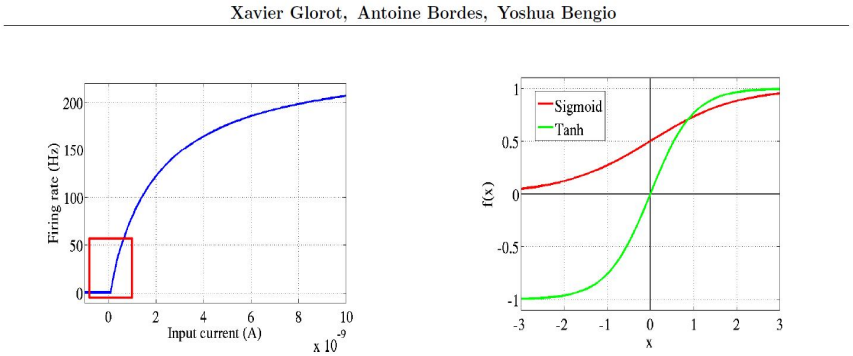

Figure 1: *Left:* **Common neural activation function motivated by biological data.** *Right:* **Commonly used activation functions** in neural networks literature: logistic sigmoid and hyperbolic tangent (*tanh*).

Figure 4.2: *Left: ReLu, right: common nonlinear functions for classification.*

In Figure 4.3 such a ReLu is shown together with a soft approximation:

$$f(x) = \text{soft}(x) = \log(1 + e^x).$$

Such approximation can be very useful as discontinuous behavior is avoided and differentiation is guaranteed:

$$\frac{\partial}{\partial x}\text{soft}(x) = \frac{1}{1 + e^{-x}} = \sigma(x).$$

[1]Note that in some of the literature these two functions are often referred to as *squashing* function due to their compression property.

Furthermore, the second derivative turns out to be

$$\frac{\partial^2}{\partial x^2}\text{soft}(x) = \sigma(x)(1-\sigma(x)),$$

which is non-negative and convex.

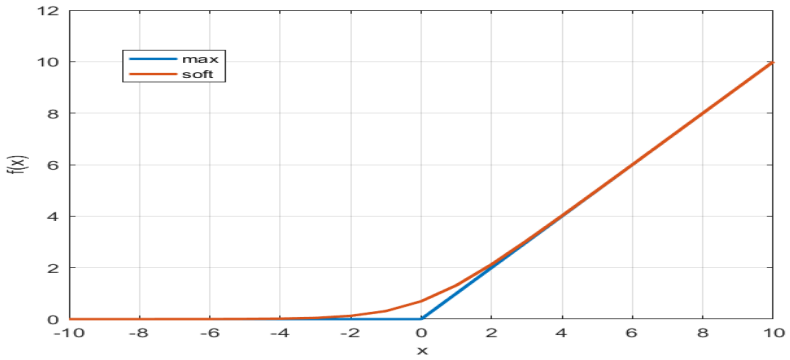

Figure 4.3: *The hinge as soft-decision function.*

4.2 Separability

Before we discuss separability, the "right" form of description for binary classification may be required, that is rather than $\{0,1\}$ we may want to have$\{-1,1\}$ or $\{-A,A\}$. We can simply obtain one description from the other by a so-called affine transformation. For example, $2x-1$ transforms:

$$2\{0,1\}-1 = \{-1,1\}.$$

Returning the binary classification problem as introduced in the first chapter, we recall that

$$\begin{aligned} \underline{\hat{x}}_i^H \underline{\hat{c}} > 0 \quad &\text{if} \quad y_i = +1 \\ \underline{\hat{x}}_i^H \underline{\hat{c}} < 0 \quad &\text{if} \quad y_i = -1 \end{aligned}$$

which can equivalently and more compactly be formulated as

$$\begin{aligned} y_i(\underline{\hat{x}}^H \underline{\hat{c}}) &> 0 \\ -y_i(\underline{\hat{x}}^H \underline{\hat{c}}) &< 0. \end{aligned} \tag{4.1}$$

If a hyperplane defined by $\underline{c}$ exists that correctly separates the two classes, then we find that

$$\max(0, -y_i(\underline{\hat{x}}^H \underline{\hat{c}})) = 0.$$

If, on the other hand, a hyperplane is not doing a perfect job, then we find

$$\max(0, -y_i(\underline{\hat{x}}^H \underline{\hat{c}})) > 0.$$

We can thus simply check for all provided data points (in the training set) what value we find for

$$g(\underline{\hat{c}}) = \sum_{i=1}^{N} \max(0, -y_i(\underline{\hat{x}}^H \underline{\hat{c}})).$$

No matter if separable or not, the best choice is

$$\underline{\hat{c}}_{opt} = \arg\min_{\underline{\hat{c}}} g(\underline{\hat{c}}).$$

Some examples of this are shown in Figure 4.4. On the left hand side the two classes cannot perfectly be separated by a hyperplane. Thus the cost function remains positive, but relatively small, indicating that only a small subset of points cannot be classified correctly. The right part of the figure shows a setup where a separating hyperplane can be found easily and thus $g(\underline{\hat{c}}) = 0$. Looking at the right part of the figure, repeated in Figure 4.5 we

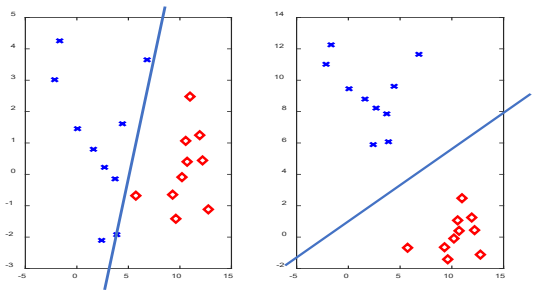

Figure 4.4: *Left: non perfectly separable, right: perfectly separable.*

wonder if there is a "street" that is open the widest. As we could then set our hyperplane right in the middle of such street and we would have the largest possible margin to the first data points in each class. As the figure shows the street with the continuous lines is wider than the one with the dashed lines. Obviously more than one solution exists and we would be interested in the solution with the largest margins.

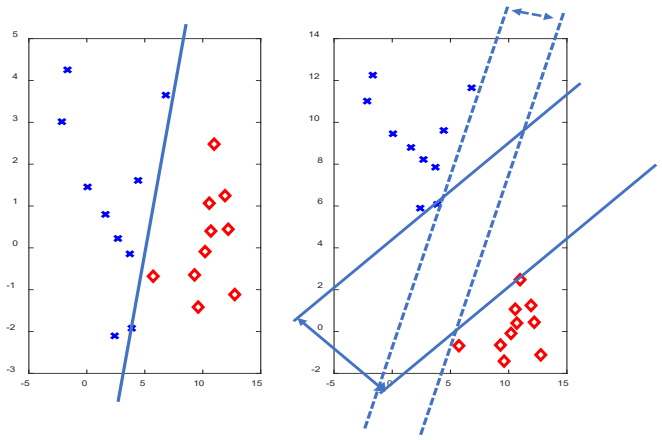

Figure 4.5: *Left: non perfectly separable, right: perfectly separable, showing two solutions with different gap.*

4.3 Margin Perceptron Learning

We now wonder how to maximize the margin, or the width of such a street. First we will reformulate our cost function based on the margin points, which is referred to as the margin perceptron. Different to before we now recognize that we do not have a single separation line (hyperplane) but two co-linear ones for which holds

$$\begin{aligned}(\underline{\hat{x}}^H\underline{\hat{c}})y_i &\geq +1\\ -(\underline{\hat{x}}^H\underline{\hat{c}})y_i &\leq -1\end{aligned}$$

which we can combine into

$$1 - y_i(\underline{\hat{x}}^H\underline{\hat{c}}) \leq 0.$$

Our cost function thus reads now

$$g(\underline{\hat{c}}) = \sum_{i=1}^{N} \max(0, 1 - y_i(\underline{\hat{x}}^H\underline{\hat{c}})),$$

for which we have to find the best choice

$$\underline{\hat{c}}_{opt,margin} = \arg\min_{\underline{c}} g(\underline{\hat{c}}).$$

As the cost function is rather non-linear we do not know a simple solution but to use a gradient approach. For this to work we have to make the nonlinear mapping smooth enough, thus we replace the ReLu with a soft decision:

$$g(\underline{\hat{c}}) = \sum_{i=1}^{N} \text{soft}(1 - y_i(\underline{\hat{x}}^H \underline{\hat{c}})).$$

To inspect the situation further, we have redrawn the most important part of the previous figure in Figure 4.6. We recognize that the two parallel lines that build the border of the class both must have at least one point of each class on them. There are the margin points, which define by their coordinates the so-called margin vectors $\underline{\hat{x}}_-$ and $\underline{\hat{x}}_+$. Computing the difference of the two vectors and projecting them orthogonal onto the street, we find

$$\left(\underline{\hat{x}}_+ - \underline{\hat{x}}_-\right) \frac{\underline{c}}{||\underline{c}||} = \frac{1 - (-1)}{||\underline{c}||} = \frac{2}{||\underline{c}||}. \tag{4.2}$$

Note that this vector $\underline{c}$ here contains only all weights up to the bias term, as the margin vector description also did not contain the bias term. Once $\underline{c}$ is identified the margin term is easily found by placing the hyperplane in the center between the margins. Thus, once we have found the vector that provides us the widest street, the width of the street is immediately given. As we like to have the widest street, we should minimize $||\underline{c}||$. We can

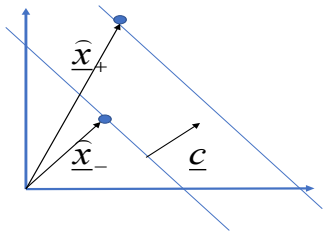

Figure 4.6: *SVM: how to incorporate the margins.*

formulate this simply by

$$\begin{gathered} \min ||\underline{c}|| \\ \text{subject to} \\ \max(0, 1 - y_i(\underline{\hat{x}}^H \underline{\hat{c}})) = 0 \quad ; i = 1, 2, ..., N. \end{gathered}$$

As we needed the support of the margin vectors, the method is called Vector Support Machine (SVM). Note that this formulation obtains $\underline{c}$ as well as $\hat{\underline{c}}$ but the optimization only delivers $\underline{c}$. we thus still need to adjust the hyperplane correctly.

Chapter 5

Robust Adaptive Filters

In the following we will give a brief introduction into the theory of robust adaptive filtering as it appears to be the right tool to describe gradient term algorithms that include non-linearities such as an activation function. A so-called classical H_∞ formulation delivers a robust solution form but does not tell us whether the filter converges. We will thus have to follow a different path in the following and derive robustness directly by means of energy passivity relations. The method allows to cover a very large class of adaptive algorithms. We start with the classic LMS algorithm that we already analyzed in the context of a stochastic setup. Now, however, we will work out its behavior in a purely deterministic environment.

Closely related with robust filtering are the passivity relations. If we consider the signals $\underline{\tilde{w}}_o$ and $v(k)$ as inputs and the corresponding error signals as output of the adaptive algorithm, we recognize that for their ratio smaller than one, less energy comes out of the system than goes in. If this is the case we call the system passive.

5.1 Local Passivity Relations

To derive passivity relations let us assume that we have selected a specific step-size so that $\mu(k)\|\underline{x}_k\|_2^2 \leq 1$ and that we have an arbitrary system estimate $\underline{q}$. By means of the Cauchy-Schwarz' inequality we can write:

$$\frac{\left|\underline{x}_k^T\underline{w} - \underline{x}_k^T\underline{q}\right|^2}{\mu^{-1}(k)\|\underline{w} - \underline{q}\|_2^2} \leq 1\,. \tag{5.1}$$

To avoid mathematic difficulties in case $\underline{q} = \underline{w}$, we can also argue with $|\underline{x}_k^T\underline{w} - \underline{x}_k^T\underline{q}|^2 - \mu^{-1}(k)\|\underline{w} - \underline{q}\|_2^2 \leq 0$ instead. The term is certainly still correct if we extend the numerator

by

$$\frac{\left|\underline{x}_k^T\underline{w} - \underline{x}_k^T\underline{q}\right|^2}{\mu^{-1}(k)\|\underline{w}-\underline{q}\|_2^2 + |v(k)|^2} \leq 1\,. \tag{5.2}$$

As such relations are true for any arbitrary estimate $\underline{q}$, as long as $\mu(k)\|\underline{x}_k\|_2^2 \leq 1$ holds, they also must be true for the estimates of the LMS algorithm, thus for

$$\frac{\left|\underline{x}_k^T\underline{w} - \underline{x}_k^T\underline{\hat{w}}_{k-1}\right|^2}{\mu^{-1}(k)\|\underline{w}-\underline{\hat{w}}_{k-1}\|_2^2 + |v(k)|^2} \leq 1\,. \tag{5.3}$$

The urgent question is now, in which sense an LMS estimate would change such limit. Denoting $e_a(k) = \underline{x}_k^T[\underline{w} - \underline{\hat{w}}_{k-1}] = \underline{x}_k^T\underline{\tilde{w}}_{k-1}$ for the undistorted a-priori error, $e_p(k) = \underline{x}_k^T[\underline{w} - \underline{\hat{w}}_k] = \underline{x}_k^T\underline{\tilde{w}}_k$ for the undistorted a-posteriori error and $\gamma(k) = [\mu^{-1}(k) - \|\underline{x}_k\|_2^2]$, then the following theorem holds.

Theorem 5.1 (Local Passivity Property) *For the adaptive gradient method (LMS algorithm with variable step-size) we have at every time instant k:*

$$\begin{aligned}
\frac{\mu^{-1}(k)\|\underline{w}-\underline{\hat{w}}_k\|_2^2 + |e_a(k)|^2}{\mu^{-1}(k)\|\underline{w}-\underline{\hat{w}}_{k-1}\|_2^2 + |v(k)|^2} &\leq 1, \\
\frac{|e_a(k)|^2 + |e_p(k)|^2}{\mu^{-1}(k)\|\underline{w}-\underline{\hat{w}}_{k-1}\|_2^2 + |v(k)|^2} &\leq 1\,, \\
\frac{\gamma(k)\|\underline{w}-\underline{\hat{w}}_k\|_2^2 + |e_p(k)|^2}{\gamma(k)\|\underline{w}-\underline{\hat{w}}_{k-1}\|_2^2 + |v(k)|^2} &\leq 1\,, \\
\frac{|e_a(k)|^2 + |e_a(k+1)|^2}{\mu^{-1}(k)\|\underline{w}-\underline{\hat{w}}_{k-1}\|_2^2 + |v(k)|^2} &\leq 1\,.
\end{aligned} \tag{5.4}$$

The first three relations are true if $\mu(k)\|\underline{x}_k\|_2^2 \leq 1$ *while* $\mu(k) \leq \min\left\{1/\|\underline{x}_k\|_2^2, 1/\|\underline{x}_{k+1}\|_2^2\right\}$ *is required for the last.*

Proof: We show the first relation. The update equations for the parameter error vector are:

$$\underline{\tilde{w}}_k = \underline{\tilde{w}}_{k-1} - \mu(k)\underline{x}_k^*[e_a(k) + v(k)], \tag{5.5}$$

where we split the distorted a-priori error $\tilde{e}_a(k) = e_a(k) + v(k)$ into an undistorted a-priori error and noise. Computing the quadratic l_2-norm on both sides, we obtain

$$\|\underline{\tilde{w}}_k\|_2^2 = \|\underline{\tilde{w}}_{k-1}\|_2^2 + \mu^2(k)\|\underline{x}_k\|_2^2|e_a(k)+v(k)|^2 - \mu(k)[e_a(k)+v(k)]e_a^*(k) - \mu(k)[e_a(k)+v(k)]^*e_a(k).$$

Note that

$$|e_a(k) + v(k)|^2 = |e_a(k)|^2 + |v(k)|^2 + e_a(k)v^*(k) + e_a^*(k)v(k).$$

We thus obtain

$$\begin{aligned} \|\underline{\tilde{w}}_k\|_2^2 &= \|\underline{\tilde{w}}_{k-1}\|_2^2 + \mu^2(k)\|\underline{x}_k\|_2^2 \left|e_a(k)+v(k)\right|^2 \\ &\quad -2\mu(k)|e_a(k)|^2 - \mu(k)\left[|e_a(k)+v(k)|^2 - |e_a(k)|^2 - |v(k)|^2\right]. \end{aligned} \tag{5.6}$$

Eventually, by reordering the terms we find

$$\|\underline{\tilde{w}}_k\|_2^2 - \|\underline{\tilde{w}}_{k-1}\|_2^2 + \mu(k)|e_a(k)|^2 - \mu(k)|v(k)|^2 = \mu(k)|e_a(k)+v(k)|^2\left[\mu(k)\|\underline{x}_k\|_2^2 - 1\right].$$

The right-hand side is negative, as long as $\mu(k)\|\underline{x}_k\|_2^2 \leq 1$. We thus have

$$\|\underline{\tilde{w}}_k\|_2^2 - \|\underline{\tilde{w}}_{k-1}\|_2^2 + \mu(k)|e_a(k)|^2 - \mu(k)|v(k)|^2 \leq 0. \tag{5.7}$$

Dividing by $\|\underline{\tilde{w}}_{k-1}\|_2^2 + \mu(k)|v(k)|^2$ and we obtain the desired relation. □

Exercise 5.1 *Derive the last three relations of Theorem 5.1.*

Exercise 5.2 *Show that the LMS algorithm can be derived by the local cost function*

$$\|\underline{w}_k - \underline{w}_{k-1}\|_2^2 + \underline{\lambda}^T\tilde{e}_p(k) + \underline{\lambda}^H\tilde{e}_p^*(k). \tag{5.8}$$

Here, $\underline{\lambda}$ is a Lagrangian multiplier.

5.2 Robustness Analysis of Gradient Type Algorithms

Although we have not proven robustness with our local passivity relations but we have achieved a first step along this path. What is missing is a global relation spanning from the initial time $k = 1$ to an arbitrary time instant $k = N$. In order to find such a global relation, we start with our local property (5.7) and reformulate it for $1 \leq k \leq N$,

$$\mu(k)|e_a(k)|^2 \;\leq\; \|\underline{w} - \underline{w}_{k-1}\|_2^2 - \|\underline{w} - \underline{w}_k\|_2^2 + \mu(k)|v(k)|^2\,.$$

Summing up over k from $k = 1$ to $k = N$, we obtain

$$\frac{\|\underline{\tilde{w}}_N\|_2^2 \;+\; \sum_{k=1}^N \mu(k)|e_a(k)|^2}{\|\underline{\tilde{w}}_0\|_2^2 + \sum_{k=1}^N \mu(k)|v(k)|^2} \;\leq\; 1, \tag{5.9}$$

The numerator of (5.9) is thus the energy of the undistorted normalized a-priori error $\sqrt{\mu(k)}e_a(k)$ from $1 \leq k \leq N$, plus the energy of the remaining parameter error vector at time instant N. Correspondingly, the denominator also consists of two terms: the

energy of the normalized noise/disturbance over the entire time period as well as the initial parameter error vector energy. This is a global energy relation: a matrix $\mathcal{T}_N$ maps the signals $\{\sqrt{\mu(k)}v(k)\}_{k=1}^N$ and $\underline{\tilde{w}}_0$ onto the normalized a-priori error signals $\{\sqrt{\mu(k)}e_a(k)\}_{k=1}^N$ and the remaining parameter error vector $\underline{\tilde{w}}_N$.

$$\begin{bmatrix} \sqrt{\mu(1)}e_a(1) \\ \vdots \\ \sqrt{\mu(N)}e_a(N) \\ \underline{\tilde{w}}_N \end{bmatrix} = \underbrace{\begin{bmatrix} x & & & \\ x & x & & \\ \vdots & & \ddots & \\ x & x & x & \end{bmatrix}}_{\mathcal{T}_N} \begin{bmatrix} \underline{\tilde{w}}_0 \\ \sqrt{\mu(1)}v(1) \\ \vdots \\ \sqrt{\mu(N)}v(N) \end{bmatrix}. \tag{5.10}$$

Such a matrix must be passive, according to (5.9), or contracting that is the induced l_2-norm of the matrix is bounded: $\|\mathcal{T}_N\|_{2,ind} \leq 1$. In terms of robust control, such induced matrix norm is called H_∞ norm. Figure 5.1 illustrates the relation.

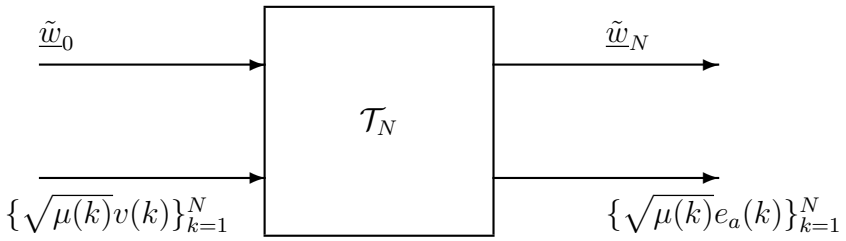

Figure 5.1: *Contracting Mapping $\mathcal{T}_N$.*

Exercise 5.3 *Show the robustness of the LMS algorithm with fixed step-size μ. Which relation has μ to satisfy, so that the algorithm is robust?*

Exercise 5.4 *Consider the gradient algorithm with regular step-size matrix M_k:*

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + M_k \underline{x}_k^* \tilde{e}_a(k). \tag{5.11}$$

Derive the following relations for $0 < \underline{x}_k^H M_k \underline{x}_k \leq 1$, $\gamma(k) = 1 - \underline{x}_k^H M_k \underline{x}_k$:

$$\frac{\underline{\tilde{w}}_k^H M_k^{-1} \underline{\tilde{w}}_k + |e_a(k)|^2}{\underline{\tilde{w}}_{k-1}^H M_k^{-1} \underline{\tilde{w}}_{k-1} + |v(k)|^2} \leq 1, \tag{5.12}$$

$$\frac{\gamma(k)\underline{\tilde{w}}_k^H M_k^{-1} \underline{\tilde{w}}_k + |e_p(k)|^2}{\gamma(k)\underline{\tilde{w}}_{k-1}^H M_k^{-1} \underline{\tilde{w}}_{k-1} + |v(k)|^2} \leq 1, \tag{5.13}$$

$$\frac{|e_a(k)|^2 + |e_p(k)|^2}{\underline{\tilde{w}}_{k-1}^H M_k^{-1} \underline{\tilde{w}}_{k-1} + |v(k)|^2} \leq 1. \tag{5.14}$$

For $M_k = \nu(k)M$ and $\nu(k) > 0, M > 0$ derive the robustness conditions of this algorithm from the first relation.

5.2.1 Minimax Optimality of Gradient Method

Let us consider again (5.9) in the following formulation:

$$\frac{\|\underline{w} - \underline{\hat{w}}_N\|_2^2 \; + \; \sum_{k=1}^N \mu(k)|e_a(k)|^2}{\|\underline{w} - \underline{\hat{w}}_0\|_2^2 + \sum_{k=1}^N \mu(k)|v(k)|^2} \; \leq \; 1. \tag{5.15}$$

Which value exactly maximizes the expression? If we consider a particular noise sequence $v(k) = -e_a(k)$, we recognize that the gradient method is not updating; in other words: the estimate $\underline{\hat{w}}_k$ remains at its initial value $\underline{\hat{w}}_k = \underline{w}_0$. We thus have

$$\max_{\underline{\hat{w}}_0 \neq \underline{w}, \sqrt{\mu(k)}v(\cdot)} \frac{\|\underline{w} - \underline{\hat{w}}_N\|_2^2 \; + \; \sum_{k=1}^N \mu(k)|e_a(k)|^2}{\|\underline{w} - \underline{\hat{w}}_0\|_2^2 + \sum_{k=1}^N \mu(k)|v(k)|^2} \; = \; 1. \tag{5.16}$$

The selection of $\underline{\hat{w}}_0 \neq \underline{w}$ is of technical matter. If we allow $\underline{\hat{w}}_0 = \underline{w}$, the the denominator can become zero. In this case we would have to argue with differences rather than ratios.

Let us consider now the ratio in (5.15) for an arbitrary algorithm $\mathcal{A}$. If we select again $v(k) = -e_a(k)$, and this time $\underline{\hat{w}}_0 = \underline{w}$, then we will have

$$\sum_{k=1}^N \mu(k)|v(k)|^2 = \sum_{k=1}^N \mu(k)|e_a(k)|^2 \leq \|\underline{w} - \underline{\hat{w}}_N\|_2^2 + \sum_{k=1}^N \mu(k)|e_a(k)|^2, \tag{5.17}$$

or differently written:

$$\max_{\bar{v}(\cdot)} \frac{\|\underline{w} - \underline{\hat{w}}_N\|_2^2 \; + \; \sum_{k=1}^N \mu(k)|e_a(k)|^2}{\|\underline{w} - \underline{\hat{w}}_0\|_2^2 + \sum_{k=1}^N \mu(k)|v(k)|^2} \; \geq \; 1. \tag{5.18}$$

If for an arbitrary algorithm $\mathcal{A}$ the relation (5.18) is true, while we know that for the gradient method (5.16) is true, then we can summarize this property in the following theorem.

Theorem 5.2 (Minimax Property of the gradient type algorithm) *The gradient type algorithm solves for $\mu(k)\|\underline{x}_k\|_2^2 \leq 1$ the following minimax problem:*

$$\min_{\textit{class of algorithms}} \; \max_{\underline{\hat{w}}, \sqrt{\mu(k)}v(\cdot)} \frac{\|\underline{w} - \underline{\hat{w}}_N\|_2^2 \; + \; \sum_{k=1}^N \mu(k)|e_a(k)|^2}{\|\underline{w} - \underline{\hat{w}}_0\|_2^2 + \sum_{k=1}^N \mu(k)|v(k)|^2}. \tag{5.19}$$

Moreover, its optimal value is one.

5.2.2 Sufficient Convergence Conditions

Theorem 5.3 (Convergence Conditions) *For* $\mu(k)\|\underline{x}_k\|_2^2 \leq 1$, $\|\underline{\tilde{w}}_0\| < \infty$ *and* $\sum_{k=1}^{\infty}\mu(k)|v(k)|^2 < \infty$ *the following is true:*

$$\sqrt{\mu(k)}e_a(k) \to 0. \tag{5.20}$$

If furthermore $\sqrt{\mu(k)}\underline{x}_k$ *is a persistent excitation, (see Lemma 9.3), then we also have*

$$\underline{\hat{w}}_k \to \underline{w}. \tag{5.21}$$

Proof: With (5.9) we have:

$$\sum_{k=1}^{N}\mu(k)|e_a(k)|^2 \leq \|\underline{\tilde{w}}_0\|_2^2 + \sum_{k=1}^{N}\mu(k)|v(k)|^2 \tag{5.22}$$

For a bounded initial error $\|\underline{\tilde{w}}_0\|_2^2 < \infty$ and bounded disturbance energy, the energy of the a-priori error must be bounded as well. Infinite series of finite energy are Cauchy series and thus: $\sqrt{\mu(k)}e_a(k) \to 0$.
Consider further the update equation at time instant k:

$$\underline{\tilde{w}}_k = \underline{\tilde{w}}_{k-1} - \mu(k)\tilde{e}_a(k)\underline{x}_k^*, \tag{5.23}$$

we can write equivalently

$$\underline{\tilde{w}}_{k+p-1} = \underline{\tilde{w}}_k - \sum_{l=1}^{p-1}\mu(k+l)\tilde{e}_a(k+l)\underline{x}_{k+l}^*. \tag{5.24}$$

Consider a series of $P > M$ vectors $\underline{x}_{k+p}; p = 1..P$, all of them can be tested on a value $\underline{\tilde{w}}_k$ and we find

$$\underline{x}_{k+p}^T\underline{\tilde{w}}_k = \underline{x}_{k+p}^T\underline{\tilde{w}}_{k+p-1} + \sum_{l=1}^{p-1}\mu(k+l)\tilde{e}_a(k+l)\underline{x}_{k+p}^T\underline{x}_{k+l}^* \tag{5.25}$$

$$= \tilde{e}_a(k+p) + \sum_{l=1}^{p-1}\mu(k+l)\tilde{e}_a(k+l)\underline{x}_{k+p}^T\underline{x}_{k+l}^*. \tag{5.26}$$

Because all $e_a(k) \to 0$ and also due to the bounded energy we have $v(k) \to 0$, and thus we conclude that $\tilde{e}_a(k) \to 0$. We thus have shown that the right-hand side of (5.26) converges towards zero. Concatenating all vectors $\underline{x}_{k+p}$ we find further:

$$\begin{bmatrix} \underline{x}_{k+1}^T \\ \underline{x}_{k+2}^T \\ \vdots \\ \underline{x}_{k+P}^T \end{bmatrix} \underline{\tilde{w}}_k \to \underline{0}. \tag{5.27}$$

Unfortunately, we cannot conclude form here that $\underline{\tilde{w}}_k \to 0$, as $\underline{\tilde{w}}_k$ could have components in the null space of the matrix. A consequence of the persistent excitation condition is that the matrix is of full rank M and thus its null space must be zero. Formally this can be shown by multiplying the hermitian transpose matrix from the left. We then have

$$\left[\underline{x}^*_{k+1}, \underline{x}^*_{k+2}, ..., \underline{x}^*_{k+P}\right] \begin{bmatrix} \underline{x}^T_{k+1} \\ \underline{x}^T_{k+2} \\ \vdots \\ \underline{x}^T_{k+P} \end{bmatrix} \underline{\tilde{w}}_k \to \underline{0}. \tag{5.28}$$

Due to the persistent excitation condition (Lemma 9.3) the left hand side lies between αI and βI, with $0 < \alpha < \beta$. Thus the matrix is regular and the null space becomes empty.□

Some interesting consequences follow. In case the noise sequence $v(k)$ is not of bounded energy a bound must be guaranteed by the step-size, for example by $\mu(k) = a/[b+k]^2$.

5.2.3 The Feedback Nature of the Gradient Method

Until here we assume that $\mu(k)\|\underline{x}_k\|_2^2 \leq 1$. On the other hand we already know that for larger values of the step-size the gradient method may work. We thus expect that it is possible to extend the local bounds.

Lemma 5.1 *For the gradient type method we find at each time instant k with $\bar{\mu}(k) = 1/\|\underline{x}_k\|_2^2$:*

$$\frac{\|\underline{\tilde{w}}_k\|_2^2 + \mu(k)\,|e_a(k)|^2}{\|\tilde{w}_{k-1}\|_2^2 + \mu(k)\,|v(k)|^2} \begin{cases} \leq 1 & for \quad 0 < \quad \mu(k) < \quad \bar{\mu}(k) \\ = 1 & if \qquad\quad \mu(k) = \quad \bar{\mu}(k) \\ \geq 1 & for \qquad\quad \mu(k) > \quad \bar{\mu}(k) \end{cases} \tag{5.29}$$

Proof: The first relation has been shown already. The second is obtained by substituting $\mu(k) = \bar{\mu}(k)$. For the third relation we consider again (5.1):

$$\|\underline{\tilde{w}}_k\|_2^2 - \|\underline{\tilde{w}}_{k-1}\|_2^2 + \mu(k)|e_a(k)|^2 - \mu(k)|v(k)|^2 = \mu(k)|e_a(k) + v(k)|^2[\mu(k)\|\underline{x}_k\|_2^2 - 1].$$

For $\mu(k) > \bar{\mu}(k)$ we find that $[\mu(k)\|\underline{x}_k\|_2^2 - 1] > 0$!

An interesting outcome of this considerations is that the local relation describes an allpass (thus lossless) always when the step-size $\mu(k) = \bar{\mu}(k)$ is selected. Obviously the induced l_2-norm becomes one, thus $\|\mathcal{T}_k\|_{2,ind} = 1$. We can also reformulate our original relation including an arbitrary step-size into a relation with particular step-size $\bar{\mu}(k)$

$$\begin{aligned} \underline{\hat{w}}_k &= \underline{\hat{w}}_{k-1} + \mu(k)\underline{x}^*_k\,[e_a(k) + v(k)] \\ &\triangleq \underline{\hat{w}}_{k-1} + \bar{\mu}(k)\underline{x}^*_k[e_a(k) + \bar{v}(k)]\,, \end{aligned} \tag{5.30}$$

and the newly introduced abbreviation

$$-\bar{v}(k) = e_a(k) - \frac{\mu(k)}{\bar{\mu}(k)}\left[e_a(k) + v(k)\right] = e_p(k). \tag{5.31}$$

Figure 5.2 illustrated this structure. The gradient type method can thus be explained as feedback structure. In the forward path there is a lossless system, an allpass with $\|\mathcal{T}_k\|_{2,ind} = 1$, while the feedback path contains a lossy system. Compare this with Equation (??) and Figure ??. Such local relation can also be reformulated into a global one, allowing

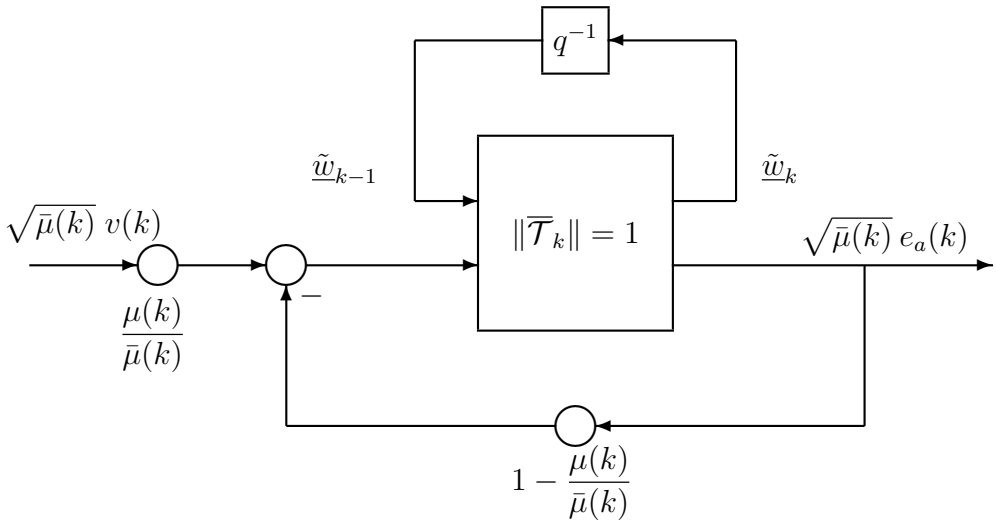

Figure 5.2: *Gradient type method as (lossless) in the forward path and lossy feedback path.*

new robustness statements. Let us consider again

$$\sum_{k=1}^{N} \bar{\mu}(k)|e_a(k)|^2 \leq \|\underline{\tilde{w}}_0\|_2^2 + \sum_{k=1}^{N} \bar{\mu}(k)|\bar{v}(k)|^2 \tag{5.32}$$

with $\bar{v}(k) = [\mu(k)/\bar{\mu}(k)]v(k) + [\mu(k)/\bar{\mu}(k) - 1]e_a(k)$. Let us consider

$$\delta(N) \triangleq \max_{1\leq k\leq N}\left|1 - \frac{\mu(k)}{\bar{\mu}(k)}\right|, \tag{5.33}$$

$$\gamma(N) \triangleq \max_{1\leq k\leq N}\frac{\mu(k)}{\bar{\mu}(k)}, \tag{5.34}$$

then we find with the triangular inequality:

$$\begin{aligned}
\sqrt{\sum_{k=1}^{N}\bar{\mu}(k)|e_a(k)|^2} &\leq \sqrt{\|\underline{\tilde{w}}_0\|_2^2}+\sqrt{\sum_{k=1}^{N}\bar{\mu}(k)|\bar{v}(k)|^2} && (5.35)\\
&\leq \sqrt{\|\underline{\tilde{w}}_0\|_2^2}+\sqrt{\sum_{k=1}^{N}\mu(k)|v(k)|^2}+\sqrt{\sum_{k=1}^{N}\bar{\mu}(k)\left[1-\frac{\mu(k)}{\bar{\mu}(k)}\right]^2|e_a(k)|^2}\\
&\leq \sqrt{\|\underline{\tilde{w}}_0\|_2^2}+\gamma(N)\sqrt{\sum_{k=1}^{N}\bar{\mu}(k)|v(k)|^2}\\
&\quad +\delta(N)\sqrt{\sum_{k=1}^{N}\bar{\mu}(k)|e_a(k)|^2} && (5.36)\\
&= \frac{1}{1-\delta(N)}\left[\sqrt{\|\underline{\tilde{w}}_0\|_2^2}+\gamma(N)\sqrt{\sum_{k=1}^{N}\bar{\mu}(k)|v(k)|^2}\right]. && (5.37)
\end{aligned}$$

As long as $\delta(N) < 1$, we can conclude in a global sense what we expected. The statements are summarized in the following theorem.

Theorem 5.4 (Extended Convergence of the Gradient Type Method) *For the gradient type method we find for* $0 < \mu(k)\|\underline{x}_k\|_2^2 < 2$, $\|\underline{\tilde{w}}_o\| < \infty$ *and* $\sum_{k=1}^{\infty}\bar{\mu}(k)|\bar{v}(k)|^2 < \infty$*:*

$$\bar{e}_a(k) \to 0. \tag{5.38}$$

Furthermore, if $\sqrt{\mu(k)}\underline{x}_k$ *is of persistent excitation (see Lemma 9.3), then we also have*

$$\underline{\hat{w}}_k \to \underline{w}. \tag{5.39}$$

Interesting to remark is the energy flow of such system. As the forward path is lossless, all energy that enters must also come out. The energy component in the parameter error vector is fed back into the input of the system. Energy can thus only be lost in the feedback path. The more energy is lost, the faster learns the algorithm. Thus, the fastest learning is obtained for $\mu(k) = \bar{\mu}(k)$ as the feedback becomes zero.

Exercise 5.5 *Show the stringent relation*

$$\frac{\|\underline{\tilde{w}}_k\|_2^2+\mu(k)\,|e_a(k)|^2}{\|\underline{\tilde{w}}_{k-1}\|_2^2+\mu(k)\,|v(k)|^2} \leq \frac{\mu(k)}{2\bar{\mu}(k)-\mu(k)} \tag{5.40}$$

for the third condition $\bar{\mu}(k) < \mu(k) < 2\bar{\mu}(k)$ in Lemma 5.1.

Exercise 5.6 *Derive the following relation:*

$$\sqrt{\sum_{k=1}^{N}\mu(k)|e_a(k)|^2} \leq \frac{\gamma^{\frac{1}{2}}(N)}{1-\delta(N)}\left[\sqrt{\|\underline{\tilde{w}}_0\|_2^2} + \gamma^{\frac{1}{2}}(N)\sqrt{\sum_{k=1}^{N}\mu(k)|v(k)|^2}\right]. \tag{5.41}$$

5.2.4 The Gauß-Newton Algorithm

The considerations for the gradient type algorithms can also be applied for a larger class of non-gradient algorithms, the so-called Gauß-Newton methods, of which the RLS algorithm is a special case. As the procedure is the same as in the previous sections, we only present the most important results and interpret them. First the definition:

Gauß-Newton Algorithm: Given the observations $\{d(k)\}_{k=1}^{N}$, an initial estimation $\underline{\hat{w}}_0$, and a positive-definite matrix Π_0. Then the recursive Gauß-Newton estimator with positive parameters $\{\lambda(i) \leq 1, \mu(i), \beta(i)\}$ is given by

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu(k)\, P_k \underline{x}_k^* \left(d(k) - \underline{x}_k^T \underline{\hat{w}}_{k-1}\right), \tag{5.42}$$

in which P_k satisfies the following Matrix-Riccati equation:

$$P_k = \frac{1}{\lambda(k)}\left(P_{k-1} - \frac{P_{k-1}\underline{x}_k^*\underline{x}_k^T P_{k-1}}{\frac{\lambda(k)}{\beta(k)} + \underline{x}_k^T P_{k-1}\underline{x}_k^*}\right), \quad P_0 = \Pi_0. \tag{5.43}$$

We recognize the RLS algorithm with exponential weighting that is obtained for $\mu(k) = \beta(k) = 1$ and $\lambda(k) = \lambda$.

From the local passivity relation we derive

$$\frac{\underline{\tilde{w}}_k^H P_k^{-1}\underline{\tilde{w}}_k + (\mu(k)-\beta(k))\,|e_a(k)|^2}{\lambda(k)\underline{\tilde{w}}_{k-1}^H P_{k-1}^{-1}\underline{\tilde{w}}_{k-1} + \mu(k)\,|v(k)|^2} \quad \begin{cases} \leq 1 & \text{for } 0 < \mu(k) < \bar{\mu}(k), \\ = 1 & \text{if } \mu(k) = \bar{\mu}(k), \\ \geq 1 & \text{for } \mu(k) > \bar{\mu}(k). \end{cases} \tag{5.44}$$

With the second property we can derive a feedback structure according to Figure 5.3.

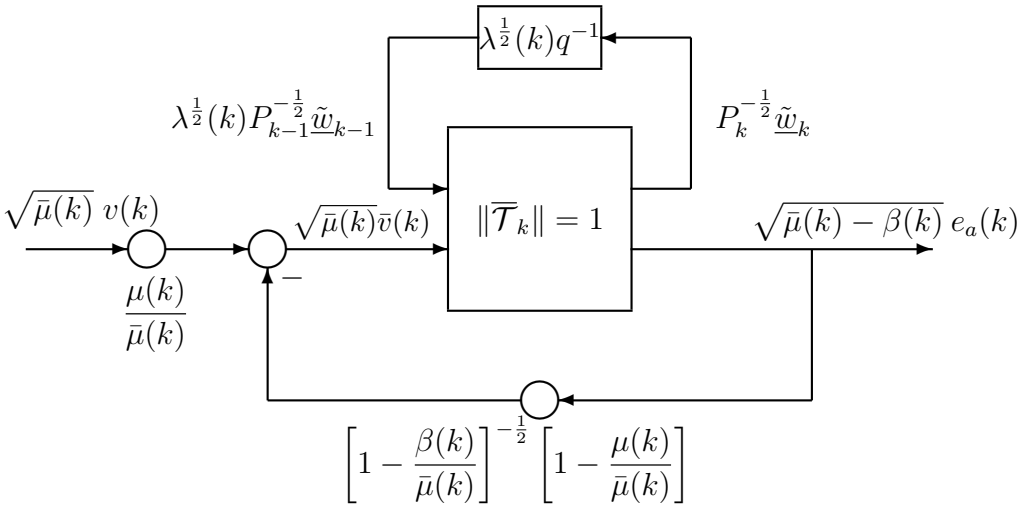

Figure 5.3: *Feedback structure of Gauß-Newton algorithm.*

Again a global relation can be derived

$$\begin{aligned}\sqrt{\sum_{k=1}^{N} \lambda^{[k+1,N]}[\bar{\mu}(k)-\beta(k)]\,|e_a(k)|^2} \leq \quad & \sqrt{\lambda^{[0,N]}\underline{\tilde{w}}_0^H P_0^{-1}\underline{\tilde{w}}_0} \quad + \sqrt{\sum_{k=1}^{N} \lambda^{[k+1,N]}\frac{\mu^2(k)}{\bar{\mu}(k)}|v(k)|^2} \\ & + \quad \sqrt{\sum_{k=1}^{N} \lambda^{[k+1,N]}\left|1-\frac{\mu(k)}{\bar{\mu}(k)}\right|^2 \bar{\mu}(k)|e_a(k)|^2} \\ \leq \frac{1}{1-\delta(N)} \quad & \left[\sqrt{\lambda^{[0,N]}\underline{\tilde{w}}_0^H P_0^{-1}\underline{\tilde{w}}_0} \quad + \gamma(N)\sqrt{\sum_{k=1}^{N} \lambda^{[k+1,N]}\bar{\mu}(k)|v(k)|^2}\right].\end{aligned} \tag{5.45}$$

We have employed the following short terms:

$$\delta(N) = \max_{1\leq k\leq N}\left|\frac{1-\frac{\mu(k)}{\bar{\mu}(k)}}{\sqrt{1-\frac{\beta(k)}{\bar{\mu}(k)}}}\right| \quad \text{und} \quad \gamma(N) = \max_{1\leq k\leq N}\ \frac{\mu(k)}{\bar{\mu}(k)}\ , \tag{5.46}$$

as well as

$$\lambda^{[i,j]} = \prod_{k=i}^{j}\lambda(k); \quad \bar{\mu}(k) = \frac{1}{\underline{x}_k^T P_k \underline{x}_k^*}. \tag{5.47}$$

Remarkably is that we always have

$$\bar{\mu}(k) \geq \beta(k). \tag{5.48}$$

Again, by substituting $\bar{\mu}(k)$ with $\mu(k)$ we find a new relationship and we obtain

$$\sqrt{\sum_{k=1}^{N} \lambda^{[k+1,N]}[\mu(k)-\beta(k)]\,|e_a(k)|^2} \qquad \leq$$

$$\leq \frac{\tilde{\gamma}^{1/2}(N)}{1-\delta(N)} \quad \left[\sqrt{\lambda^{[0,N]}\underline{\tilde{w}}_0^H P_0^{-1}\underline{\tilde{w}}_0} \quad +\gamma^{1/2}(N)\sqrt{\sum_{k=1}^{N}\lambda^{[k+1,N]}\mu(k)|v(k)|^2}\right]. \tag{5.49}$$

now with a modified short term:

$$\tilde{\gamma}(N) = \max_{1\leq k\leq N} \frac{\mu(k)-\beta(k)}{\bar{\mu}(k)-\beta(k)}\ , \tag{5.50}$$

Exercise 5.7 *Find the stability bound for the step-size $\mu(k)$ of the Gauß-Newton Algorithm. Consider both variants (5.45) and (5.49).*

Exercise 5.8 *For the particular case $P_0 = \epsilon I$, $\lambda(k) = \lambda$, $\mu(k) = \mu_o\bar{\mu}(k)$ and $\beta(k) = \beta_o\bar{\mu}(k)$ find the stability bound as well as the robustness measures. Compare the results to the gradient algorithm.*

5.3 Algorithms with Nonlinear Filter without Memory in the Estimation Path

The method described so far can also be modified in order to include nonlinearities in the estimation path. We can distinguish two variants that are being treated similarly in mathematical terms but have entirely different applications: neural networks for pattern classification and adaptive equalizers.

5.3.1 The Perceptron-Learning Algorithm

Let us first consider the neural network structure in its simplest form. Let us assume there are two sets of $M-$dimensional real-valued vectors $\underline{x}$, $\mathcal{S}_0$ and $\mathcal{S}_1$, characterized by having either the (exclusive) property A or B, thus

$$\mathcal{S}_0 = \{\underline{x} \in I\!R^M \mid \underline{x} \text{ is of property } A\}\ ,$$

$$\mathcal{S}_1 = \{\underline{x} \in I\!R^M \mid \underline{x} \text{ is of property } B\} \ .$$

If both sets are separable then a classification scheme (synapse, linear perceptron) can be employed to decide to which class a given vector $\underline{x}$ belongs to.

A linear synapse comprises of a linear combiner (as before) but has additionally a nonlinear device $f[z]$ at the output of the linear combiner as shown in Figure 5.4. Such nonlinear function is called activation function. Its value can also be interpreted as likelihood to which class a given vector $\underline{x}$ belongs to. A common choice of such activation functions $f[z]$

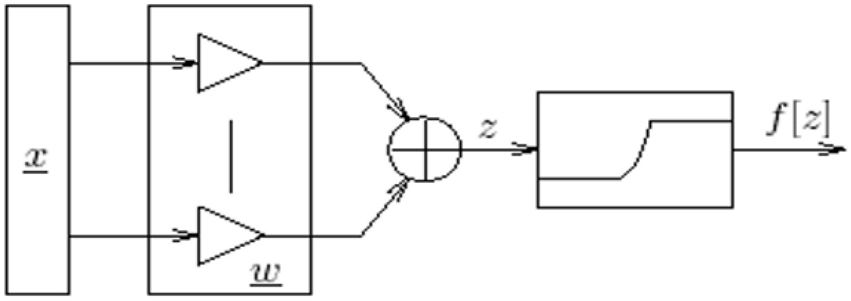

Figure 5.4: *The linear perceptron.*

are so-called Sigmoid-functions:

$$f_\beta[z] = \frac{1}{1+e^{-\beta z}} \ , \quad \beta > 0. \tag{5.51}$$

This is a monotone increasing function from 0 to 1 for inputs $z \in (-\infty, \infty)$. In its switching area that is around $z = 0$ it is more or less steep, depending on its parameter β. For $\beta \to \infty$ the sigmoid-function becomes a step-function (hard limiter), loosing its continuity property:

$$f_\infty[z] = \frac{1+\text{sgn}[z]}{2} = \begin{cases} 0 & \text{if } z < 0 \\ \frac{1}{2} & \text{if } z = 0 \\ 1 & \text{if } z > 0 \end{cases} \ .$$

Let us consider now a set of possible input vectors $\{\underline{x}_k\}$ with their corresponding correct output decisions $\{y(k)\}$. The values $\{y(k)\}$ belong to the range of the activating function $f[\cdot]$ that is there are unknown vectors $\underline{w}$, such that

$$y(k) = f[\underline{x}_k^T \underline{w}] \quad \text{for a particular } \underline{w}. \tag{5.52}$$

In a supervised learning data pairs $\{\underline{x}_k, y(k)\}$ are presented to the synapse so that the adaptation algorithm can estimate the unknown $\underline{w}$. Most well-known is the Perceptron

Learning Algorithm (PLA). The algorithm starts with an initial guess $\underline{w}_1$ and applies the following rule:

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu \underline{x}_k \left(y(k) - f[\underline{x}_k^T \underline{\hat{w}}_{k-1}] \right). \tag{5.53}$$

In order to keep the result more general we added also noise $v(k)$ in the reference. This can be interpreted as modeling error. The additively distorted reference values we denominate again by $\{d(k)\}$. We thus observe

$$d(k) = f[\underline{x}_k^T \underline{w}] + v(k) = y(k) + v(k). \tag{5.54}$$

We will analyze the following form of the PLA:

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu(k) \underline{x}_k \left(d(k) - f[\underline{x}_k^T \underline{\hat{w}}_{k-1}] \right), \tag{5.55}$$

for which we also included a variable step-size. The only difference compared to our previous method is the nonlinear mapping $f[\cdot]$, as it occurs in the estimation path. By the mean value theorem we have:

$$f[\underline{x}_k^T \underline{w}] - f[\underline{x}_k^T \underline{\hat{w}}_{k-1}] = f'[\eta(k)] e_a(k).$$

Figure 5.5 exhibits the feedback structure in this case. The nonlinear mapping occurs in the passive feedback path, resulting in a modified convergence condition. Writing down the equations for global convergence we obtain:

$$\delta(N) \stackrel{\Delta}{=} \max_{1 \le k \le N} \left| 1 - f'[\eta(k)] \frac{\mu(k)}{\bar{\mu}(k)} \right|, \tag{5.56}$$

which defines the condition for convergence and thus robustness: $\delta(N) < 1$.

5.3.2 Adaptive Equalizer Structures

Adaptive equalizers work very similarly to the PLA method, when training them by so-called training sequences. Such sequence is known at the receiver, typically sent at the beginning of a TDMA transmission. The receiver utilizes them in order to find the optimal set of coefficients. As the transmitted symbols are from a finite alphabet, a nonlinear mapping of so-called soft symbols to such symbols from the alphabet follows a linear filter. Analogue to the PLA we also have a concatenation of a linear adaptive filter with a fixed nonlinear mapping.

Very typical for the PLA is that the analysis is limited to real-valued signals and so is its typical application. The problem of the limitation is the definition of the nonlinear mapping for complex valued signals. As this is a requirement for adaptive equalizers, we will have to have a closer look at it now. Let us consider Figure 1.8 of the first chapter.

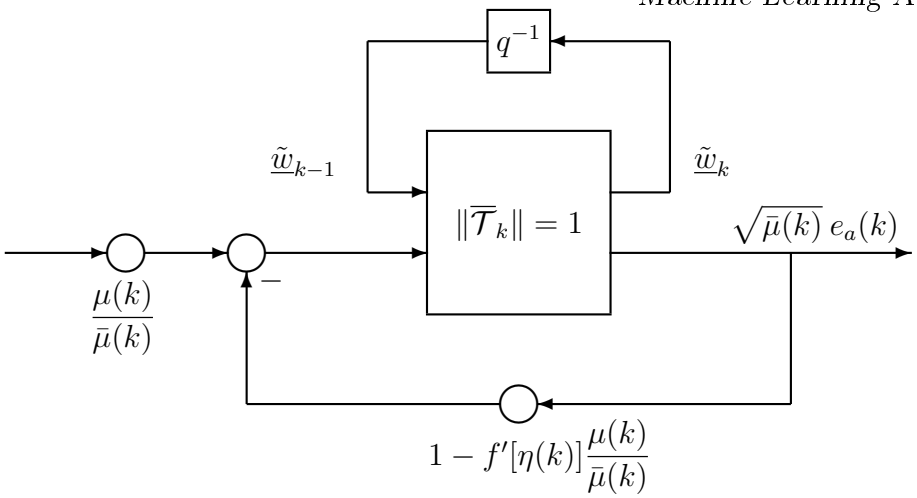

Figure 5.5: *The Perceptron-Learning Algorithm as feedback structure.*

A complex valued symbol $s(k)$ is transmitted through a linear channel c. Additive noise alters the received symbol further before it is sent through a linear adaptive filter. We do not expect that a pure linear filtering will recover the symbol $s(k-D)$ entirely (up to an unavoidable delay D). On the other hand only very particular symbols are being sent. We expect that after linear filtering the symbol value lies close to a valid symbol from the transmission alphabet and can thus be mapped onto the correct symbol by a suitable nonlinear mapping. We thus assume that this structure with a given filter length and optimal parameter set $\underline{w}_o$ to "equalize" the channel so that the output is close to the correct symbol. Our reference model thus delivers a value $y(k)$, which we can compare with the estimate $\hat{y}(k)$ conditioned by a parameter set $\underline{w}_{k-1}$. We call this a training modes as we are lacking a concrete reference signal.

Let us consider this situation more closely. The reference signal (assume $s(k-D)$, or a function of it) is given by $f[y(k)] = f[\underline{x}^T \underline{w}_o]$, by which we can compute the error signal $e_o(k) = f[y(k)] - f[\hat{y}(k)]$. In order to relate it to the a-priori error, we write:

$$\begin{aligned} e_o(k) &= f[y(k)] - f[\hat{y}(k)] && (5.57)\\ &= \frac{f[y(k)] - f[\hat{y}(k)]}{y(k) - \hat{y}(k)} e_a(k) && (5.58)\\ &\triangleq h[y(k), \hat{y}(k)] e_a(k). && (5.59) \end{aligned}$$

With help of this new function the update equations can be formulated in typical form:

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu(k)\underline{x}_k^* e_o(k) = \underline{\hat{w}}_{k-1} + \mu(k)\underline{x}_k^* h[y(k), \hat{y}(k)] e_a(k). \quad (5.60)$$

The difficulty is given with the function $h[\cdot,\cdot]$. In order to guarantee l_2-stability, we must have

$$\delta(N) \triangleq \max_{1\le k\le N} \left|1 - \frac{\mu(k)}{\bar{\mu}(k)} h[y(k), \hat{y}(k)]\right| < 1. \tag{5.61}$$

Example 7.1: Consider for example BPSK transmission. The excepted symbols are thus $[-1, 1]$. We select as nonlinear mapping $f[z] = \text{sgn}[z]$. Then we obtain

$$h[y(k), \hat{y}(k)] = \frac{\text{sgn}[y(k)] - \text{sgn}[\hat{y}(k)]}{y(k) - \hat{y}(k)}. \tag{5.62}$$

As negative values of $\text{sgn}[y(k)] - \text{sgn}[\hat{y}(k)]$ can only occur when the difference of the arguments is negative, the function itself is positive and thus a step-size exists for which stability is guaranteed.

In case there is no reference signals the algorithm can be driven in the so-called blind mode. As we have some a-priori information about the transmitted symbols, we can select the nonlinear mapping, so that a constant appears at the output if the filter has been selected correctly. We can for example employ so-called constant modulus (CM) signals, thus signals whose amplitude remains constant. The information is transmitted in its phase. Computing $|z(k)| = |\underline{x}_k^T \underline{w}_o|$, the optimal system will give out a constant amplitude, thus $|z(k)| = \gamma$. An adaptive algorithm can thus be:

CMA-q-2 Algorithm

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu(k)\underline{x}_k^* \hat{y}(k)[\gamma - |\hat{y}(k)|^q]. \tag{5.63}$$

Corresponding stability conditions are obtained from the function $h[\cdot,\cdot]$.

Exercise 5.9 *Compute the stability conditions of the step-size $\mu(k)$ in case of BPSK training and utilization of a sign function.*

Exercise 5.10 *CM-signals are being used for a transmission and the CMA-2-2 algorithm for training the equalizer. What is its stability condition?*

Exercise 5.11 *BPSK is being transmitted and the equalizer run in blind mode. Its updates are given by:*

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu(k)\underline{x}_k^* [sgn[\hat{y}(k)] - \hat{y}(k)]. \tag{5.64}$$

Draw the algorithm in its feedback structure and define its stability conditions.

5.3.3 Tracking of Equalizer Structures

Adaptive Equalizer structures do not fit into the system identification scheme and are thus often wrongly or at least overly simplified interpreted. Figure 5.6 intends to support a better understanding of the problem. Let us assume that a reference structure exists, that can guarantee the equalization and thus recovery of the symbols. During the training phase known signals are offered as *desired* signal. This is equivalent to the mode when the switch is selected on MMSE. In this mode the filter coefficients $\underline{\hat{w}}$ are selected so that the MSE is minimized, a requirement that can only be satisfied with some error term as the optimal solution requires a double infinite length equalizer. Let us denote the received signal by $r(k) = \underline{c}^T \underline{s}_k + v(k)$. The MMSE can be computed as

$$e_{MMSE}(k) = z(k) - \underline{r}_k^T \underline{\hat{w}}_{k-1} = \underbrace{s(k-D) - \underline{r}_k^T \underline{\hat{w}}_{k-1}}_{e_{NL}(k)} + \underbrace{z(k) - s(k-D)}_{-g(z)}, \tag{5.65}$$

where we have written the receiver values $r(k)$ in the vector $\underline{r}_k^T = [r(k), r(k-1), .., r(k-M+1)]$. In the blind mode of operation (also *decision directed mode*) the reference signal

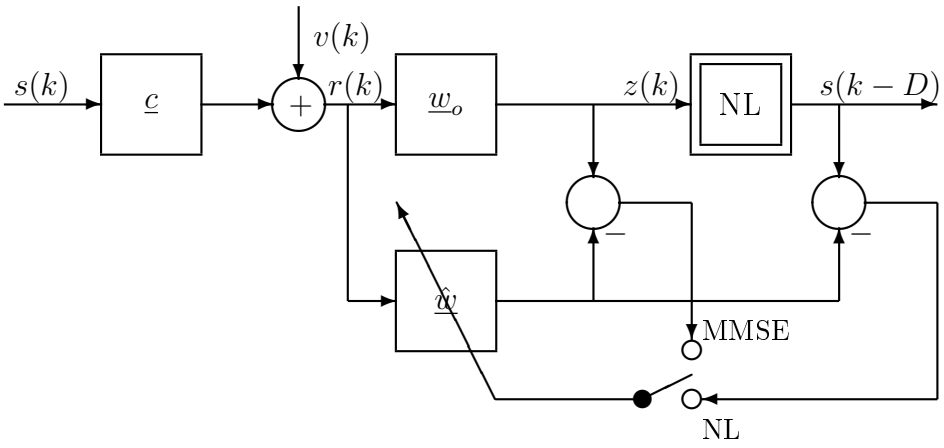

Figure 5.6: *Reference model for adaptive equalizers.*

is extracted out of the linear equalized signal by a nonlinear mapping. We thus have a nonlinear device in the reference path (Switch in position NL). The relation between the so obtained error signal $e_{NL}(k)$ and the MMSE is shown in Equation (5.65). However, it was assumed in (5.65) that only correct symbol $s(k-D)$ has been recovered. In the blind mode we do not have the correct values but only estimates of them. Equation (5.65) thus requires some correction:

$$e_{MMSE}(k) = \underbrace{\hat{s}(k-D) - \underline{r}_k^T \underline{\hat{w}}_{k-1}}_{\hat{e}_{NL}(k)} + \underbrace{z(k) - \hat{s}(k-D)}_{-\hat{g}(z)}. \tag{5.66}$$

The updates thus occur with an error signal $\hat{e}_{NL}(k) = e_{MMSE}(k) + \hat{g}[z(k)]$. Out of this we can conclude that:

- The adaptive filter of a nonlinear equalizer works in a system identification mode.
- An additive disturbance is present in form of the function $\hat{g}[z(k)]$.
- The excitation for the system identification is a composite signal, comprising of linearly filtered transmit symbols and additive noise.

Important for the equalizer is the tracking behavior of the algorithm. This can be described well by the feedback structure. Consider the update equation in the form:

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu(k)\underline{x}_k^* f[\hat{y}(k)]. \tag{5.67}$$

For the CMA-2-2 algorithm we obtain for example $f[\hat{y}(k)] = \hat{y}(k)[\gamma - |\hat{y}(k)|^2]$. Reformulating in the well-known form we obtain:

$$\bar{\mu}(k)|e_a(k)|^2 + \|\underline{\tilde{w}}_k\|^2 = \bar{\mu}(k)\left|e_a(k) - \frac{\mu(k)}{\bar{\mu}(k)} f[\hat{y}(k)]\right|^2 + \left\|\underline{\tilde{w}}_{k-1}\right\|^2 \tag{5.68}$$

If we consider the signals as random processes, we can compute the expectation on both ends. In steady-state we find that $E[\|\underline{\tilde{\mathbf{w}}}_k\|^2] = E[\left\|\underline{\tilde{\mathbf{w}}}_{k-1}\right\|^2]$ and thus we have

$$\mathrm{E}\left[\bar{\mu}(k)|\mathbf{e}_a(k)|^2\right] = \mathrm{E}\left[\bar{\mu}(k)\left|\mathbf{e}_a(k) - \frac{\mu(k)}{\bar{\mu}(k)} f[\hat{\mathbf{y}}(k)]\right|^2\right]. \tag{5.69}$$

This expression can be simplified based on the two following assumptions:

- We assume that in steady-state the transmitted signals $\mathbf{s}(k-D)$ and the undistorted a-priori error $\mathbf{e}_a(k)$ are statistically independent.
- We assume that in steady-state the reciprocal instantaneous energy $\bar{\mu}(k)$ and the estimated value $\hat{\mathbf{y}}(k)$ are statistically independent.

With these assumptions we can further process Equation (5.69). For small and constant step-sizes we obtain

$$\mathrm{E}[|\mathbf{e}_a(k)|^2] \approx \mu \frac{\mathrm{E}\left[|\mathbf{s}(k)|^2\gamma^2 - 2\gamma|\mathbf{s}(k)|^4 + |\mathbf{s}(k)|^6\right]}{2\mathrm{E}\left[\delta|\mathbf{s}(k)|^2 - \gamma\right]} \mathrm{E}\left[\|\underline{\mathbf{x}}_k\|_2^2\right] \tag{5.70}$$

with $\delta = 2$ for the complex-valued case and $\delta = 3$ for the real valued case. A few remarks to the procedure and the result (5.70).

- It is interesting that the steady-state error energy can now also be minimized with respect to γ and thus we can find the smallest steady-state error energy.

- Utilizing a gradient algorithm, the learning speed is strongly dependent on the eigenvalue spread of the input process. This spread is defined only by the channel, if we assume a white data sequence to be transmitted. On the other hand, the additive white noise works positively to decrease the eigenvalue spread.
- Generalizations towards Decision-Feedback Equalizers (DFE) and oversampled equalizers (Fractionally-Spaced Equalizer =FSE) are immediately possible. Note however that the correlation of the input signal becomes stronger and the learning thus slower [67].
- Solutions to this problem are certainly all mentioned methods to decorrelate and thus increasing learning speed, in particular polyphase filter structures [51, 52].
- The method does not require the existence of a feedback structure. The pure decomposition into in and output are sufficient.
- The method can also be applied in situations with nonlinear mapping in the error signal, an adaptive structure that will be analyzed in the next section.

Exercise 5.12 *Optimize with respect to γ for the CMA-1-2 algorithm.*

Exercise 5.13 *Compute the Excess-Mean-Square Error of the LMS algorithm based on the statistical method presented here and compare the results with those in Chapter 3. Compute also the steady-state error energy for the Least-Mean Fourth algorithm.*

Exercise 5.14 *Under the assumption of a reference model $\underline{\mathbf{w}}_k = \underline{\mathbf{w}}_{k-1} + \underline{\mathbf{q}}_k$ in which the vectors $\underline{\mathbf{q}}_k$ are statistically independent, compute the Excess-Mean-Square error in dependence to the step-size for the LMS algorithm.*

5.4 Adaptive Algorithms with Linear Filter in the Error Path

Adaptive Algorithms with linear filter in the error path (or inverse control) constitute their own class of adaptive algorithms. As it has many different applications this class is not even small. The simplest case is the LMS algorithm with delayed error (also-called Delayed Update LMS=DLMS Algorithm). If we implement the LMS algorithm in form of an ASIC, we quickly reach a learning rate limit, due to the recursive form of the algorithm. Improved speeds in digital realizations are often achieved by pipelining of the algorithmic partitions. This, however, is not possible for the LMS algorithm as we first have to compute the error before we can apply the update equation. Pipelining results in an error signal that is available only a few clock cycles later. If this is the case the update equation takes on the form:

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu(k)\underline{x}_k^* \tilde{e}_a(k-D), \tag{5.71}$$

in which the error occurs D cycles delayed. This is the simplest case of filtering.

In active noise control the error is constructed in the acoustic path and captured by a microphone. The path from the loudspeaker through the control unit is part of the linear filter path of the update error.

A further application of such filtered error path are adaptive IIR filter. Until now we only considered transversal filter structures (and linear combiners). If the reference model consists of an IIR filter, a large amount of coefficients in a transversal filter would need to be estimated (depending on the pole locations), while a corresponding IIR filter would require only few taps. This motivated already in the 70s many researchers to investigate adaptive IIR algorithms [83, 77, 19]. Their success, however, remained very small, the major problem being stability. The reason for this problem are again the occurrence of a linear filter in the error path as we will show next.

In adaptive IIR structures we have to distinguish between the so-called output error and the equation error. In the first form we apply estimated output values as input of the filter, thus the output of the estimator

$$\hat{y}(k) = \underline{u}_k^T \underline{\hat{w}}_{k-1} \tag{5.72}$$

itself is a part of the filter input

$$\underline{u}_k^T = [x(k), x(k-1), ..., x(k-M+1), \hat{y}(k-1), \hat{y}(k-2), ..., \hat{y}(k-N)]. \tag{5.73}$$

As opposed to the equation error method where noise outputs of the reference model are being employed:

$$\underline{u}_k^T = [x(k), x(k-1), ..., x(k-M+1), d(k-1), d(k-2), ..., d(k-N)]. \tag{5.74}$$

Applying the equation error we can straightforwardly write down the Wiener solution for random signals:

$$\mathrm{E}[\underline{\mathbf{u}}_k^* \underline{\mathbf{u}}_k^T] \underline{\hat{w}} = \mathrm{E}[\mathbf{d}_k^* \underline{\mathbf{u}}_k^T]. \tag{5.75}$$

As parts of $\mathbf{d}(k)$ are entries of the regression vector, there is more correlation than desired. Splitting the regression vector $\underline{\mathbf{u}}_k$ into two components $\underline{\mathbf{x}}_k$ and $\underline{\mathbf{d}}_k$, we obtain

$$\mathrm{E}\begin{bmatrix} R_{\underline{\mathbf{xx}}} & R_{\underline{\mathbf{dx}}} \\ R_{\underline{\mathbf{xd}}} & R_{\underline{\mathbf{dd}}} \end{bmatrix} \underline{\hat{w}} = \mathrm{E}\begin{bmatrix} \underline{r}_{\mathbf{d}\underline{\mathbf{x}}} \\ \underline{r}_{\mathbf{dd}} \end{bmatrix}. \tag{5.76}$$

Assuming white additive noise, we find $\underline{r}_{\mathbf{dd}} = \underline{r}_{\mathbf{yy}}$. On the left hand side of the equation we find a term $R_{\underline{\mathbf{dd}}} = R_{\underline{\mathbf{yy}}} + \sigma_{\mathbf{v}}^2 I$ that behaves differently than usual. Due to the noise it has an additional component. The so-obtained estimator is not bias free. For the output error

method on the other hand it is not expected that such a bias occurs. We will understand this better after a detailed analysis.

Until now it remains unclear why the output error method would belong to the algorithms with linear filter in the error path. To understand this we use the IIR reference model:

$$\underline{w}^T = [b(0), b(1), ..., b(M-1), a(1), a(2), ..., a(N)] = [\underline{b}^T, \underline{a}^T]. \tag{5.77}$$

with the IIR filter coefficients $b(0)..b(M-1)$ and $a(1)..a(N)$. For the undistorted a-priori output error we find

$$\begin{aligned}
e_o(k) &= \underline{x}_k^T\underline{b} + \underline{y}_k^T\underline{a} - \underline{x}_k^T\hat{\underline{b}}_{k-1} - \hat{\underline{y}}_k^T\hat{\underline{a}}_{k-1} && (5.78)\\
&= \underline{x}_k^T\underline{b} + \underline{y}_k^T\underline{a} - \underline{x}_k^T\hat{\underline{b}}_{k-1} - \hat{\underline{y}}_k^T\hat{\underline{a}}_{k-1} + \hat{\underline{y}}_k^T\underline{a} - \hat{\underline{y}}_k^T\underline{a} && (5.79)\\
&= [\underline{x}_k^T\underline{b} - \underline{x}_k^T\hat{\underline{b}}_{k-1}] + [\hat{\underline{y}}_k^T\underline{a} - \hat{\underline{y}}_k^T\hat{\underline{a}}_{k-1}] + [\underline{y}_k^T\underline{a} - \hat{\underline{y}}_k^T\underline{a}] && (5.80)\\
&= \underline{u}_k^T[\underline{w} - \hat{\underline{w}}_{k-1}] + [\underline{y}_k - \hat{\underline{y}}_k]^T\underline{a} && (5.81)\\
&= \underline{u}_k^T\tilde{\underline{w}}_{k-1} + \underline{e}_{o,k}^T\underline{a}. && (5.82)
\end{aligned}$$

The linear combination $\underline{e}_{o,k}^T\underline{a}$ can be interpreted as a linear filter operation:

$$\underline{e}_{o,k}^T\underline{a} = A[e_o(k)]. \tag{5.83}$$

With this the undistorted a-priori error becomes

$$e_o(k) = \frac{1}{1 - A(q^{-1})}[\underline{u}_k^T\tilde{\underline{w}}_{k-1}] \tag{5.84}$$

and the update equation can be reformulated to

$$\begin{aligned}
\hat{\underline{w}}_k &= \hat{\underline{w}}_{k-1} + \mu(k)\underline{u}_k^*[v(k) + e_o(k)] && (5.85)\\
&= \hat{\underline{w}}_{k-1} + \mu(k)\underline{u}_k^*\left[v(k) + \frac{1}{1 - A(q^{-1})}[e_a(k)]\right]. && (5.86)
\end{aligned}$$

We recognize a particular linear filtering in the error path. In case of a constant step-size $\mu(k) = \mu$ the algorithm is called Feintuch-algorithm [19]. These considerations are of course not restricted to simple gradient methods. Methods in the class of Gauß-Newton can be treated equally. Utilizing an output error with the Gauß Newton type algorithm, the corresponding algorithm is called pseudo linear regression algorithm (PLR).

We thus deal in general with a filter structure like:

$$\hat{\underline{w}}_k = \hat{\underline{w}}_{k-1} + \mu(k)\underline{x}_k^* F\left[v(k) + e_a(k)\right]. \tag{5.87}$$

Figure 5.7 illustrates the location of the filter function $F(q^{-1})$.

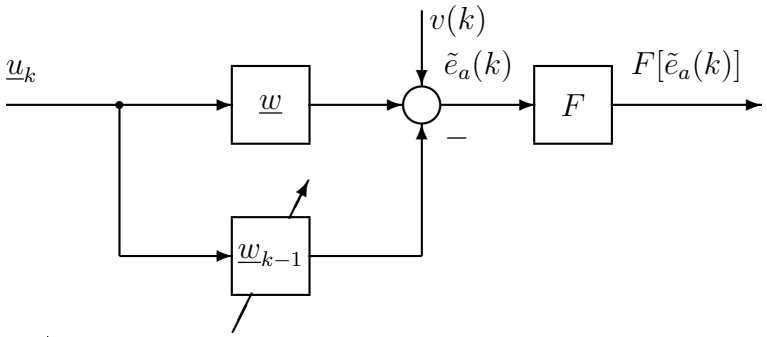

Figure 5.7: *Adaptive algorithm structure with linear filter in the error path.*

If $v(k)$ and $e_a(k)$ experiences a different filter, a joint filtering process can for example be obtained by a proper substitution (for example $v(k) = F^{-1}[v'(k)]$). The update equation (5.87) can be reformulated into

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \bar{\mu}(k)\underline{x}_k^* \left[e_a(k) + \underbrace{\left(\frac{\mu(k)}{\bar{\mu}(k)} F[v(k)] + \frac{\mu(k)}{\bar{\mu}(k)} F[e_a(k)] - e_a(k) \right)}_{\bar{v}(k)} \right]. \tag{5.88}$$

In Figure 5.8 the feedback structure is illustrated.

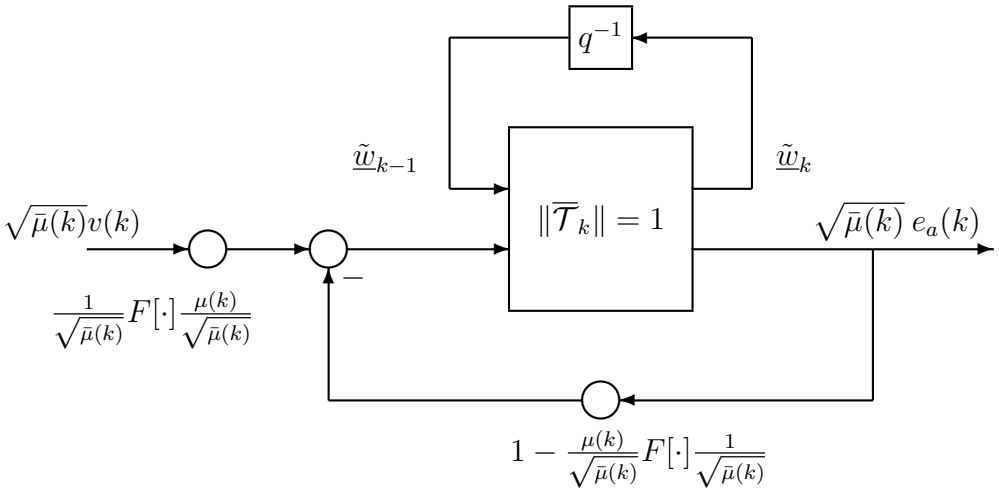

Figure 5.8: *Gradient type algorithm with linear filter in the error path in feedback structure.*

If the error path $F[\cdot]$ is given in matrix form, for example here with three coefficients

f_0, f_1, f_2:

$$F_N = \begin{bmatrix} f_0 & & & & \\ f_1 & f_0 & & & \\ f_2 & f_1 & f_0 & & \\ & f_2 & f_1 & f_0 & \\ & & \ddots & \ddots & \ddots \end{bmatrix} \tag{5.89}$$

and furthermore step-sizes in diagonal matrix form $\bar{M}_k$ and M_k, then the abbreviations are

$$\delta(N) \triangleq \|I - \bar{M}_N^{-\frac{1}{2}} M_N F_N \bar{M}_N^{-\frac{1}{2}}\|_{2,ind} \tag{5.90}$$

$$\gamma(N) \triangleq \|\bar{M}_N^{-\frac{1}{2}} M_N F_N \bar{M}_N^{-\frac{1}{2}}\|_{2,ind}. \tag{5.91}$$

With such definitions it is possible to derive robustness conditions also for the case of linearly filtered errors.

Theorem 5.5 *For the gradient method with linearly filtered error we find l_2-stability under the following definitions (5.90) and (5.91):*

$$\sqrt{\sum_{k=1}^{N} \bar{\mu}(k)|e_a(k)|^2} \leq \frac{1}{1-\delta(N)} \left[\|\underline{\tilde{w}}_0\|_2 + \gamma(N) \sqrt{\sum_{k=1}^{N} \bar{\mu}(k)|v(k)|^2} \right], \tag{5.92}$$

$$\sqrt{\sum_{k=1}^{N} \mu(k)|e_a(k)|^2} \leq \frac{\gamma^{\frac{1}{2}}(N)}{1-\delta(N)} \left[\|\underline{\tilde{w}}_0\|_2 + \gamma^{\frac{1}{2}}(N) \sqrt{\sum_{k=1}^{N} \mu(k)|v(k)|^2} \right]. \tag{5.93}$$

The proof follows the previous procedure (see also Exercise 7.11). However, the stability condition is much harder to check with the abbreviations in (5.90)

$$\|I - \bar{M}_N^{-\frac{1}{2}} M_N F_N \bar{M}_N^{-\frac{1}{2}}\|_{2,ind} < 1, \tag{5.94}$$

as we have to deal with time variant components in matrices. For relatively large filters M we can claim approximately that $\bar{\mu}(k) = 1/\|\underline{x}_k\|_2^2 \approx M\sigma_{\mathbf{x}}^2$. With constant step-size $\mu(k) = \mu_o$ the following relation is true:

$$\max_{\Omega} \left| 1 - \frac{\mu_o}{M\sigma_{\mathbf{x}}^2} F(e^{j\Omega}) \right| < 1. \tag{5.95}$$

The step-size μ_o, providing the fastest convergence, can be found by the following minimax optimization:

$$\mu_{opt} = \arg \min_{\mu_o} \max_{\Omega} \left| 1 - \frac{\mu_o}{M\sigma_{\mathbf{x}}^2} F(e^{j\Omega}) \right|. \tag{5.96}$$

From the stability condition (5.95) one can recognize that it is not necessarily satisfied for all positive step-sizes. The linear function $F[\cdot]$ can exhibit a negative real value for various frequencies. In this case the algorithm behaves unstable even for small step-sizes (assuming excitation at these frequencies). The necessary condition for $F[\cdot]$ is known in the literature as strict positive real (SPR):

$$\text{Real}\left(F(e^{j\Omega})\right) > 0; \text{ for all } \Omega. \tag{5.97}$$

Exercise 5.15 *Prove Theorem 5.5.*

Exercise 5.16 *Extend the proof to include the Gauß-Newton algorithm with linearly filtered error path.*

Exercise 5.17 *An adaptive IIR filter with two feedback coefficients $a(1)$ and $a(2)$ is to be adapted by the Feintuch algorithm. Which condition must the two coefficients satisfy, so that a stable algorithm s obtained? What is its optimal step-size?*

Exercise 5.18 *Let an undisturbed, nonlinear system be: $y(k) = a_k^T y_k + \underline{b}_1^T \underline{x}_k + \underline{b}_2^T \underline{xx}_k$ with $\underline{xx}_k^T = [x(k)x(k), x(k)x(k-1), .., x(k)x(k-M_2+1)]$. Derive stability conditions for the corresponding gradient method with output error.*

Exercise 5.19 *A compromise could be to derive an adaptive algorithm whose update error is partially output error and partially equation error. Derive such n algorithm and find its stability conditions.*

Exercise 5.20 *Steiglitz and McBride had the idea [79], that the adaptive IIR filter can be improved by a pre-filtering in the update error of the Feintuch algorithm by $1 - \hat{A}(q^{-1})$. Derive the stability condition in this case.*

Exercise 5.21 *The neural network by Narendra and Parthasarathy (see Figure 1.10) can be interpreted as adaptive IIR filter with nonlinearity in the estimation path. Derive a gradient algorithm for the training of such network and find the required stability conditions.*

Matlab Exercise 5.1 Write a Matlab programme to identify the following system:

$$y(k) = a(1)y(k-1) + a(2)y(k-2) + x(k) + x(k-1) + x(k-2).$$

Let the system be disturbed additively with white Gaussian noise of $\sigma_v^2 = 0.01$. Utilize an equation error as well as an output error and compare the result. Use the three different sets

$$\begin{aligned} a(1) &= -1.6, a(2) = -0.8; \\ a(1) &= -0.8, a(2) = -0.9; \\ a(1) &= 0.4, a(2) = -0.2. \end{aligned}$$

For excitation use a) white noise and b)

$$x(k) = \sum_{i=1}^{5} \sin(2\pi f_i k)$$

with the frequencies $f_1 = 0.1, f_2 = 0.17, f_3 = 0.25, f_4 = 0.35, f_5 = 0.4$. Find the step-sizes for fastest convergence and the stability bound.

5.5 Literature

Good tutorials and introductions to robust control are found in [14, 41]. An introduction into adaptive, robust filtering can be found in [46]. The small-gain-theorem is well explained in [42, 82]. First publications for robustness of LMS algorithms are in [29, 30]. In [73] the minimax explanation of the LMS algorithm can be found. In [61, 62] the feedback structure of gradient type algorithms as well as the Gauß-Newton method is explained. In [31, 37, 43] are good tutorials for neural networks. In [60] the explanation to the Feintuch algorithm is found. The DLMS algorithm has been treated classically in [44, 45] and [58]. Further details to tracking of equalizers are in [47]. Details to the FXLMS algorithm can be found in [68].

Chapter 6

Optimization Problems

6.1 Reformulations of the optimization problem for classification

Let us return to the original problem formulation for binary classification that we solved by LS. We were interested in minimizing the following cost function

$$\underline{w}_{LS} = \min_{\underline{w}} \sum_{\underline{\hat{x}}_i \in \mathcal{S}_1} \left(1 - \underline{\hat{w}}^H \underline{\hat{x}}_i\right)^2 + \sum_{\underline{\hat{x}}_i \in \mathcal{S}_2} \left(-1 - \underline{\hat{w}}^H \underline{\hat{x}}_i\right)^2 .$$

However, we understand that the solution may not be unique and an additional constraint on $\underline{\hat{w}}$ maybe wise. For example, it maybe good to select the solution with smallest length which can be formulated by

$$\underline{w}_{opt_1} = \min_{\underline{w}} \sum_{\underline{\hat{x}}_i \in \mathcal{S}_1} \left(1 - \underline{\hat{w}}^H \underline{\hat{x}}_i\right)^2 + \sum_{\underline{\hat{x}}_i \in \mathcal{S}_2} \left(-1 - \underline{\hat{w}}^H \underline{\hat{x}}_i\right)^2 + \lambda ||\underline{\hat{w}}||^2. \tag{6.1}$$

An additional complication may arise if also a non-linear activation function $\sigma(x)$ is being applied:

$$\underline{w}_{opt_2} = \min_{\underline{w}} \sum_{\underline{\hat{x}}_i \in \mathcal{S}_1} \left(1 - \sigma(\underline{\hat{w}}^H \underline{\hat{x}}_i)\right)^2 + \sum_{\underline{\hat{x}}_i \in \mathcal{S}_2} \left(-1 - \sigma(\underline{\hat{w}}^H \underline{\hat{x}}_i)\right)^2 + \lambda ||\underline{\hat{w}}||^2.$$

Such non-linear terms often generate saddle points through which gradient approaches have difficulties to find their way. With the additional term these saddle points are often deformed into a continuous monotone region which can be successfully searched through with a gradient approach. The additional term can be interpreted in different ways

1. additive weight with $\lambda > 0$
2. Lagrangian Multiplier
3. reformulated problem with side constraints

6.1.1 Interpretation 1: additive weight

In the LS solution we compute the Gramian

$$R = \sum_{\underline{\hat{x}}_i \in \mathcal{S}} \underline{\hat{x}}_i \underline{\hat{x}}_i^H.$$

Proceeding with an additional term, we now obtain a slightly different Gramian

$$R = \lambda I + \sum_{\underline{\hat{x}}_i \in \mathcal{S}} \underline{\hat{x}}_i \underline{\hat{x}}_i^H.$$

A small but positive value of λ can ensure the positive definiteness of the Gramian. In this case we call this a regularisation parameter. If, on the other hand, λ becomes very large, then the Gramian is negligible and the identity matrix dominates. This problem may be far off from what we are interested in. In the context of regression, adding such a regularisation parts is named ridge regression.

If we design an optimal gradient approach for such a cost function, we find a so-called leaky-LMS with updates:

$$\underline{\hat{w}}_k = (1 - \lambda)\underline{\hat{w}}_{k-1} + \mu \underline{x}_i (y_i - \underline{\hat{w}}_{k-1} \underline{x}_i)$$

which has a forgetting perspective in its solution. Such algorithm is well suited for tracking, as long as the movement of the data is relatively low.

6.1.2 Interpretation 2: Lagrangian Multiplier

In our second interpretation we treat the additional constraint as a Lagrangian multiplier. In this case the side constraint needs to be satisfied. Computing the derivative of cost function (6.1, we find

$$(R + \lambda I)\underline{\hat{w}} = \underline{p}$$

with the abbreviations $R = \sum \underline{\hat{x}}_i \underline{\hat{x}}_i^H$ and $\underline{p} = \sum_{\underline{\hat{x}}_i \in \mathcal{S}_1} \underline{\hat{x}}_i - \sum_{\underline{\hat{x}}_i \in \mathcal{S}_2} \underline{\hat{x}}_i$. Substituting this into the original formulation ends with

$$\min_{\lambda} \sum_{\underline{\hat{x}}_i \in \mathcal{S}_1} \left(1 - \underline{p}^H (R + \lambda_I)^{-1} \underline{\hat{x}}_i\right)^2 + \sum_{\underline{\hat{x}}_i \in \mathcal{S}_2} \left(-1 - \underline{p}^H (R + \lambda_I)^{-1} \underline{\hat{x}}_i\right)^2 + \lambda \underline{p}^H (R + \lambda_I)^{-2} \underline{p}.$$

Although nonlinear in λ we have now a nonlinear problem in only one variable that can be solved by a gradient approach.

6.1.3 Interpretation 3: Reformulated Problem

As our objective is to minimize $||\underline{\hat{w}}||$ we can also formulate our problem as

$$\begin{aligned} &\min_{\underline{\hat{w}}} ||\underline{\hat{w}}||_2 \\ &\text{subject to} \\ &\underline{\hat{w}}^H \underline{\hat{x}}_i = y_i; \quad i = 1, 2, ... \end{aligned}$$

which can be brought into an equivalent Lagrangian optimization problem

$$\min_{\underline{\hat{w}}} \frac{1}{2}||\underline{\hat{w}}||^2 + \sum \lambda_i \left(\underline{\hat{w}}^H \underline{\hat{x}}_i - y_i\right). \tag{6.2}$$

Differentiating with respect to $\underline{\hat{w}}$ delivers

$$\underline{\hat{w}} = -\sum \lambda_i \underline{\hat{x}}_i$$

which can be substituted in (6.2) and we obtain a formulation only in λ_i:

$$\min_{\lambda_i} \frac{1}{2} \left\| \sum \lambda_i \underline{\hat{x}}_i \right\|^2 + \sum \lambda_i y_i.$$

This is quadratic in λ_i and can be solved explicitly. We pack all λ_i into a vector $\underline{\lambda}$ and find

$$\min_{\lambda_i} \frac{1}{2} \underline{\lambda}^H X^H X \underline{\lambda} + \underline{\lambda}^H \underline{y}$$

where we used short notations for $X = [\underline{\hat{x}}_1, \underline{\hat{x}}_2, ..., \underline{\hat{x}}_N]$ and $\underline{y} = [y_1, y_2, ..., y_N]^T$. The solution of this is thus $\underline{\lambda} = -(X^H X)^{-1} \underline{y}$, which we can now plug into the original form and obtain:

$$\min_{\underline{\hat{w}}} \frac{1}{2}||\underline{\hat{w}}||^2 + \underline{\lambda}^H (X\underline{\hat{w}} - \underline{y}) = \min_{\underline{\hat{w}}} \frac{1}{2}||\underline{\hat{w}}||^2 - \underline{y}^H (X^H X)^{-1} (X\underline{\hat{w}} - \underline{y}),$$

which is quadratic in $\underline{\hat{w}}$ and thus can be solved directly:

$$\underline{\hat{w}}_{opt} = X^H (X^H X)^{-1} \underline{y}.$$

We recognize this as the underdetermined LS solution of a LS problem. However, in our setup we expect the amount of data to be much larger than the number of parameters to estimate. Such a solution is thus not useful.

Example 6.2 (Ridge Regression) *We now consider the case of an underdetermined situation. Our task is to find the best fitting coefficients* $c_k; k = 0, 1, ..., m$ *for the polynomial*

$$\hat{y}_i = c_0 + c_1 x_i + c_2 x_i^2 + ... + c_m x^m; i = 1, 2, ..., n.$$

As long as $n < m$ *we have an underdetermined problem. In Figure 6.1 we can observe the behavior of a classifier on a set of* $n = 5$ *points. If the order of the polynomial is small, in this example* $m = 3$*, we have an overdetermined problem and LS delivers a good solution. A ridge regression delivers a less good solution but is closer to a straight line. If we now increase the order of our polynomial to* $m = 7$*, we obtain the solution on the right hand side. For LS we now have to take the underdetermined LS form and we recognize a rather large oscillation behavior between the five points. However, the ridge solution now ensures a relatively smooth fitting that typically is desired*[1]*.*

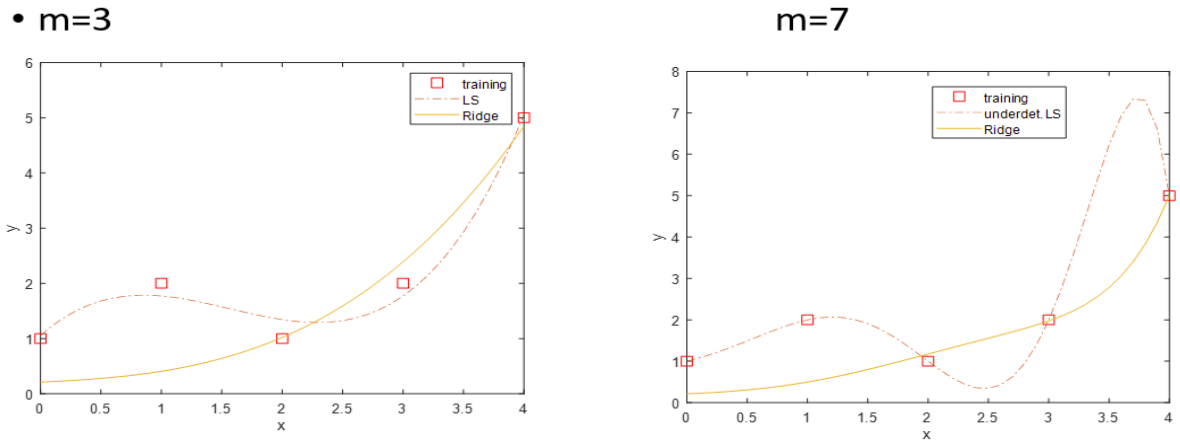

Figure 6.1: *An Ridge regression: left with polynomial order m=3, right with m=7.*

6.2 Solving SVMs

It took Vladimir Vapnic to come up with a formulation that finally turned out to be the right one, beginning of the 90ies. For this to understand we return to the problem formulation of maximizing the width of a street in between the two classes we like to discriminate, see Figure 4.6. In Equation (4.2) we learned that the maximal street width is given by

[1] In literature a ridge regression is viewed as a special case of a so called Tikhonov regularization, named after the Russian mathematician Andrej Tikhonov.

$2/||\underline{w}||$, where $\underline{w}$ stands for the weight vector without the bias term. Vapnik combined the formulation of the two classes $\mathcal{S}_1$ with $y_i = +1$ and $\mathcal{S}_2$ with $y_i = -1$ to a compact form

$$y_i(\underline{w}^H \underline{x}_i + b) \geq 1; i = 1, 2, ..., N \tag{6.3}$$

as we have seen already in (4.1). Here we write the bias term b explicitly. This allows us to reformulate our problem in maximizing the street width, or equivalently minimizing $||\underline{w}||$ with the side constraints of (6.3):

$$\min_{\underline{w},b} \frac{1}{2}||\underline{w}||^2 - \sum_{i=1}^{N} \lambda_i y_i(\underline{w}^H \underline{x}_i + b - 1) \tag{6.4}$$

which is nothing else but the maximization of the street width with Lagrangian side constraints to satisfy the classification conditions. Differentiating with respect to $\underline{\hat{w}}$ and b delivers two conditions:

$$\underline{w}_{opt} = \sum_{i=1}^{N} \lambda_i y_i \underline{x}_i \tag{6.5}$$

$$0 = \sum_{i=0}^{N} \lambda_i y_i. \tag{6.6}$$

Plugging in these two in the original formulation (6.4) provides us with

$$-\frac{1}{2}\left\|\sum_{i=1}^{N} \lambda_i y_i \underline{x}_i\right\|^2 = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j \mathbf{y}_i \mathbf{y}_j \left\langle \underline{x}_i, \underline{x}_j \right\rangle, \tag{6.7}$$

a formulation that is quadratic in λ_i and can thus be solved relatively easily.We also recognize that the vectors $\underline{x}_i$ only appear in form of an inner product.

The solution of

$$\min_{\lambda_i} -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j \mathbf{y}_i \mathbf{y}_j \left\langle \underline{x}_i, \underline{x}_j \right\rangle$$

is equivalent of solving the so-called dual problem:

$$\max_{\lambda_j} \sum_{j=1}^{N} \lambda_j - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j \mathbf{y}_i \mathbf{y}_j \left\langle \underline{x}_i, \underline{x}_j \right\rangle.$$

Here only the support vectors have $\lambda_j = 0$, all others are simply zero. We thus obtain a simple computation rule: first select the support vectors and compute (6.5) and finally

solve the dual problem. We will show a simple example to illustrate this next.

Suppose we have two classes $\mathcal{S}_1$ with a marginal vector $\underline{f}_1$ and $\mathcal{S}_2$ with a support vector $\underline{g}_2$. In this case only two Lagrangian variables are non-zero, say λ_1 and λ_2. As the problem is symmetric, we find $\lambda_1 = \lambda_2$ and we need to maximize the dual problem

$$\max_{\lambda_1} 2\lambda_1 - \frac{1}{2}\lambda_1^2 ||\underline{f}_1 - \underline{g}_2||^2.$$

We immediately find the optimal Lagrangian to be

$$\lambda_1 = \lambda_2 = \frac{2}{||\underline{f}_1 - \underline{g}_2||^2}$$

and

$$\underline{w} = \frac{2}{||\underline{f}_1 - \underline{g}_2||^2}(\underline{f}_1 - \underline{g}_2).$$

We finally find the bias term b from the margin condition $\underline{w}^T \underline{f}_1 + b = 1$

$$b = 1 - \frac{2}{||\underline{f}_1 - \underline{g}_2||}(\underline{f}_1 - \underline{g}_2)^T \underline{f}_1.$$

6.3 Increasing Dimensions: Kernels

I r
t e

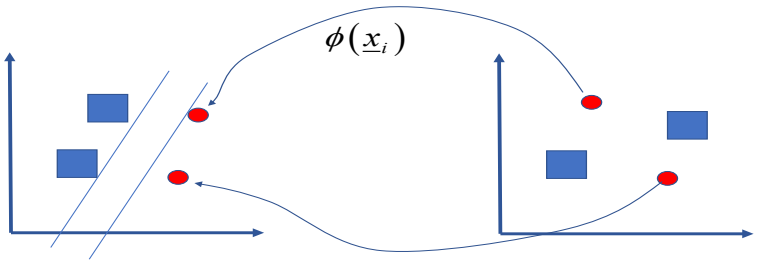

Figure 6.2: *Left: linearly separable, right: XOR distribution of elements, thus not linearly separable.*

such problem we essentially have two options:

1. increasing the dimensions of the feature space
2. nonlinear mapping of the feature vectors

Figure 6.2 already indicates that a nonlinear mapping $\phi(\underline{x}_i)$ can lead to a solution. In this case we increase the dimension of the feature vector. For example, a vector

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

is mapped to

$$\phi(\underline{x}) = \begin{bmatrix} \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \sqrt{2}x_1x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}.$$

Recall that for the optimization problem only the inner vector products count. If we now use a nonlinear mapping $\phi(\underline{x}_1)$ and $\phi(\underline{x}_2)$, we obtain a polynomial kernel

$$\langle \phi(\underline{x}_1), \phi(\underline{x}_2) \rangle = (1 + \langle \underline{x}_1, \underline{x}_2 \rangle)^2 - 1$$

This idea was introduced 1992 by Vapnik and solved many nonlinear problems. The problem now is to find the right kernel. Table 6.1 shows some well-known kernels. Alternatively,

Name	Kernel description
Neuronal Network	$K_0(\underline{x}, \underline{y}) = \tanh(\beta^2 \underline{x}^H \underline{y} + \alpha)$
Polynomial Kernel of degree n	$K_1(\underline{x}, \underline{y}) = (\underline{x}^H \underline{y} + 1)^n - 1$
Radial Basis Function	$K_2(\underline{x}, \underline{y}) = e^{-\beta \|\underline{x} - \underline{y}\|^2}$
Fourier Kernel	$K_3(\underline{x}, \underline{y}) = \prod_{n=1}^{N} \frac{\sin((2D+1)\pi(\mathbf{x}_n - y_n))}{\sin(\pi(x_n - y_n))} - 1$

Table 6.1: Typical Kernels.

such a solution can also be obtained if more features are available and the feature vectors can be enriched. The non-linear kernels are thus a systematic way to enrich the feature vectors without obtaining new features. Figure 6.3 depicts how the XOR problem discussed before can be solved by adding one more dimension in the feature space. Now a simply hyperplane (shown in yellow) can discriminate the two sets.

Figure 6.3: *Left: linearly separable XOR problem by lifting the points in z direction, right: XOR distribution of elements, thus not linearly separable.*

Chapter 7

Neural Networks

In this chapter we will extend the single layer perceptron as we have studied it so far into multiple layers. To this end we first have to learn how to combine intermediate results (sub-classifications) into final more complex results. Consider for example the problem in Figure 7.1. On the right hand side of the figure just a few red spots were added, making the problem no longer linearly separable. However, we can consider two sub-classes, which properly combined would result in the desired result. In principle two perceptrons in parallel and one that properly combines their output could do this job. In the following Figure 7.2

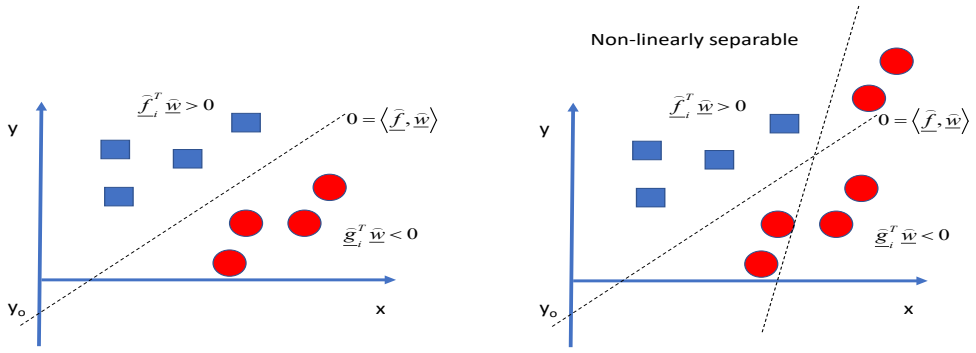

Figure 7.1: *Left: linearly separable problem, right: not linearly separable but two sub-classes possible.*

the problem is even more complex as now the desired class with the red spots is surrounded by blue squares. Here, three linear classifiers properly combined can define the solution. Different to the previous case, we now have to add also a half plane towards the bottom

while the other two have to reach to the top. What we need to establish is and "and" operation to combine sets.

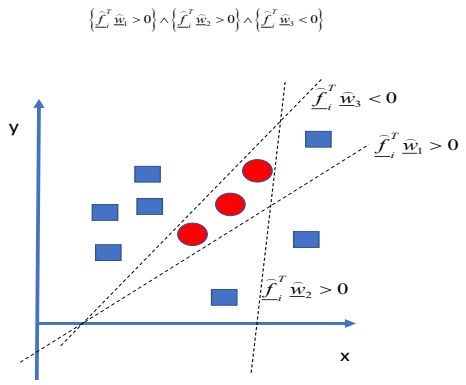

Figure 7.2: *non-linear separable problem with three sub-classes.*

Figure 7.3 depicts the XOR problem that was already discussed. It can be divided into two sub-classes that needs to be combined. Let's analyze the solution in more detail. To this end we separate the feature vectors belonging to class 1 into $\underline{\hat{f}}_{1,i}$ and $\underline{\hat{f}}_{2,i}$. Consider first the upper line for which we need to have $\underline{\hat{f}}_{1,i}^T \underline{\hat{w}}_1 > 0$.

$$\begin{array}{ccc} [1, x_{1,i}, y_{1,i}]^T [w_{1,0}, w_{1,1}, w_{1,2}] & > 0 & \\ y_{1,i} = & & -\frac{w_{1,0}}{w_{1,2}} - \frac{w_{1,1}}{w_{1,2}} x_{1,i} \\ \text{if} \quad w_{1,2} > 0 \rightarrow & & w_{1,0}, w_{1,1} < 0. \end{array}$$

Similarly, we find for the lower line $\underline{\hat{f}}_{2,i}^T \underline{\hat{w}}_1 < 0$.

$$\begin{array}{ccc} [1, x_{2,i}, y_{2,i}]^T [w_{2,0}, w_{2,1}, w_{2,2}] & > 0 & \\ y_{2,i} = & & -\frac{w_{2,0}}{w_{2,2}} - \frac{w_{2,1}}{w_{2,2}} x_{2,i} \\ \text{if} \quad w_{2,2} > 0 \rightarrow & & w_{2,0} > 0, w_{2,1} < 0. \end{array}$$

The required connection in terms of a two-layer network is depicted in Figure 7.4. Obviously, what we need to combine the two classes is something that works as a "logical and". This can be achieved by the following sigmoid function

$$z = \frac{1}{1 + e^{-10(w_{3,2} z_2 + w_{3,1} z_1 - 1.5)}}.$$

The bias term -1.5 is selected so that only a positive value is achieved if both outputs are sufficiently positive. As $w_{3,2}$ is assumed to be negative we combine a positive z_1 with a negative z_1.

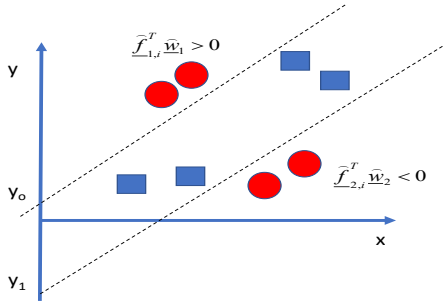

Figure 7.3: *The XOR problem divided into two sub-classes.*

Here, the question arises of how many layers are required to systematically discriminate certain islands of data. See for example the distribution of data in Figure 7.5. We need to have a third layer which allows to combine islands that the second layer defines. General belief of the 90ies was that three layer neurons are sufficient as all shapes can be formed by them. The later part of this sentence can be proven. All inner layers that are not input or output layers are called hidden layers. A three layer network has one hidden layer.

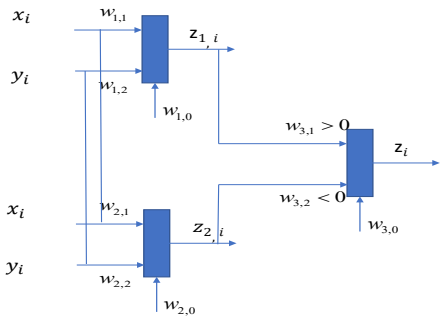

Figure 7.4: *Combining the two sub-classes in a two-layer network to solve the XOR problem.*

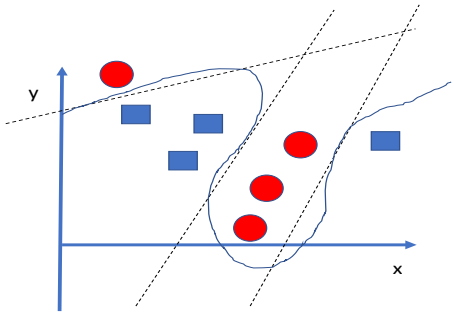

Figure 7.5: *Even a complex distribution of data in a class suggest that two to three layers are sufficient.*

7.1 Two Layer Perceptron

We thus need to extend the single layer learning algorithm to two layers. We recall the update of the PLA here for the second layer

$$\underline{\hat{w}}_{2,i} = \underline{\hat{w}}_{2,i-1} + \mu \underline{\tilde{x}}_i^* (y_i - f(\underline{\tilde{x}}_i^T \underline{\hat{w}}_{2,i-1})) f'(\underline{\tilde{x}}_i^T \underline{\hat{w}}_{2,i-1}),$$

where the input vector $\underline{\tilde{x}}_i$ comprises of

$$\underline{\tilde{x}}_i = [1, y_{1,1,i}, y_{1,2,i}, y_{1,3,i}, ...]^T,$$

i.e., next to the bias term 1 all output variables of the first layer. We have used here the notation $y_{k,l,i}$ to denote the output of the l-th node of the k-th layer at the i-th epoch. We explicitly wrote the derivative $f'(x)$ as we will see that this term will evolve once w extend to several layers. Recall that in our analysis we could consume it into a varying step-size as long as it remained positive. Updating the first layer perceptrons is much more complicated as the derivative needs more consideration. We find for the update of the first layer weights:

$$\begin{aligned}
\underline{\hat{w}}_{1,l,i} &= \underline{\hat{w}}_{1,l,i-1} + \mu(y_i - f(\underline{\tilde{x}}_i^T \underline{\hat{w}}_{2,i-1})) f'(\underline{\tilde{x}}_i^T \underline{\hat{w}}_{2,i-1}) \frac{\partial \underline{x}_i^T \underline{\hat{w}}_{2,i-1}}{\partial \underline{\hat{w}}_{1,l,i-1}} \\
&= \underline{\hat{w}}_{1,l,i-1} + \mu(y_i - f(\underline{\tilde{x}}_i^T \underline{\hat{w}}_{2,i-1})) f'(\underline{\tilde{x}}_i^T \underline{\hat{w}}_{2,i-1}) (\underline{\hat{w}}_{2,i-1})_l f'(\underline{x}_i^T \underline{\hat{w}}_{1,l,i-1}) \underline{x}_i.
\end{aligned}$$

We have here introduced another index to identify which perceptrons weights are meant: $\underline{\hat{w}}_{k,l,i}$ the l-th perceptron of the k-th layer at epoch i. We also note that next to the artificial vector $\underline{\tilde{x}}_i$ also the original input vector $\underline{x}_i$ comes into the game.

For each iteration a data pair $(y_i, \underline{x}_i)$ is available. These can be the same values repeated several times or simply one by one from the training set. The obtained solution is not unique due to the symmetries of such network (swap two neurons, and you can obtain the same behaviour). Due to such symmetries, the initial values for an iterative algorithm may need to be selected carefully (i.e., different to ensure they do not select the same "line"). Due to the non-uniqueness of the solution, we cannot take the parameter error vector as a performance measure, only the output error is feasible.

7.2 Learning for Multiple Layers: Backpropagation

As we want to add more layers, things can become rather complicated in terms of notations. As every stage of layers may comprise of several perceptrons and each of that has inputs and outputs and several weights, it may be helpful to have a consistent notation. For this let us consider perceptron $l (l = 1, 2, ..., L)$ at layer $k (k = 1, 2, ..., K)$. Its input are the outputs of the previous layer, thus $y_{k-1,l}$ with $l = 1, 2, ..., L$ to which we add the bias term

$y_{k-1,0} = 1$. These previous layer outputs are thus inputs of the current layer and constitute its input vector

$$\underline{x}_k = [1, y_{k-1,1}, y_{k-1,2}, ..., y_{k-1,L}]^T$$

which for the first layer (input-layer) simply is the input feature vector

$$\underline{x}_1 = \underline{\hat{x}} = [1, x_1, x_2, ..., x_L]^T.$$

The weights at the input of the l-th perceptron at the k-th layer are denoted $w_{k,l,m}$ with index $m = 0, 1, 2, .., L$ also containing the bias term for $m = 0$. We write vector $\underline{\hat{w}}_{k,l}$ to denote all $L + 1$ elements of the weights.Its estimate is then $\underline{\hat{w}}_{k,l}$.

Furthermore, as our estimation keeps going with every new data pair $(\underline{\hat{x}}_i, y_i)$ that is offered to the network as training input, we add a further index i to indicate the state of input, or epoch, or iteration number of the update equations, i.e., $\underline{x}_{k,i}, y_{k,l,i}$, and $\underline{\hat{w}}_{k,l,i}$.

As such indexing is rather heavy we drop those indexes that are obvious. For example, the input vector $\underline{x}_{1,i}$ is simply $\underline{x}_i$, the last perceptron weights are $\underline{\hat{w}}_{K,1,i} = \underline{\hat{w}}_{K,i}$ and the final outcome is $y_{K,1,i} = y_{K,i}$.

With such nomenclature the previously explained two-layer network reads now

$$\begin{aligned}
y_{1,l,i} &= f(\underline{x}_i^T \underline{\hat{w}}_{1,l,i-1}); l = 1, 2, .., L \\
\underline{x}_{2,i} &= [1, y_{1,1,i}, y_{1,2,i}, ..., y_{1,L,i}]^T, \\
\underline{\hat{w}}_{2,i} &= \underline{\hat{w}}_{2,i-1} + \mu \underline{x}_{2,i}(y_{2,i} - f(\underline{x}_{2,i}^T \underline{\hat{w}}_{2,i-1}))f'(\underline{x}_{2,i}^T \underline{\hat{w}}_{2,i-1}), \\
\underline{\hat{w}}_{1,l,i} &= \underline{\hat{w}}_{1,l,i-1} + \mu(y_{2,i} - f(\underline{x}_{2,i}^T \underline{\hat{w}}_{2,i-1}))f'(\underline{x}_{2,i}^T \underline{\hat{w}}_{2,i-1}) \frac{\partial \underline{x}_{2,i}^T \underline{\hat{w}}_{2,i-1}}{\partial \underline{\hat{w}}_{1,l,i-1}}, \\
&= \underline{\hat{w}}_{1,l,i-1} + \mu(y_{2,i} - f(\underline{x}_{2,i}^T \underline{\hat{w}}_{2,i-1}))f'(\underline{x}_{2,i}^T \underline{\hat{w}}_{2,i-1}) \hat{w}_{2,l,i-1} f'(\underline{x}_i^T \underline{\hat{w}}_{1,l,i-1}) \underline{x}_i \\
&\quad ; l = 1, 2, ..., L.
\end{aligned}$$

Here we computed the derivative for the first layer by

$$\frac{\partial \underline{x}_{2,i}^T \underline{\hat{w}}_{2,i-1}}{\partial \underline{\hat{w}}_{1,l,i-1}} = \frac{\partial y_{1,l}}{\partial \underline{\hat{w}}_{1,l,i-1}} \underline{\hat{w}}_{2,l,i-1} = \underline{\hat{w}}_{2,l,i-1} f'(\underline{x}_i^T \underline{\hat{w}}_{1,l,i-1}) \underline{x}_i,$$

taking into account that for the $l-$th perceptron only one weight $\underline{\hat{w}}_{1,l,i-1}$ contributes. If we move on to a three layer network, the same will happen to the second layer but the first layer will have a more evolved derivative as there exist several paths from a given weight at the first layer to the last layer that all needs to be taken into account. We thus find for

the three layer network:

$$
\begin{aligned}
y_{2,l,i} &= f(\underline{x}_{2,i}^T \underline{\hat{w}}_{2,l,i-1}); l = 1,2,..,L \\
\underline{x}_{3,i} &= [1, y_{2,1,i}, y_{2,2,i}, ..., y_{2,L,i}]^T, \\
\underline{\hat{w}}_{3,i} &= \underline{\hat{w}}_{3,i-1} + \mu \underline{x}_{3,i}(y_{3,i} - f(\underline{x}_{3,i}^T \underline{\hat{w}}_{3,i-1}))f'(\underline{x}_{3,i}^T \underline{\hat{w}}_{3,i-1}), \\
y_{1,l,i} &= f(\underline{x}_i^T \underline{\hat{w}}_{1,l,i-1}); l = 1,2,..,L \\
\underline{x}_{2,i} &= [1, y_{1,1,i}, y_{1,2,i}, ..., y_{1,L,i}]^T, \\
\underline{\hat{w}}_{2,l,i} &= \underline{\hat{w}}_{2,l,i-1} + \mu(y_{3,i} - f(\underline{x}_{3,i}^T \underline{\hat{w}}_{3,i-1}))f'(\underline{x}_{3,i}^T \underline{\hat{w}}_{3,i-1}) \frac{\partial \underline{x}_{3,i}^T \underline{\hat{w}}_{3,i-1}}{\partial \underline{\hat{w}}_{2,l,i-1}}, \\
&= \underline{\hat{w}}_{2,l,i-1} + \mu(y_{3,i} - f(\underline{x}_{2,i}^T \underline{\hat{w}}_{3,i-1}))f'(\underline{x}_{3,i}^T \underline{\hat{w}}_{3,i-1}) \hat{w}_{3,l,i-1} f'(\underline{x}_{2,i}^T \underline{\hat{w}}_{2,l,i-1}) \underline{x}_{2,i} \\
&\quad ; l = 1,2,...,L \\
\frac{\partial y_{2,k,i-1}}{\partial \underline{\hat{w}}_{1,l,i-1}} &= f'(\underline{x}_{2,i}^T \underline{\hat{w}}_{2,k,i-1}) \hat{w}_{2,k,l,i-1} \frac{\partial y_{1,l,i-1}}{\partial \underline{\hat{w}}_{1,l,i-1}} \\
&= f'(\underline{x}_{2,i}^T \underline{\hat{w}}_{2,k,i-1}) \hat{w}_{2,k,l,i-1} f'(\underline{x}_i^T \underline{\hat{w}}_{1,l,i-1}) \hat{w}_{2,k,l,i-1}) \underline{x}_i, \\
\underline{\hat{w}}_{1,l,i} &= \underline{\hat{w}}_{1,l,i-1} + \mu(y_{3,i} - f(\underline{x}_{3,i}^T \underline{\hat{w}}_{3,i-1}))f'(\underline{x}_{3,i}^T \underline{\hat{w}}_{3,i-1}) \sum_{k=1}^{L} \frac{\partial y_{2,k,i-1}}{\partial \underline{\hat{w}}_{1,l,i-1}} \hat{w}_{3,k,i-1}, \\
&\quad ; l = 1,2,...,L
\end{aligned}
$$

We recognize that many parts of the update have been computed in the upper layers and can be reused. With a clever short notation the algorithm can be written much more compactly:

$$
\begin{aligned}
\delta_{1,k,l,i} &= \hat{w}_{2,k,l,i-1} f'(\underline{x}_{1,i}^T \underline{\hat{w}}_{1,l,i-1}); l = 1,2,...,L \\
\delta_{2,l,i} &= \hat{w}_{3,1,l,i-1} f'(\underline{x}_{2,i}^T \underline{\hat{w}}_{2,l,i-1}); l = 1,2,...,L \\
\delta_{3,i} &= (y_{3,1,i} - f(\underline{x}_{3,i}^T \underline{\hat{w}}_{3,l,i-1}))f'(\underline{x}_{3,i}^T \underline{\hat{w}}_{3,l,i-1}), \\
\underline{\hat{w}}_{3,1,i} &= \underline{\hat{w}}_{3,1,i-1} + \mu \delta_{3,i} \underline{x}_{3,i}, \\
\underline{\hat{w}}_{2,l,i} &= \underline{\hat{w}}_{2,l,i-1} + \mu \delta_{3,i} \delta_{2,l,i} \underline{x}_{2,i}; l = 1,2,...,L \\
\underline{\hat{w}}_{1,l,i} &= \underline{\hat{w}}_{1,l,i-1} + \mu \delta_{3,i} \underline{x}_{3,i} \left(\sum_{k=1}^{L} \delta_{2,k,i} \delta_{1,k,l,i} \right) \underline{x}_{1,i}; l = 1,2,...,L,
\end{aligned}
$$

in which we have used the full index notation as you can recognize how the development of networks with more than three layers would continue. Also it becomes quite clear that the initial values for the weights should not be zero as they appear proportional to the updates and thus would stop such updates.

Many modifications are possible:

1. For example, once $\underline{\hat{w}}_3$ is computed, δ_3 can be renewed and then $\underline{\hat{w}}_1$ and $\underline{\hat{w}}_2$ computed with the new δ_3. The same goes with δ_2 once $\underline{\hat{w}}_2$ is computed.

2. Different step-sizes $\mu_{k,l}$ for each layer and possibly adaptive step-sizes can be very useful.

3. Different nonlinear functions can be applied at each layer.

4. Extensions to more than three layers are possible.

So far we have just assumed to apply one activation function throughout the entire network. However, as there are many different ones, it may be useful to apply the "right" one, depending on the application of the network. Principally we have the following choices:

ReLU (rectifying Linear Unit) are very popular to form classes or pre-classes

Sigmoid or hyptan are commonly used for the binary classification problem as choice for the output layer.

Normalized activation functions are often selected when a selection "one out of many" is to be achieved.

Linear outcome (no non-linear mapping) can be selected as the outcome of the latent (center) nodes of an auto-encoder.

As the outputs of each neuron are between zero and one, they can be interpreted as probabilities to obtain 0 and 1. A measure in terms of probabilities is cross entropy

$$\sum_{i=1}^{M} y_i \ln(f(\underline{x}_i^T \underline{\hat{w}})) - (1 - y_i) \ln(1 - f(\underline{x}_i^T \underline{\hat{w}}))$$

Often to ensure probability measures, new nonlinear activity functions are introduced by proper normalization:

$$y_i = f(\underline{x}_i^T \underline{\hat{w}}) = \frac{e^{\underline{x}_i^T \underline{\hat{w}}}}{\sum e^{\underline{x}_i^T \underline{\hat{w}}}}.$$

Now all potential outcomes y_i are between zero and one and add up to one just as probabilities do.

Chapter 8

Boosting

Although the gradient approach is our method of choice, it can also show problems in practice. A slow learning is often a problem. The reason for this can be manifold. Often the input vectors (ans also those at hidden layers) are correlated slowing down a gradient approach as we have learned form the stochastic behavior of the LMS algorithm. Here, a decorrelation approach such as the Newton Method can be successful. We will thus investigate the so-called RLS algorithm to understand how the decorrelation process works. But sometimes a gradient type algorithm slows down simply because of a saddle point in which the error surface flattens out. Theoretically the algorithm can even get stuck there. A cure for this can be the so-called Momentum LMS Algorithm that will be explained in the following of this chapter. An other problem occurs if the data are changing over time and the learning algorithm needs to track, that is to forget old values and put more emphasis on new values. To this end we have to understand the tracking behavior of gradient algorithms which will be analysed further on. Note, however, as the amount of data and processing power has substantially increased over the past 30 years, increasing learning speed is not such a big concern any more as it used to be in the 80ies.

8.1 Choice of the Step-Size

Analysing the LMS algorithm we found that the choice of the step-size depends strongly on the input statistics. In general, we can say: a small step-size causes slow learning. However, the opposite is not necessarily true as a too-large step-size causes instability of the algorithm and no learning at all is experienced. The question is thus how to find an **optimal** step-size, thus a step-size for which the algorithm learns stably and does it in a fast manner. One solution in this direction is **Normalization**. There exist several approaches:

- All input signals are normalized, for example, to lie in the range $[-1, 1]$. The method is simple and needs only be done once at initialization. Essentially this does not change the problem of correlation but restricts the proper choice of a step-size dramatically.

- All input vectors (regression vectors) are normalized before applying. This is a typical step that we have observed in the NLMS. It indeed does not only restrict the input range but also impacts the correlation. However, it comes with the price of higher computation as this needs to be done before every update in every neuron of the neural network.
- Another option is to adaptively optimize the step-size search by computing the gradient w.r.t. the step-size as well. This comes with the cost of an even higher computational complexity.
- Finally, it is possible to weight each element of the regression vector individually by including an adaptive positive definite diagonal matrix. This is known as the PNLMS algorithm [16, 69].

Any of these methods can significantly improve the learning rate but still require searching for an optimal step-size value, however, now in a much more limited range. An overview of methods applied at LMS can be found in [6].

8.2 The Momentum LMS Algorithm

A typical problem that occurs in learning when utilizing a gradient type algorithm is that if the error surface becomes flat, the algorithm gets stuck as depicted in Figure 8.1. Typically saddle points are a problem as the gradient algorithm may get stuck right on the point where the gradient becomes zero. In literature many alternatives are known to overcome this mostly in the context of offline algorithms. The most popular ones are simulated annealing and genetic algorithms. In on-line algorithms however, gradient algorithms remain best suited. A cure for this problem is to introduce a so-called momentum term. Such term takes past directions into account during a phase in which the gradient has not been zero. The LMS update then reads

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \mu_k e_k \underline{x}_k^H + \eta_k(\underline{\hat{w}}_{k-1} - \underline{\hat{w}}_{k-2}).$$

The additional term called momentum term causes that past directions are taken into account to prevent a complete stop of the algorithm. However, in a regular learning mode with steep gradients one needs to take into account that the term η may speed up the learning and thus the original step-size μ_k needs to be lowered. A smarter method observes the error energy $|e_k|^2$ and increases η_k only if it becomes small.

8.3 Increasing the Dimension

Another method that is very useful is increasing the dimensions. In fact this is likely **The Method** that explains the success of deep learning. First of all it sounds counter-intuitive

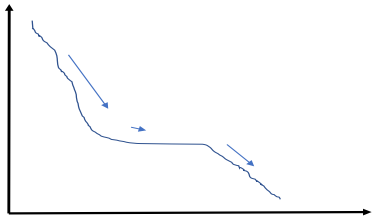

Figure 8.1: *Problems of gradient type algorithm.*

as we know that the learning rate of the gradient algorithm decreases with the number of coefficients. However, the problem in multi-layer networks is the large number of local minima in which a gradient search gets easily stuck into. Increasing the dimension converts many of such local minima into saddle points which are much easier to cope with for the gradient type algorithm. Once hitting such a local minimum, the algorithm simply moves on in another direction.

Let's analyse this on the following example. We assume an error function of the form:

$$f(x,y) = (R^2 - x^2 - y^2)^2 + R^4 \arctan(x).$$

Without the $\arctan(x)$ term we find a spherical behavior for which the minimum is obtains as long as $x^2 + y^2 = R^2$. If only one dimension (x) is given, we find two symmetric minima at $x_o = (-R, R)$. Due to the additional $\arctan(x)$ term, the first minimum at $x = -R$ becomes lower and thus the global minimum. A simple 1D gradient search, starting at the right of the local minima gets stuck there. Figure 8.2 depicts the situation for $R = 3$. A gradient type algorithm starting right of the local minimum, say for example at $x_0 = 5$ gets stuck at $x_* = 3$.

Figure 8.3 shows the result when using gradient type algorithms for the problem stated above. The simple gradient algorithm in a single dimension gets quickly stuck at the local minimum for $x = 3$. Adding a momentum term ($\eta = 0.9$) results in an oscillation but the algorithm finds its way over the maximum to the global minimum at $x = -3$. As we also use older values of x, the recursive algorithm now becomes of order two which explains the oscillations. If the dimension is increased by utilizing the y parameter in the error

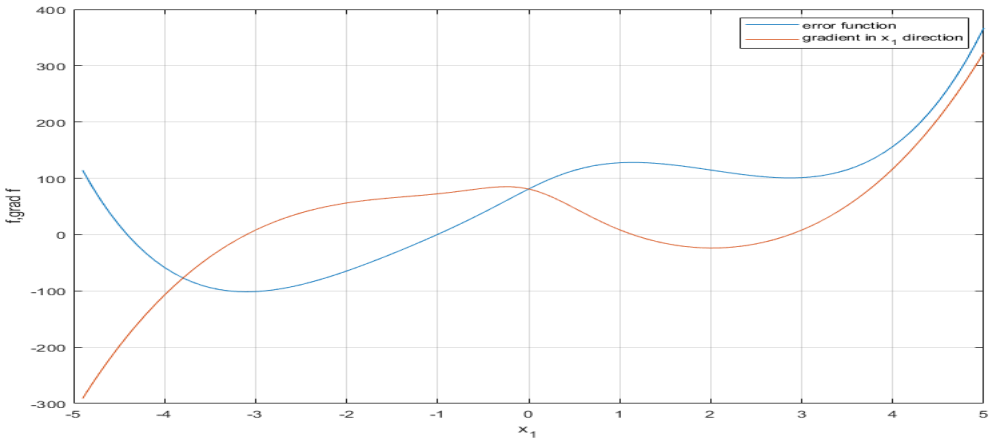

Figure 8.2: *Problems of gradient type algorithm: error function with local and global minimum (in blue) as well as corresponding gradient (in red). We have a local minimum at* $x = 3$ *and a global one at* $x = -3$.

function, the gradient algorithm recognizes the local minimum as saddle point and moves onto another direction leading to the desired global minimum. All three algorithms were run with the same (constant) step-size $\mu = 0.002$.

8.4 Transfer Learning

Transfer Learning reaches back into the 70ies [7] when it was already speculated that similar situations result in similar learning, for example training a car identifier could also be useful for trucks. Originally more influential for the psychological literature on the transfer of learning, in the 90ies more and more technique were developed to speed up training by re-using existing training. Currently it is viewed as the next driver of ML commercial success.

8.5 Adversarial Learning

In the beginning of the 00ies some researchers found that when distorting inputs that were classified correctly in their original form, the classification failed, even so the distortion could be rather small. An illustrative example was demonstrated in [25] where a relative

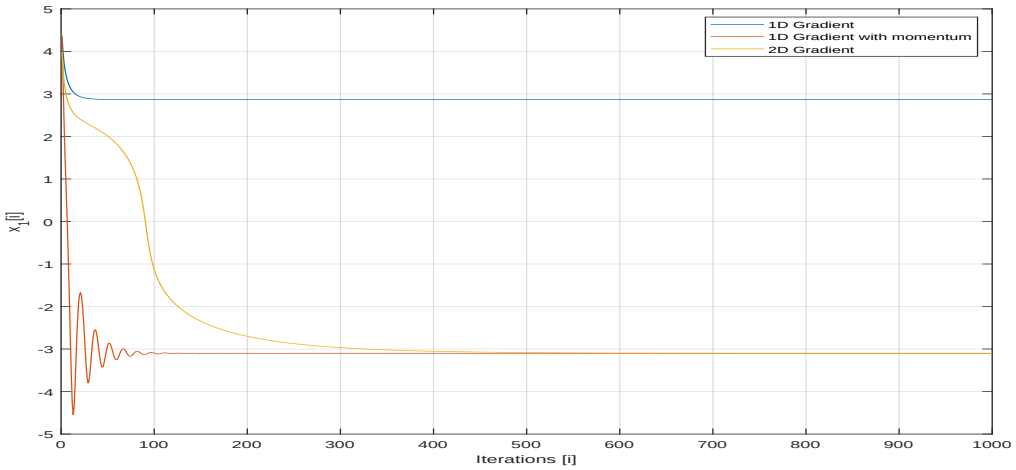

Figure 8.3: *Problems of gradient type algorithm: 1D gradient gets stuck in local minimum, momentum gradient algorithm finds global minimum after some oscillation, adding one dimension makes the gradient approach becoming successful.*

small but systematically corrupted picture led to a complete failure in classification. Figure 8.4 depicts the example. Since that time, researchers are applying corrupted inputs to make the classifiers more robust.

Also, the RLS and Kalman Filter can be viewed as boosting methods for the performance of learning Algorithms. They will be explained in detail in the next sections.

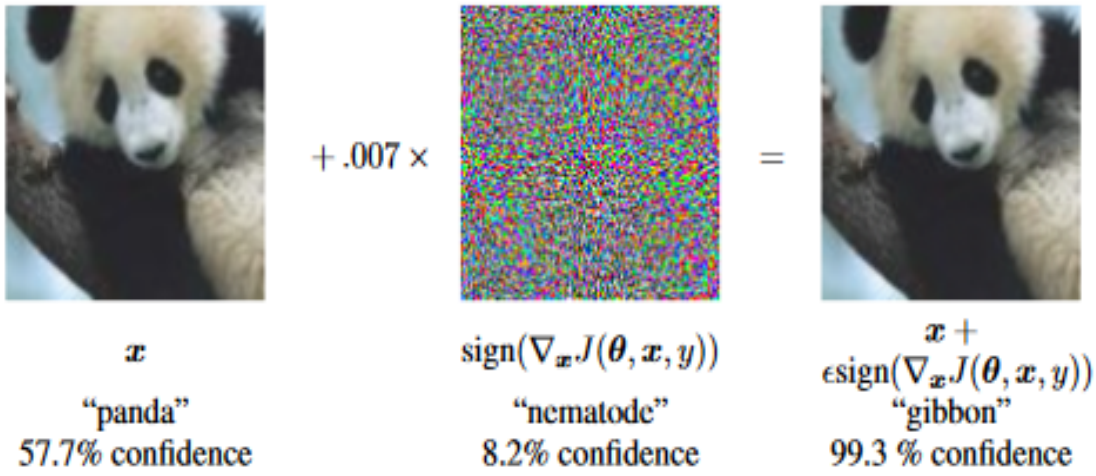

Figure 8.4: *Adversarial learning: A demonstration of fast adversarial example generation applied to GoogLeNet on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input and only influence the LSB, the GoogLeNet's classification of the image could be changed entirely.*

Chapter 9

The RLS Algorithm

Next to the LMS algorithm the Recursive Least Squares (RLS) algorithm is the most prominent one. The problems that come with RLS often exclude it in practical applications. As the RLS algorithm is *just* a recursive implementation of the LS problem, its properties are identical to a classic LS solution. We therefore will start with a brief introduction into the problem of least squares.

9.1 Least Squares Problems

We consider again the estimation problem based on the following observation

$$\underline{d}_N = X_N \underline{w}_o + \underline{v}_N. \tag{9.1}$$

Here we wrote N observations $d(k) = \underline{w}_o^T \underline{x}_k + v(k), k = 1..N$ in vectors and matrices:

$$\underline{d}_N^T = [d(1), d(2), ..., d(N)], \tag{9.2}$$

$$\underline{v}_N^T = [v(1), v(2), ..., v(N)], \tag{9.3}$$

$$\underline{w}_o^T = [w_o(1), w_o(2), ..., w_o(M)], \tag{9.4}$$

$$X_N = \begin{bmatrix} x(1) & x(2) & ... & x(M) \\ x(2) & x(3) & ... & x(M+1) \\ x(3) & x(4) & ... & x(M+2) \\ \vdots & \vdots & ... & \vdots \\ x(N) & x(N+1) & ... & x(N+M-1) \end{bmatrix} = \begin{bmatrix} \underline{x}_1^T \\ \underline{x}_2^T \\ \underline{x}_3^T \\ \vdots \\ \underline{x}_N^T \end{bmatrix}. \tag{9.5}$$

Depending on whether we have $N < M$ or $N \geq M$ we distinguish the *underdetermined* and the *overdetermined* case (of a system of equations). The problem is to estimate the parameter vector $\underline{w}_o$ optimally based on the observation $\underline{d}_N$ and X_N, that is to minimize the cost function

$$g_{LS}(\underline{\hat{w}}) = \|\underline{d}_N - X_N \underline{\hat{w}}\|_2^2. \tag{9.6}$$

Note that the solution depends on the number of observations N. Is the value of N small so that the set of equations is underdetermined and grows with time, the problem is called LS with growing window[1]. The solution is given by

$$\underline{w}_{LS,N} = \underline{w}_N = \arg\min_{\underline{\hat{w}}} \|\underline{d}_N - X_N\underline{\hat{w}}\|_2^2. \tag{9.7}$$

As we only discuss LS estimations in this chapter, we will leave out the index 'LS'. We keep, however, the index N as it does not only indicate of how many observations we are using but for a growing window it also denotes the time. If not indicated otherwise, we will assume in the following that

$$\text{rank}(X_N) = \min(M, N). \tag{9.8}$$

As mentioned above, we have to distinguish two cases:

- The system of equations is underdetermined, that is $M > N$ –we have less observations than parameter to estimate. In this case we assume: $\text{rank}(X_N) = N$.
- The system of equations is overdetermined, that is $M \leq N$ –we have more observations than parameters to estimate. In this case we assume: $\text{rank}(X_N) = M$.

Differentiating of the quadratic form (9.7) with respect to the unknown vector leads to the following orthogonality condition:

$$\frac{\partial\|\underline{d}_N - X_N\underline{\hat{w}}\|_2^2}{\partial\underline{\hat{w}}} = -\left(\underline{d}_N - X_N\underline{\hat{w}}\right)^H X_N = \underline{0}. \tag{9.9}$$

From here we find the solution $\underline{\hat{w}}_{LS,N}$

$$X_N^H X_N \underline{\hat{w}}_{LS,N} = X_N^H \underline{d}_N. \tag{9.10}$$

By differentiating a second time we can validate that we indeed have a minimum:

$$\frac{\partial^2\|\underline{d}_N - X_N\underline{\hat{w}}\|_2^2}{\partial^2\underline{\hat{w}}} = X_N^H X_N > 0. \tag{9.11}$$

The form $X_N^H X_N > 0$ indicates that the matrix $X_N^H X_N$ is positive definite. The minimum cost function can be computed without explicitly knowing the LS solution:

$$\begin{aligned} g_{LS}(\underline{\hat{w}}_{LS,N}) &= (\underline{d}_N - X_N\underline{\hat{w}}_{LS,N})^H(\underline{d}_N - X_N\underline{\hat{w}}_{LS,N}) = \underline{d}_N^H(\underline{d}_N - X_N\underline{\hat{w}}_{LS,N}) && (9.12)\\ &= \|\underline{d}_N\|_2^2 - \underline{d}_N^H X_N (X_N^H X_N)^{-1} X_N^H \underline{d}_N. && (9.13) \end{aligned}$$

Compare this equation with the corresponding equations of the *steepest descent* method. Except of the expectation values they formally appear identical. Also the various terms

[1] in contrast to a sliding window whose size remains constant

can be interpreted as instantaneous estimates of such expectation values: $X_N^H \underline{d}_N = \hat{\underline{r}}_{\underline{\mathbf{x}}\mathbf{d}}, X_N^H X_N = \hat{R}^*_{\underline{\mathbf{x}}\underline{\mathbf{x}}}$. Also, a general quadratic form in terms of $\underline{\hat{w}}_{LS,N}$ can be provided:

$$g_{LS}(\underline{\hat{w}}_N) = g_{LS}(\underline{\hat{w}}_{LS,N}) + (\underline{\hat{w}}_{LS,N} - \underline{\hat{w}}_N)^H X_N^H X_N (\underline{\hat{w}}_{LS,N} - \underline{\hat{w}}_N). \tag{9.14}$$

Both equations, (9.10) and (9.12), can be combined as the so-called normal equations:

$$\begin{bmatrix} \underline{d}_N^H \underline{d}_N & \underline{d}_N^H X_N \\ X_N^H \underline{d}_N & X_N^H X_N \end{bmatrix} \begin{bmatrix} 1 \\ -\underline{\hat{w}}_N \end{bmatrix} = \begin{bmatrix} g_{LS}(\underline{\hat{w}}_N) \\ \underline{0} \end{bmatrix}. \tag{9.15}$$

9.1.1 Existence

Before we continue with closed form solutions, we have to consider **existence** and **uniqueness** of the LS solution.

Lemma 9.1 *If X_N is of row rank $N \geq M$, then the solution is unique and given by $\underline{\hat{w}}_{LS,N} = [X_N^H X_N]^{-1} X_N^H \underline{d}_N$.*
If X_N is not of row rank N, then the normal equations have more than one solution of which any two solutions $\underline{\hat{w}}_1$ and $\underline{\hat{w}}_2$ differ by a vector in the nullspace of X_N, thus $X_N[\underline{\hat{w}}_2 - \underline{\hat{w}}_1] = \underline{0}$.

Proof: Let us assume there is a vector $\underline{z}$, so that

$$\underline{z} = X_N^H X_N \underline{p} = X_N^H \underline{q}.$$

The columns of $X_N^H X_N$ span a space (column range) to which the vector $\underline{z}$ belongs to. If there is a vector, different from the zero vector, $\underline{p}$ for which we have that $X_N^H X_N \underline{p} = \underline{0}$, then such vector $\underline{p}$ belongs to the null space of $X_N^H X_N$. Assuming that X_N is of full row rank M (defined by the dimension M of the vector $\underline{p}$), the nullspace is empty. As X_N and $X_N^H X_N$ have the same nullspace (see also Appendix E), there cannot exist a vector different from the zero vector for which $X_N \underline{p} = \underline{0}$. Thus, with choice of vector $\underline{q}$ the vector $\underline{p}$ is **uniquely** defined and vice versa.

In the second part we assume that X_N is not of full row rank M, i-e., $N < M$. Then $X_N^H X_N$ is also not of full row rank and solutions $\underline{p}$ different from the zero vector exist in the null space of $X_N^H X_N$ to which exist the same solutions in the nullspace of X_N (as it is the same nullspace). The choice of $\underline{q}$ is now not uniquely defined by $\underline{p}$. □

In the case of the underdetermined equation system ($\text{rank}(X_N) = N < M$) we obtain the following solution:

$$\underline{\hat{w}}_{LS,N} = X_N^H \left[X_N X_N^H\right]^{-1} \underline{d}_N. \tag{9.16}$$

The matrix $X_N X_N^H$ spans a smaller space and thus the nullspace will become larger. A multitude of solutions exist under which those with minimum norm are of most interest.

With Singular Value Decomposition (SVD) it can be shown [33], that the solution (9.16) has minimum norm.

9.1.2 LS Estimation

Lemma 9.2 *Consider the following linear model*

$$\underline{\mathbf{d}}_N = X_N \underline{w}_o + \underline{\mathbf{v}}_N. \tag{9.17}$$

Let X_N be a deterministic matrix of full rank and $\underline{w}_o$ an unknown deterministic variable. Let the distortion $\underline{\mathbf{v}}_N$ be a zero-mean random vector with variance one. Then the LS estimator $\underline{\hat{\mathbf{w}}}_{LS,N} = [X_N^H X_N]^{-1} X_N^H \underline{\mathbf{d}}_N$ has the following properties:

- $E[\underline{\hat{\mathbf{w}}}_{LS,N}] = \underline{w}_o$,
- $E[(\underline{\hat{\mathbf{w}}}_{LS,N} - \underline{w}_o)(\underline{\hat{\mathbf{w}}}_{LS,N} - \underline{w}_o)^H] = [X_N^H X_N]^{-1}$,
- *the LS-Estimator is the best, linear estimate without a bias (Best Linear UnBiased=BLUE).*

Proof: Plugging in the estimate in the first property and we find:

$$\mathrm{E}[\underline{\hat{\mathbf{w}}}_{LS,N}] = \mathrm{E}\left[[X_N^H X_N]^{-1} X_N^H \underline{\mathbf{d}}_N\right].$$

If we furthermore substitute $\underline{\mathbf{d}}_N$ from (9.17) we obtain:

$$\mathrm{E}[\underline{\hat{\mathbf{w}}}_{LS,N}] = \underline{w}_o + \mathrm{E}\left[[X_N^H X_N]^{-1} X_N^H \underline{\mathbf{v}}_N\right] = \underline{w}_o$$

due to the zero-mean property of the distortion.

The second part is also shown by straightforward substitution:

$$\begin{aligned} \mathrm{E}[(\underline{\hat{\mathbf{w}}}_{LS,N} - \underline{w}_o)(\underline{\hat{\mathbf{w}}}_{LS,N} - \underline{w}_o)^H] &= \mathrm{E}\left[\left([X_N^H X_N]^{-1} X_N^H \underline{\mathbf{v}}_N\right)\left([X_N^H X_N]^{-1} X_N^H \underline{\mathbf{v}}_N\right)^H\right] \\ &= \mathrm{E}\left[[X_N^H X_N]^{-1} X_N^H \underline{\mathbf{v}}_N \underline{\mathbf{v}}_N^H X_N [X_N^H X_N]^{-1}\right]. \end{aligned}$$

The first expectation over the noise results in the unit matrix and finally we find the desired result.
The third property is shown by assuming an arbitrary linear estimator B, thus

$$\underline{\bar{\mathbf{w}}} = B \underline{\mathbf{d}}_N. \tag{9.18}$$

In order to be unbiased it needs to be $BX_N = I$. Thus the estimator is

$$\underline{\bar{\mathbf{w}}} = \underline{w}_o + B \underline{\mathbf{v}}_N. \tag{9.19}$$

Its covariance matrix becomes

$$\mathrm{E}[(\bar{\underline{\mathbf{w}}} - \underline{w}_o)(\bar{\underline{\mathbf{w}}} - \underline{w}_o)^H] = BB^H. \tag{9.20}$$

Consider now the positive semidefinite matrix CC^H, that is constructed by $C = B - [X_N^H X_N]^{-1} X_N^H$, we find

$$BB^H \geq [X_N^H X_N]^{-1} \tag{9.21}$$

and thus the desired result. □

9.1.3 Conditions on Excitation

Until now we assumed the matrix $X_N^H X_N$ to be of full rank. With the definition in Equation (9.5) we can also write the matrix as sum of outer vector products:

$$X_N^H X_N = \sum_{i=1}^{N} \underline{x}_i \underline{x}_i^H. \tag{9.22}$$

With each additional term $\underline{x}_i \underline{x}_i^H$ the matrix can grow in rank. In similar form as in the LMS algorithm we can also provide a necessary condition on excitation.

Lemma 9.3 *A set of input vectors* $\{\underline{x}_k, k > 0\}$ *is a persistent excitation if positive numbers* α, β, N_o *exist so that*

$$\alpha I < \sum_{k=n}^{n+N_o} \underline{x}_k \underline{x}_k^H \leq \beta I, \text{ for all } n. \tag{9.23}$$

9.1.4 Generalizations and special cases

The considerations so far can be extended by augmenting the equations with the initial conditions $\bar{\underline{w}}$ and by utilizing a positive definite weight matrix Q. Furthermore, the initial values obtain also a weighting Π_o^{-1}, that can be interpreted as confidence in such initial values. If the initial value $\bar{\underline{w}}$ is very certain, we set Π_o^{-1} to a large value. The first term is then dominating the following ones. In the following we assume the matrices $\Pi \in I\!R^{M\times M}$ and $Q \in I\!R^{N\times N}$, to by symmetric and positive definite. We then obtain the following cost function:

$$g_{WLS}(\hat{\underline{w}}_N) = (\hat{\underline{w}}_N - \bar{\underline{w}})^H \Pi_o^{-1} (\hat{\underline{w}}_N - \bar{\underline{w}}) + (\underline{d}_N - X_N \hat{\underline{w}}_N)^H Q (\underline{d}_N - X_N \hat{\underline{w}}_N). \tag{9.24}$$

The corresponding normal equations are:

$$\begin{bmatrix} (\underline{d}_N - X_N\underline{\bar{w}})^H Q(\underline{d}_N - X_N\underline{\bar{w}}) & (\underline{d}_N - X_N\underline{\bar{w}})^H Q X_N \\ X_N^H Q(\underline{d}_N - X_N\underline{\bar{w}}) & \Pi_o^{-1} + X_N^H Q X_N \end{bmatrix} \begin{bmatrix} 1 \\ \underline{\bar{w}} - \underline{\hat{w}}_{WLS,N} \end{bmatrix} = \begin{bmatrix} g_{WLS}(\underline{\hat{w}}_{WLS,N}) \\ \underline{0} \end{bmatrix}. \tag{9.25}$$

All LS methods discussed so far assume a growing window over the input values. As a consequence all values impact the final solution. If the system changes during the adaptation process, the estimation would have to be wrong. A weighting in the LS solution allows to emphasize the newer values than the older ones (see also Exercise 9.1 ahead). A formulation in which similar to the LMS algorithm only a set of new values is being applied and old ones discarded (apart from the accumulation in the a-priori estimates) applies a rectangular window (also-called sliding window) over the input vectors. We then minimize the following function:

$$g_{APA,k}(\underline{\hat{w}}) = \sum_{i=k-N+1}^{k} |d(i) - \underline{x}_i^T \underline{\hat{w}}|^2. \tag{9.26}$$

By the choice of the window length N, the solution is determined. If $N < M$, we have an underdetermined problem and we need to take solution (9.16). Alternatively such solution is useful if the algorithm is not excited persistently. This underdetermined algorithm is also known in literature under the name *Affine Projection Algorithm* (APA). For the spacial case $N = 1$ we obtain the NLMS algorithm with step-size $\mu(k) = 1/[\|\underline{x}_k\|_2^2]$.

9.1.5 Summary

Finally we like to compare the RLS algorithm in Table 9.1 with respect to a stochastic and a deterministic view. In order to achieve this we extended (9.17) in such a way that the estimated system is considered a random variable. Table 9.1 reveals a few unexpected analogies.

Exercise 9.1 *Consider the LS cost function in the form*

$$g_{LS}(\underline{\hat{w}}_N) = \sum_{k=1}^{N} \lambda^{N-k} |\tilde{e}_a(k|\underline{\hat{w}}_N)|^2 = \sum_{k=1}^{N} \lambda^{N-k} |d(k) - \underline{x}_k^T \underline{\hat{w}}_N|^2 \tag{9.27}$$

Derive the normal equations.

Exercise 9.2 *Derive the normal equations of (9.25).*

stochastic	deterministic
$\underline{\mathbf{d}} = X\underline{\mathbf{w}} + \underline{\mathbf{v}}$,	$\underline{d} = X\underline{w} + \underline{v}$
$m_{\underline{\mathbf{w}}} = \mathrm{E}[\underline{\mathbf{w}}]$	$\underline{w}_o = \bar{\underline{w}}$
$\mathrm{E}[(\underline{\mathbf{w}} - m_{\underline{\mathbf{w}}})(\underline{\mathbf{w}} - m_{\underline{\mathbf{w}}})^H] = R_{\underline{\mathbf{ww}}}$	Π_o
$m_{\underline{\mathbf{v}}} = \mathrm{E}[\underline{\mathbf{v}}]$	$\underline{v}_o$
$\mathrm{E}[(\underline{\mathbf{v}} - m_{\underline{\mathbf{v}}})(\underline{\mathbf{v}} - m_{\underline{\mathbf{v}}})^H] = R_{\underline{\mathbf{vv}}}$	Q^{-1}
$m_{\underline{\mathbf{d}}} = Xm_{\underline{\mathbf{x}}} + m_{\underline{\mathbf{v}}}$	$\underline{d}_o = X\underline{w}_o + \underline{v}_o$
$\hat{\underline{\mathbf{w}}}$	$\hat{\underline{w}}$
$\min_K \mathrm{E}\|\underline{\mathbf{w}} - m_{\underline{\mathbf{w}}} - K(\underline{\mathbf{d}} - m_{\underline{\mathbf{d}}})\|_2^2$	$\min_{\underline{w}} \left[(\underline{w} - \underline{w}_o)^H \Pi_o^{-1} (\underline{w} - \underline{w}_o) + \|\underline{d} - X\underline{w} - \underline{v}_o\|_Q^2\right]$
$K_o = R_{\underline{\mathbf{ww}}} X^H [X R_{\underline{\mathbf{ww}}} X^H + R_{\underline{\mathbf{vv}}}]^{-1}$	$K_o = \Pi_o X^H [X \Pi_o X^H + Q^{-1}]^{-1}$
$K_o = [R_{\underline{\mathbf{ww}}}^{-1} + X^H R_{\underline{\mathbf{vv}}}^{-1} X]^{-1} X^H R_{\underline{\mathbf{vv}}}^{-1}$	$K_o = [\Pi_o^{-1} + X^H Q X]^{-1} X^H Q$
$\hat{\underline{\mathbf{w}}} = K_o[\underline{\mathbf{d}} - Xm_{\underline{\mathbf{w}}} - m_{\underline{\mathbf{v}}}]$	$\hat{\underline{w}} = K_o[\underline{d} - X\underline{w}_o - \underline{v}_o]$

Table 9.1: *Comparison of terms of the LS algorithm in stochastic and deterministic descriptions.*

Exercise 9.3 *Based on the statement of Lemma 9.1 show that the LS algorithm is also capable of a linear prediction. For this consider the autoregressive random process of order* P

$$\mathbf{x}(k) = \sum_{i=1}^{P} \mathbf{x}(k-i)a(i) + \mathbf{v}(k) \tag{9.28}$$

and estimate by LS its coefficients $a(i)$*. Show the estimator's properties by formulating the random process in vector notation of length* $M > P$*. Show that for such vectors* $\hat{\underline{\mathbf{v}}}_k = \underline{\mathbf{x}}_k - \mathbf{X}_k \hat{\underline{\mathbf{a}}}$ *we have*

$$\begin{bmatrix} \underline{\mathbf{x}}_k^T \\ \mathbf{X}_k^T \end{bmatrix} \frac{\hat{\underline{\mathbf{v}}}_k}{\|\hat{\underline{\mathbf{v}}}_k\|_2^2} = \begin{bmatrix} 1 \\ \underline{0} \end{bmatrix}. \tag{9.29}$$

Exercise 9.4 *To transmit data over a wireless channel, a constant modulus signal (*$|x(k)| = 1$*) is employed. The best channel estimation can be achieved if trace(*$[X_N^H X_N]^{-1}$*) becomes minimal. By which property(ies) of the transmitted training signal can this be achieved?*

Exercise 9.5 *Minimize the following cost function with constraint*

$$\|\hat{\underline{w}}_k - \hat{\underline{w}}_{k-1}\|_2^2 + \lambda \|\underline{d}_k - X_P(k)\hat{\underline{w}}_k\|_2^2, \tag{9.30}$$

for optimal $\hat{\underline{w}}_k$*. Let* $X_P(k)$ *be a matrix with instantaneous value* $\underline{x}_k$ *and past values* $\underline{x}_{k-1} \cdots \underline{x}_{k-P+1}$*.*
Consider alternative the formulation

$$\|\hat{\underline{w}}_k - \hat{\underline{w}}_{k-1}\|_2^2 + \underline{\lambda}^T [\underline{d}_k - X_P(k)\hat{\underline{w}}_k] + \underline{\lambda}^H [\underline{d}_k - X_P(k)\hat{\underline{w}}_k]^* \tag{9.31}$$

Let the factor $\underline{\lambda}$ *be a Lagrange multiplier.*

9.2 Classic RLS Derivation

A major drawback of the LS method is so far that the inverse of the Gramian matrix $X_N^H X_N$ needs to be computed every time anew once a new observation arrives. As soon as there are more observations N than parameters M, the Gramian to invert is of dimension $M \times M$, and can, assuming it is of full rank M be inverted by standard Matrix-Inversion methods that are of order $O(M^3)$. Furthermore, the number of terms to compute the matrix $X_N^H X_N$ grows with the observations linearly in NM. It is thus of great interest to develop a method that allows to compute the result with at least operations and memory as possible. In order to achieve this, let us consider the case of $N+1$ observations:

$$\begin{aligned} g_{LS}(\underline{\hat{w}}_{N+1}) &= (\underline{\hat{w}}_{N+1} - \underline{\bar{w}})^H \Pi_o^{-1} (\underline{\hat{w}}_{N+1} - \underline{\bar{w}}) + (\underline{d}_{N+1} - X_{N+1}\underline{\hat{w}}_{N+1})^H (\underline{d}_{N+1} - X_{N+1}\underline{\hat{w}}_{N+1}), \\ &= (\underline{\hat{w}}_{N+1} - \underline{\bar{w}})^H \Pi_o^{-1} (\underline{\hat{w}}_{N+1} - \underline{\bar{w}}) + \left\| \begin{bmatrix} \underline{d}_N \\ d(N+1) \end{bmatrix} - \begin{bmatrix} X_N \\ \underline{x}_{N+1}^T \end{bmatrix} \underline{\hat{w}}_{N+1} \right\|_2^2. \quad (9.32) \end{aligned}$$

If we consider the solution at N observations, we can split the solution at $N+1$ into an already computed part and a new part:

$$\begin{aligned} \underline{\hat{w}}_{N+1} &= \left[\Pi_o^{-1} + X_{N+1}^H X_{N+1}\right]^{-1} X_{N+1}^H \underline{d}_{N+1} & (9.33) \\ &= \left[\Pi_o^{-1} + \left[X_N^H \underline{x}_{N+1}^*\right] \begin{bmatrix} X_N \\ \underline{x}_{N+1}^T \end{bmatrix}\right]^{-1} \left[X_N^H \underline{x}_{N+1}^*\right] \begin{bmatrix} \underline{d}_N \\ d(N+1) \end{bmatrix} & (9.34) \\ &= \left[\Pi_o^{-1} + X_N^H X_N + \underline{x}_{N+1}^* \underline{x}_{N+1}^T\right]^{-1} \left[X_N^H \underline{d}_N + \underline{x}_{N+1}^* d(N+1)\right]. & (9.35) \end{aligned}$$

Defining the matrix

$$P_{N+1} \triangleq \left[\Pi_o^{-1} + X_{N+1}^H X_{N+1}\right]^{-1};\ P_0 = \Pi_o, \quad (9.36)$$

we recognize the following recursion:

$$P_{N+1}^{-1} = P_N^{-1} + \underline{x}_{N+1}^* \underline{x}_{N+1}^T;\ P_0 = \Pi_o. \quad (9.37)$$

With help of the Matrix-inversion Lemma 3.1 we can describe also this recursion in its inverted form and obtain:

$$P_{N+1} = P_N - \frac{P_N \underline{x}_{N+1}^* \underline{x}_{N+1}^T P_N}{1 + \underline{x}_{N+1}^T P_N \underline{x}_{N+1}^*},\ P_0 = \Pi_o. \quad (9.38)$$

We now recognize the meaning of our initial certainty parameter Π_o. This recursion does not require a matrix inversion, that is instead of $O(M^3)$ we only require $O(M^2)$ operations.

The substitution of the recursive form in Eq. (9.35) results in the following:

$$\begin{aligned}
\underline{\hat{w}}_{N+1} &= P_{N+1}\left[X_N^H \underline{d}_N + \underline{x}_{N+1}^* d(N+1)\right] && (9.39)\\
&= \left[P_N - \frac{P_N \underline{x}_{N+1}^* \underline{x}_{N+1}^T P_N}{1+\underline{x}_{N+1}^T P_N \underline{x}_{N+1}^*}\right]\left[X_N^H \underline{d}_N + \underline{x}_{N+1}^* d(N+1)\right] && (9.40)\\
&= \underbrace{P_N X_N^H \underline{d}_N}_{\underline{\hat{w}}_N} - \frac{P_N \underline{x}_{N+1}^* \underline{x}_{N+1}^T}{1+\underline{x}_{N+1}^T P_N \underline{x}_{N+1}^*} \underbrace{P_N X_N^H \underline{d}_N}_{\underline{\hat{w}}_N} + P_N \underline{x}_{N+1}^* \underbrace{\left[1 - \frac{\underline{x}_{N+1}^T P_N \underline{x}_{N+1}^*}{1+\underline{x}_{N+1}^T P_N \underline{x}_{N+1}^*}\right]}_{\frac{1}{1+\underline{x}_{N+1}^T P_N \underline{x}_{N+1}^*}} d(N+1)\\
&= \underline{\hat{w}}_N + \frac{P_N \underline{x}_{N+1}^*}{1+\underline{x}_{N+1}^T P_N \underline{x}_{N+1}^*}\left[d(N+1) - \underline{x}_{N+1}^T \underline{\hat{w}}_N\right]. && (9.41)
\end{aligned}$$

This description is not so different to the description of the LMS algorithm. The essential difference is the new regression vector, thus a new direction for the updates than before. In the RLS algorithm this direction depends on the previous directions. Let us consider the regression vector

$$\begin{aligned}
\underline{k}_{N+1} &= \frac{P_N \underline{x}_{N+1}^*}{1+\underline{x}_{N+1}^T P_N \underline{x}_{N+1}^*} && (9.42)\\
&= P_{N+1}\underline{x}_{N+1}^* && (9.43)\\
&= P_N \underline{x}_{N+1}^* \gamma(N+1). && (9.44)
\end{aligned}$$

In the matrix P_{N+1} all past values of the vectors $\underline{x}_k, k = 1..N$ are gathered. The scalar term $\gamma(N+1)$ is called *conversion factor*. We have:

$$\gamma(N+1) = \frac{1}{1+\underline{x}_{N+1}^T P_N \underline{x}_{N+1}^*}. \tag{9.45}$$

With such definition, we can also reformulate (9.38):

$$P_{N+1} = P_N - \frac{\underline{k}_{N+1}\underline{k}_{N+1}^H}{\gamma(N+1)};\ P_0 = \Pi_o. \tag{9.46}$$

A further interesting connection is given between the a-priori and the a-posteriori error, that is

$$\begin{aligned}
\tilde{e}_a(N+1) &= d(N+1) - \underline{x}_{N+1}^T \underline{\hat{w}}_N && (9.47)\\
\tilde{e}_p(N+1) &= d(N+1) - \underline{x}_{N+1}^T \underline{\hat{w}}_{N+1}. && (9.48)
\end{aligned}$$

By substituting the recursion (9.41) in the definition of the a-priori error, we obtain:

$$\tilde{e}_p(N+1) = \gamma(N+1)\tilde{e}_a(N+1). \tag{9.49}$$

As $\gamma(N+1)$ is strictly smaller than one, the a-posteriori error is always smaller than the a-priori error in magnitude.

A recursive formulation is also possible for the cost function g_{LS}

$$
\begin{aligned}
g_{LS}(\hat{\underline{w}}_{N+1}) &= \left[\underline{d}_N^H, d^*(N+1)\right] \begin{bmatrix} \underline{d}_N - X_N\hat{\underline{w}}_{N+1} \\ d(N+1) - \underline{x}_{N+1}^T\hat{\underline{w}}_{N+1} \end{bmatrix} && (9.50)\\
&= \left[\underline{d}_N^H, d^*(N+1)\right] \begin{bmatrix} \underline{d}_N - X_N\{\hat{\underline{w}}_N + \underline{k}_{N+1}\tilde{e}_a(N+1)\} \\ d(N+1) - \underline{x}_{N+1}^T\{\hat{\underline{w}}_N + \underline{k}_{N+1}\tilde{e}_a(N+1)\} \end{bmatrix} && (9.51)\\
&= \left[\underline{d}_N^H, d^*(N+1)\right] \begin{bmatrix} \underline{d}_N - X_N\hat{\underline{w}}_N - X_N\underline{k}_{N+1}\tilde{e}_a(N+1) \\ d(N+1) - \underline{x}_{N+1}^T\hat{\underline{w}}_N - \underline{x}_{N+1}^T\underline{k}_{N+1}\tilde{e}_a(N+1) \end{bmatrix} && (9.52)\\
&= \underline{d}_N^H[\underline{d}_N - X_N\hat{\underline{w}}_N] - \underline{d}_N^H X_N\underline{k}_{N+1}\tilde{e}_a(N+1) && (9.53)\\
&\quad + d^*(N+1)[d(N+1) - \underline{x}_{N+1}^T\hat{\underline{w}}_N] - d^*(N+1)\underline{x}_{N+1}^T\underline{k}_{N+1}\tilde{e}_a(N+1)\\
&= g_{LS}(\hat{\underline{w}}_N) + \tilde{e}_a(N+1)\left[d^*(N+1) - \underline{d}_{N+1}^H X_{N+1}\underline{k}_{N+1}\right] && (9.54)\\
&= g_{LS}(\hat{\underline{w}}_N) + \tilde{e}_a(N+1)\left[d^*(N+1) - \underline{d}_{N+1}^H X_{N+1}P_{N+1}\underline{x}_{N+1}^*\right] && (9.55)\\
&= g_{LS}(\hat{\underline{w}}_N) + \tilde{e}_a(N+1)\left[d(N+1) - \underline{x}_{N+1}^T\hat{\underline{w}}_{N+1}\right]^* && (9.56)\\
&= g_{LS}(\hat{\underline{w}}_N) + \tilde{e}_a(N+1)\tilde{e}_p^*(N+1) && (9.57)\\
&= g_{LS}(\hat{\underline{w}}_N) + |\tilde{e}_a(N+1)|^2\gamma(N+1). && (9.58)
\end{aligned}
$$

With the last substitution from (9.49) we also recognize that the conversion factor $\gamma(N+1)$ is real valued.

9.2.1 Underdetermined Forms

We already mention the APA (Affine Projection Algorithm) as a special solution to the recursive LS problem with the advantage that its cost function does not grow over the entire observation horizon but utilizes a sliding window of length N. To understand its adaptation better we generalize the algorithm first to the so-called ϵ-APA:

$$\hat{\underline{w}}_N(k) = \hat{\underline{w}}_N(k-1) + \mu\left(\epsilon I + X_N^H(k)X_N(k)\right)^{-1} X_N^H(k)\left[\underline{d}_N(k) - X_N(k)\hat{\underline{w}}_N(k-1)\right]. \quad (9.59)$$

The index N only indicates that the observation duration stretches over N elements in the past. The step-size μ and the *regularization parameter* ϵ are both assumed to be positive. A time index k is required now to distinguish the various terms. For the classical RLS algorithm this was not required as with growing window N also the time was defined.

Note that the inner matrix $X_N^H(k)X_N(k)$ is of dimension $M \times M$. Selecting an observation window $N < M$ results in an underdetermined matrix. Due to the positive regularization parameter $\epsilon > 0$ the modified matrix $\epsilon I + X_N^H(k)X_N(k)$ can be inverted.

However, it is not necessary to invert the matrix of large dimension $M \times M$ as we can apply the matrix inversion lemma:

$$\left(\epsilon I + X_N^H(k)X_N(k)\right)^{-1} X_N^H(k) = X_N^H(k)\left(\epsilon I + X_N(k)X_N^H(k)\right)^{-1}. \tag{9.60}$$

We thus have only to invert a matrix of dimension $N \times N$. The update equation of the ϵ-APA is thus given by:

$$\underline{\hat{w}}_N(k) = \underline{\hat{w}}_N(k-1) + \mu X_N^H(k)\left(\epsilon I + X_N(k)X_N^H(k)\right)^{-1}\left[\underline{d}_N(k) - X_N(k)\underline{\hat{w}}_N(k-1)\right]. \tag{9.61}$$

Two special cases are of particular interest and will be discussed next. The first case is for $N = 1$. We then obtain the $\epsilon-$LMS algorithm and recognize that it can be interpreted as a special case of the ϵ-APA with observation window length one.

The second special case is given for $\mu = 1$ and $\epsilon = 0$. If the matrix $X_N(k)X_N^H(k)$ is of full rank for every k, it can be inverted and the algorithm will converge. The dependency of the a-posteriori errors to the a-priori errors is now of interest. Considering N values in a vector we find:

$$\underline{\tilde{e}}_a(k) \triangleq \left[\underline{d}_N(k) - X_N(k)\underline{\hat{w}}_N(k-1)\right], \quad \underline{\tilde{e}}_p(k) \triangleq \left[\underline{d}_N(k) - X_N(k)\underline{\hat{w}}_N(k)\right]. \tag{9.62}$$

Substituting this in the update equation and we obtain the desired relation:

$$\underline{\tilde{e}}_p(k) = \left[I - X_N^H(k)(X_N(k)X_N^H(k))^{-1}X_N(k)\right]\underline{\tilde{e}}_a(k). \tag{9.63}$$

The matrix $I - X_N^H(k)(X_N(k)X_N^H(k))^{-1}X_N(k)$ is a so-called projection matrix. Consider a vector $\underline{z}$ comprising of two orthogonal components: a linear combination of $X_N^H(k)$ and a vector orthogonal to the subspace spanned by $X_N^H(k)$, thus $\underline{z} = \underline{y} + X_N^H(k)\underline{x}$. Multiplying the projection matrix from the right by such vector we recognize that it will become smaller by the amount of the linear combination in $X_N^H(k)$, thus $\underline{x}$ disappears. On the other hand, the orthogonal part $\underline{y}$ remains unchanged. We can further show that the a-priori error vector $\underline{\tilde{e}}_a(k)$ can be composed only by a linear combination in $X_N^H(k)$ and thus the a-posteriori error $\underline{\tilde{e}}_p(k) = \underline{0}$, must disappear.

With such property the cost function of the APA can also be described as:

$$\min \|\underline{\hat{w}}_N(k) - \underline{\hat{w}}_N(k-1)\| \text{ with the constraint } \underline{\tilde{e}}_p(k) = \underline{0}.$$

There exists an infinite amount of vectors $\underline{\hat{w}}_N(k)$, who solve $\underline{\tilde{e}}_p(k) = \underline{0}$ (all with different orthogonal part $\underline{y}$). This set of solutions will be denoted as affine subspace (also: hyperplane, manifold) to indicate that the plane defined by the set of solutions is not necessarily passing $\underline{\hat{w}}_N(k) = \underline{0}$. For the special case $N = 1$ we say that the APA (NLMS algorithm with normalized step-size $\alpha = 1$) obtains the solution $\underline{\hat{w}}_N(k)$ by a projection

with respect to the affine subspace. For $N > 1$ the solution is the **intersection** of all these affine subspaces. The APA thus finds its solution by projection onto the **intersection** of all affine subspaces. Note that such projection properties get lost for the overdetermined case $N > M$, thus for the RLS algorithm.

Exercise 9.6 *Derive the recursive form of the LS algorithm with sliding rectangular window (9.26).*

Exercise 9.7 *Derive the recursive form of the LS algorithm with exponential window :*

$$g_{LS}(\underline{\hat{w}}) = \sum_{i=1}^{N} \lambda^{N-i} |\tilde{e}_a(i)|^2 = \sum_{i=1}^{N} \lambda^{N-i} |d(i) - \underline{x}_i^T \underline{\hat{w}}|^2. \tag{9.64}$$

Find the cost function as function of a-priori- and a-posteriori error.

Exercise 9.8 *Is it possible to derive for the LMS algorithm a relation like in (9.41) between a-priori- and a-posteriori error?*

Exercise 9.9 *Derive the LMS algorithm from the Steepest-Descent algorithm for the following estimates:*

$$\hat{R}_{\underline{\mathbf{x}}\underline{\mathbf{x}}} = \frac{1}{N} \sum_{l=0}^{N-1} \underline{x}_{l-k}^* \underline{x}_{l-k}^T \tag{9.65}$$

$$\hat{\underline{r}}_{\underline{\mathbf{x}}\mathbf{d}}^* = \frac{1}{N} \sum_{l=0}^{N-1} \underline{x}_{l-k}^* d(l-k). \tag{9.66}$$

Assume for this derivation the driving process $\mathbf{x}(k)$ *to be stationary. Under which step-size condition do you obtain the RLS algorithm?*

9.3 Stationary Behavior

Similar to the LMS algorithm there is also key performance parameters for the RLS algorithm describing its behavior in stationary environments, that is its learning speed and steady-state precision. We consider the reference model from Lemma 9.2:

$$\underline{d}_N = X_N \underline{w}_o + \underline{v}_N. \tag{9.67}$$

As the LS problem (same goes for its recursive form) is solved in form of a set of linear equations of order M, a stationary solution will exist after $N = M$ steps, whose value only alternates further with the noise terms. If applying the recursive form, we typically start with zero vectors $\underline{x}_k$ (initialized by zero entries) and thus for the first M steps, the algorithm cannot work properly. We thus require $2M$ steps from starting with zero to the convergence of the algorithm. This is a substantial speed-up compared to the LMS algorithm even compared to the Newton-LMS algorithm for which the RLS algorithm can be viewed as approximation: $P^{-1} \approx R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}$. Similar to the Newton-LMS algorithm the learning speed is independent of the driving sequence correlation.

To compute the steady-state misadjustment, we consider the RLS algorithm with exponential decaying window as its most common form. Its update is given by:

$$\underline{\hat{w}}_k = \underline{\hat{w}}_{k-1} + \underline{k}_k[d(k) - \underline{x}_k^T\underline{\hat{w}}_{k-1}], \tag{9.68}$$

$$\underline{k}_k = \frac{\lambda^{-1}P_{k-1}\underline{x}_k^*}{1+\lambda^{-1}\underline{x}_k^T P_{k-1}\underline{x}^*}, \tag{9.69}$$

$$P_k = \lambda^{-1}[P_{k-1} - \underline{k}_k\underline{x}_k^T P_{k-1}]. \tag{9.70}$$

Note that we changed notation slightly: instead of an index N that denoted time as well as growing size before, the index now denotes the time instant k. On the one hand we like to point out by this the formal similarity to the LMS algorithm but on the other hand we also like to point out that there is no growing window of size N any more.

We consider the additive noise as random process by which the estimate $\underline{\hat{\mathbf{w}}}_k$ now also becomes a random process. From the last line in (9.25) and the reference model we find for $\Pi^{-1} = 0$, $Q = \Lambda$ and arbitrary $\sigma_{\mathbf{v}}^2$

$$\lim_{k\to\infty} \mathrm{E}_{\mathbf{v}}[(\underline{w}_o - \underline{\hat{\mathbf{w}}}_k)(\underline{w}_o - \underline{\hat{\mathbf{w}}}_k)^H] = \lim_{k\to\infty}[X_{k-1}^H\Lambda X_{k-1}]^{-1}[X_{k-1}^H\Lambda^2 X_{k-1}][X_{k-1}^H\Lambda X_{k-1}]^{-1}\sigma_{\mathbf{v}}^2. \tag{9.71}$$

The diagonal entries of matrix Λ are $\Lambda_{ii} = \lambda^{k-i}$. Correspondingly the driving process $\mathbf{x}(k)$ can be viewed as random process which allows to compute (approximately) expectations in $\mathbf{X}_{k-1}$:

$$\begin{aligned}\mathrm{E}_{\mathbf{x}}\left[[\mathbf{X}_{k-1}^H\Lambda\mathbf{X}_{k-1}]^{-1}[\mathbf{X}_{k-1}^H\Lambda^2\mathbf{X}_{k-1}][\mathbf{X}_{k-1}^H\Lambda\mathbf{X}_{k-1}]^{-1}\right] &\approx \left(R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}\sum_{i=1}^{k}\lambda^{k-i}\right)^{-1} \\ &\quad \left[R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}\sum_{i=1}^{k}\lambda^{2k-2i}\right]\left(R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}\sum_{i=1}^{k}\lambda^{k-i}\right)^{-1}.\end{aligned} \tag{9.72}$$

This term can be further simplified to

$$\lim_{k\to\infty} \mathrm{E}[(\underline{w}_o - \underline{\hat{\mathbf{w}}}_k)(\underline{w}_o - \underline{\hat{\mathbf{w}}}_k)^H] \approx \frac{1-\lambda}{1+\lambda}R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}^{-1}\sigma_{\mathbf{v}}^2. \tag{9.73}$$

We can thus compute the mismatch of the parameter error vector

$$\lim_{k\to\infty} \mathrm{tr}\left[\mathrm{E}[(\underline{w}_o - \underline{\hat{\mathbf{w}}}_k)(\underline{w}_o - \underline{\hat{\mathbf{w}}}_k)^H]\right] \approx \frac{1-\lambda}{1+\lambda}\mathrm{tr}[R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}^{-1}]\sigma_{\mathbf{v}}^2 \tag{9.74}$$

$$= \frac{1-\lambda}{1+\lambda}\sigma_{\mathbf{v}}^2 \sum_{i=1}^{M} \frac{1}{\lambda_i} \tag{9.75}$$

The steady-state value of the a-priori error is found to be (assuming $\sigma_{\mathbf{x}}^2 = 1$):

$$\lim_{k\to\infty} \mathrm{E}[|\tilde{\mathbf{e}}_a(k)|^2] = \sigma_{\mathbf{v}}^2 + \lim_{k\to\infty} \mathrm{tr}\left[\mathrm{E}[(\underline{w}_o - \underline{\hat{\mathbf{w}}}_k)(\underline{w}_o - \underline{\hat{\mathbf{w}}}_k)^H] R_{\underline{\mathbf{x}}\underline{\mathbf{x}}}\right] \tag{9.76}$$

$$= \sigma_{\mathbf{v}}^2 \left(1 + M\frac{1-\lambda}{1+\lambda}\right) \tag{9.77}$$

and thus the misadjustment

$$m_{LS} = M\frac{1-\lambda}{1+\lambda}. \tag{9.78}$$

Note that for real- and complex-valued Gaussian processes more precise expressions exist, as the terms can be modeled as Wishart process. Typically the provided expressions are valid for dimensions of $M \geq 10$.

Matlab Experiment 4.1 Repeat Matlab Experiments 3.1 and 3.2, however with exponentially weighted RLS algorithm. Instead of various step-sizes, apply forgetting factors λ in the range $[0.7, 1.0]$. Compare experimental results with theoretical predictions.

Exercise 9.10 *Compute the exact expression of (9.72) in case the forgetting factor is one. Assume the driving process to be a zero-mean Gaussian process. What is then obtained for misadjustment and mismatch?*

9.4 Literature

Good tutorials on LS and RLS algorithms can be found in [33]. Polyphase implementations of FTF versions are explained in [27]. Detailed descriptions to various implementations are in [71]. Details to CORDIC implementations are in [3].

Chapter 10

Tracking behavior of Adaptive Systems

Until now our assumption was that the system under consideration is time-invariant, thus a fixed $\underline{w}_o$. But not all systems are fixed. Due to aging and temperature properties of systems alter slowly. Some systems change fast: the loudspeaker-room-microphone system changes rapidly if the speaker moves through the room. Also the wireless channel may change rapidly with moving receiver or moving scattering objects. Next to the initial learning or transient response there is also a tracking (Ger: Nachführverhalten) behavior. This describes how an adaptive filter reacts if the system is permanently changing. One possibility to describe such behavior in a general form, is to assume a rotational change, as described in the following:

$$d(k) = \underline{x}_k^T \underline{w}_o e^{j\Omega_o k} + v(k) \,. \tag{10.1}$$

The system $\underline{w}_o$ that is to be estimated now rotates with an unknown frequency Ω_o. A direct application of this formulation is given in the wireless channel estimation under frequency offset. As the receiver utilizes a different oscillator than the transmitter, a frequency offset Ω_o occurs. We will recognize that the reaction of the adaptive system of such a rotation is typically of linear nature. This means that the reaction of an arbitrarily changing time-variant system can be treated as the superposition of individual rotational components. It is thus sufficient to analyze the behavior for a single rotation at frequency Ω_o.

10.1 Tracking Behavior of LMS and RLS Algorithm

With the reference model (10.1) the error vector can be defined anew:

$$\underline{\tilde{w}}_k \triangleq \underline{w}_o e^{j\Omega_o k} - \underline{\hat{w}}_k \,. \tag{10.2}$$

Thus we obtain for the a-priori error:

$$\begin{aligned}
\tilde{e}_a(k) &= d(k) - \underline{x}_k^T \hat{\underline{w}}_{k-1} && (10.3)\\
&= v(k) + \underline{x}_k^T \underline{w}_o e^{j\Omega_o k} - \underline{x}_k^T \hat{\underline{w}}_{k-1} \\
&= v(k) + \underline{x}_k^T \underline{w}_o e^{j\Omega_o (k-1)} - \underline{x}_k^T \hat{\underline{w}}_{k-1} + \underline{x}_k^T \underline{w}_o e^{j\Omega_o k} - \underline{x}_k^T \underline{w}_o e^{j\Omega_o (k-1)} \\
&= v(k) + \underline{x}_k^T \tilde{\underline{w}}_{k-1} + \underline{x}_k^T \underline{w}_o e^{j\Omega_o k} \left(1 - e^{-j\Omega_o}\right). && (10.4)
\end{aligned}$$

The update equations for the LMS and RLS algorithm can be provided in a unified form:

$$\tilde{\underline{w}}_k = (I - \underline{g}_k^* \underline{x}_k^T)\tilde{\underline{w}}_{k-1} - v(k)\underline{g}_k^* + (I - \underline{g}_k^* \underline{x}_k^T)\underline{w}_o e^{j\Omega_o k}(1 - e^{-j\Omega_o}). \tag{10.5}$$

The vector $\underline{g}_k$ is simply $\mu \underline{x}_k$ for the LMS algorithm and $\underline{g}_k = P_k \underline{x}_k$ in case of the RLS, thus

$$\underline{g}_k = \begin{cases} \mu \underline{x}_k & ;\text{LMS} \\ P_k \underline{x}_k & ;\text{RLS} \end{cases}. \tag{10.6}$$

In the next step, we consider the signals $v(k)$ and $\underline{x}_k$ as random processes, thus $\mathbf{v}(k)$ and $\underline{\mathbf{x}}_k$. We can now compute the expectation with respect to the driving source:

$$\mathrm{E}[\tilde{\underline{\mathbf{w}}}_k] = (I - A)\mathrm{E}[\tilde{\underline{\mathbf{w}}}_{k-1}] + (I - A)\underline{w}_o e^{j\Omega_o k}(1 - e^{-j\Omega_o}). \tag{10.7}$$

The matrix A is given by $\mu R^*_{\underline{\mathbf{xx}}}$ in case of the LMS and $[1 - \lambda]I$ in case of the RLS algorithm with exponentially decaying window.

Theorem 10.1 *The stationary solution for the LMS and RLS algorithm for a system that changes periodically with frequency Ω_o in the mean is given by:*

$$E[\hat{\underline{\mathbf{w}}}_k] = \left\{ I - (e^{j\Omega_o} - 1)\left[e^{j\Omega_o} I - (I - A)\right]^{-1} (I - A) \right\} \underline{w}_o e^{j\Omega_o k}. \tag{10.8}$$

Proof: As $\mathrm{E}[\tilde{\underline{\mathbf{w}}}_k]$ is an output of a linear system, we must have a solution of the form:

$$E[\tilde{\underline{\mathbf{w}}}_k] = \underline{a} e^{j\Omega_o (k+1)}. \tag{10.9}$$

The solution for $\underline{a}$ is found by the substitution

$$\underline{a} = (1 - e^{-j\Omega_o})\left[e^{j\Omega_o} I - (I - A)\right]^{-1} (I - A)\underline{w}_o. \tag{10.10}$$

Thus, the expectation of the parameter error vector also becomes periodically time variant. For $k \to \infty$ the initial transients disappear and the mean parameter error vector becomes eventually

$$\mathrm{E}[\tilde{\underline{\mathbf{w}}}_k] = (1 - e^{-j\Omega_o})\left[e^{j\Omega_o} I - (I - A)\right]^{-1} (I - A)\underline{w}_o e^{j\Omega_o (k+1)}, \tag{10.11}$$

or, equivalently in form of the mean estimator:

$$\mathrm{E}[\underline{\hat{\mathbf{w}}}_k] = \left\{ I - (e^{j\Omega_o} - 1) \left[e^{j\Omega_o} I - (I - A) \right]^{-1} (I - A) \right\} \underline{w}_o e^{j\Omega_o k} \,. \tag{10.12}$$

□

Obviously frequency Ω_o as well as algorithmic specific parameters like μ and λ influence the result. Essentially we can state that the estimate runs behind the true value $\underline{w}_o e^{j\Omega_o k}$ in magnitude (smaller) and in phase.

We can formulate the result (10.12) for an arbitrarily small frequency range $d\Omega$:

$$d\mathrm{E}[\underline{\hat{\mathbf{w}}}_k(\Omega)] = \left\{ I - (e^{j\Omega} - 1) \left[e^{j\Omega} I - (I - A) \right]^{-1} (I - A) \right\} \underline{w}_o(\Omega) e^{j\Omega k} d\Omega.$$

This interpretation allows the computation of the algorithmic response to arbitrary system changes:

$$\mathrm{E}[\underline{\hat{\mathbf{w}}}_k] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \left\{ I - (e^{j\Omega} - 1) \left[e^{j\Omega} I - (I - A) \right]^{-1} (I - A) \right\} \underline{w}_o(\Omega) e^{j\Omega k} d\Omega \,. \tag{10.13}$$

The kernel of this integral is the Fourier-transform of the algorithmic response or also-called the Fourier-transform G of the Green's function of the LMS/RLS algorithm:

$$G(\Omega) \triangleq \left\{ I - (e^{j\Omega} - 1) \left[e^{j\Omega} I - (I - A) \right]^{-1} (I - A) \right\} \,.$$

This *Green function in the mean* is thus obtained by the inverse Fourier-transform:

$$g(k) = I - \left[(I - A)^k u(k) - (I - A)^{(k-1)} u(k-1) \right]$$

with the step-response $u(k)$. In other words: the algorithmic response to arbitrarily changing systems can be computed by a convolution with the Green function $g(k)$. Two well-known results can be obtained by this:

- In case of a frequency offset we find: $\underline{w}_o(\Omega) = \underline{w}_o \delta(\Omega - \Omega_o)$ and we obtain:

$$\mathrm{E}[\underline{\hat{\mathbf{w}}}_k] = \left\{ I - (e^{j\Omega_o} - 1) \left[e^{j\Omega_o} I - (I - A) \right]^{-1} (I - A) \right\} \underline{w}_o e^{j\Omega_o k}.$$

- In the initial phase of the adaptation we have: $\underline{w}_o(\Omega) = \underline{w}_o / [1 - e^{-j\Omega}]$ and obtain

$$\mathrm{E}[\underline{\hat{\mathbf{w}}}_k] = \left(I - [I - A]^{k+1} \right) \underline{w}_o.$$

Theorem 10.2 *Under white excitation the LMS and RLS algorithm show identical tracking behavior in the mean.*

Proof: We obtain $A = \mu I$ for the LMS algorithm and $[1-\lambda]I$ for the RLS. In other words the choice $\mu = 1 - \lambda$ results in the same tracking behavior.□

Matlab Experiment 5.1 A system $\underline{w}_o^T = [1, 10, 1]$ excited by a white sequence is changing periodically with frequency Ω_o. Compute the algorithmic response of the LMS and the RLS algorithm as a function of the frequency. Simulate this in Matlab and verify your result. Plot the relative parameter error vector for a range of Ω over μ and $1 - \lambda$. How can $\underline{\hat{\mathbf{w}}}_k$ be used to estimate the unknown frequency Ω?

10.2 Kalman Algorithm

The analysis of LMS and RLS algorithm has shown that the adaptive filter solution lags behind the true value. Although the choice of step-size and forgetting factor impact the solution, even at optimal values the lag increases with higher frequency. If there exists a-priori knowledge about the systems changes, such knowledge can be brought into the design of the adaptive algorithm. Let us assume a general system in state-space form:

$$\underline{w}_k = F_k \underline{w}_{k-1} + G_k \underline{u}_k, \; k = 1, 2, ... \tag{10.14}$$

$$\underline{d}_k = X_k \underline{w}_{k-1} + \underline{v}_k. \tag{10.15}$$

The unknown system $\underline{w}_k$ varies according to (10.14), driven by the signal $\underline{u}_k$. The output of the system is a linear combination of system and input X_k with an additional noise term $\underline{v}_k$. In the general case also the output values are vectors. Such systems are often called Multiple Input-Multiple Output (MIMO).

In order to estimate such a system $\underline{w}_k$ the adaptive algorithm should reflect as much of the a-priori knowledge as possible, for example the state-space form. In the recursive update part of the adaptive algorithm, we introduce a prediction component $F_k \underline{\hat{w}}_k$. As we have errors now in vector form, we have to introduce an optimal matrix step-sise M_k. The optimal adaptation algorithm is thus given by:

$$\underline{\tilde{e}}_{a,k} = \underline{d}_k - X_k \underline{\hat{w}}_{k-1} \tag{10.16}$$

$$\underline{\hat{w}}_k = F_k \underline{\hat{w}}_{k-1} + M_k \underline{\tilde{e}}_{a,k}, \; k = 1, 2, ... \tag{10.17}$$

The only open problem is now to find the optimal step-size matrix M_k.

For this we assume that the signals $\underline{u}_k$ and $\underline{v}_k$ are random processes. We will further assume that

$$\mathrm{E}[\underline{\mathbf{v}}_k \underline{\mathbf{v}}_i^H] = R_{\underline{\mathbf{vv}}} \delta(k-i); \quad \mathrm{E}[\underline{\mathbf{u}}_k \underline{\mathbf{u}}_i^H] = R_{\underline{\mathbf{uu}}} \delta(k-i); \quad \mathrm{E}[\underline{\mathbf{v}}_k \underline{\mathbf{u}}_i^H] = 0. \tag{10.18}$$

Further an initial state $\underline{\mathbf{w}}_0$ is assumed as zero-mean random variable with

$$\mathrm{E}[\underline{\mathbf{w}}_0\underline{\mathbf{v}}_i^H] = 0; \quad \mathrm{E}[\underline{\mathbf{w}}_0\underline{\mathbf{u}}_i^H] = 0; \quad \mathrm{E}[\underline{\mathbf{w}}_0\underline{\mathbf{w}}_0^H] = P_0. \tag{10.19}$$

All conditions can compactly formulated as:

$$\mathrm{E}\left(\begin{bmatrix} \underline{\mathbf{u}}_k \\ \underline{\mathbf{v}}_k \\ \underline{\mathbf{w}}_0 \\ 1 \end{bmatrix} \begin{bmatrix} \underline{\mathbf{u}}_i \\ \underline{\mathbf{v}}_i \\ \underline{\mathbf{w}}_0 \end{bmatrix}^H\right) = \begin{bmatrix} R_{\underline{\mathbf{uu}}}\delta(k-i) & 0 & 0 \\ 0 & R_{\underline{\mathbf{vv}}}\delta(k-i) & 0 \\ 0 & 0 & P_0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{10.20}$$

In order to ensure a unique solution we also have to assume that $R_{\underline{\mathbf{vv}}}$ is positive definite, a condition that is in general satisfied. Note that other more relaxed conditions are possible as well. The solution only becomes then more and more difficult to interpret.

Let us now consider the random a-priori error vector $\tilde{\underline{\mathbf{e}}}_{a,k}$:

$$\tilde{\underline{\mathbf{e}}}_{a,k} = \underline{\mathbf{d}}_k - X_k\hat{\underline{\mathbf{w}}}_{k-1} = X_k\tilde{\underline{\mathbf{w}}}_{k-1} + \underline{\mathbf{v}}_k. \tag{10.21}$$

If we define the covariance matrix of the parameter error vector as

$$P_k \triangleq \mathrm{E}[\tilde{\underline{\mathbf{w}}}_k\tilde{\underline{\mathbf{w}}}_k^H], \tag{10.22}$$

then we can also compute the covariance matrix of the a-priori error vector:

$$\mathrm{E}[\tilde{\underline{\mathbf{e}}}_{a,k}\tilde{\underline{\mathbf{e}}}_{a,k}^H] = R_{\underline{\mathbf{vv}}} + X_kP_{k-1}X_k^H = R_{\underline{\mathbf{ee}}}. \tag{10.23}$$

The optimal step-size matrix can be found by minimizing the recursion of the covariance matrix with respect to M_k:

$$P_k = F_kP_{k-1}F_k^H + M_k\mathrm{E}[\tilde{\underline{\mathbf{e}}}_{a,k}\tilde{\underline{\mathbf{e}}}_{a,k}^H]M_k^H + G_kR_{\underline{\mathbf{uu}}}G_k^H - F_k\mathrm{E}[\tilde{\underline{\mathbf{w}}}_{k-1}\tilde{\underline{\mathbf{e}}}_{a,k}^H]M_k^H - M_k\mathrm{E}[\tilde{\underline{\mathbf{e}}}_{a,k}\tilde{\underline{\mathbf{w}}}_{k-1}^H]F_k^H. \tag{10.24}$$

The minimization w.r.t M_k is obtained by comparison with the optimal form $(M_k - \bar{M}_k)B(M_k - \bar{M}_k)^H$. We find $B = R_{\underline{\mathbf{ee}}}$ and

$$\begin{aligned} \bar{M}_k &= F_k\mathrm{E}[\tilde{\underline{\mathbf{w}}}_{k-1}\tilde{\underline{\mathbf{e}}}_{a,k}^H]R_{\underline{\mathbf{ee}}}^{-1} && (10.25)\\ &= F_k\mathrm{E}[\tilde{\underline{\mathbf{w}}}_{k-1}(X_k\tilde{\underline{\mathbf{w}}}_{k-1} + \underline{\mathbf{v}}_k)^H]R_{\underline{\mathbf{ee}}}^{-1} && (10.26)\\ &= F_kP_{k-1}X_k^HR_{\underline{\mathbf{ee}}}^{-1}. && (10.27) \end{aligned}$$

Having this available we can now formulate the equation for the parameter covariance matrix:

$$P_k = F_kP_{k-1}F_k^H - \bar{M}_kR_{\underline{\mathbf{ee}}}\bar{M}_k^H + G_kR_{\underline{\mathbf{uu}}}G_k^H. \tag{10.28}$$

Eventually the complete Kalman algorithm is obtained.

Kalman Algorithm: Consider the state space description of a time-variant system:

$$\underline{\mathbf{w}}_k = F_k \underline{\mathbf{w}}_{k-1} + G_k \underline{\mathbf{u}}_k \tag{10.29}$$
$$\underline{\mathbf{d}}_k = X_k \underline{\mathbf{w}}_{k-1} + \underline{\mathbf{v}}_k. \tag{10.30}$$

with conditions

$$\mathrm{E}\left(\begin{bmatrix} \underline{\mathbf{u}}_k \\ \underline{\mathbf{v}}_k \\ \underline{\mathbf{w}}_0 \\ 1 \end{bmatrix} \begin{bmatrix} \underline{\mathbf{u}}_i \\ \underline{\mathbf{v}}_i \\ \underline{\mathbf{w}}_0 \end{bmatrix}^H \right) = \begin{bmatrix} R_{\underline{\mathbf{uu}}}\delta(k-i) & 0 & 0 \\ 0 & R_{\underline{\mathbf{vv}}}\delta(k-i) & 0 \\ 0 & 0 & P_0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{10.31}$$

The optimal estimator for the system $\underline{\mathbf{w}}_k$ is given by:

$$\bar{M}_k = F_k P_{k-1} X_k^H [R_{\underline{\mathbf{vv}}} + X_k P_{k-1} X_k^H]^{-1}, \tag{10.32}$$
$$\underline{\hat{\mathbf{w}}}_k = F_k \underline{\hat{\mathbf{w}}}_{k-1} + \bar{M}_k \underline{\tilde{\mathbf{e}}}_{a,k}, \tag{10.33}$$
$$P_k = F_k P_{k-1} F_k^H - \bar{M}_k [R_{\underline{\mathbf{vv}}} + X_k P_{k-1} X_k^H] \bar{M}_k^H + G_k R_{\underline{\mathbf{uu}}} G_k^H. \tag{10.34}$$

Note that the application of the Kalman algorithm requires the assumption of random signals. This was not the case for the previous algorithms. In the LMS algorithm we needed the statistic only for finding optimal step-size parameters, and to compute its properties. The RLS worked entirely without randomness. We should further mention that the complexity of the Kalman algorithm is given by $O(M^3)$ as we have to invert a matrix. Only in very simple cases this can be avoided. Classically the Kalman algorithm is found in automation control where the time variant nature of the system is somewhat known and there is sufficient time to compute the algorithmic equations between to observation samples. In the last years more applications included satellite communications and adaptive equalizers in wireless systems. Often the time-variant behavior is only partly known or only in approximation. Then next to the Kalman equations also estimates for the system parameters are required, for example for matrix F_k. This is called the extended Kalman filter. In automation control the term LQC=linear, quadratic control is being used to describe that the controler is linear and the cost function quadratic. Here the actuating variable $\{\underline{u}_k\}$ in (10.29) is to tune so that

$$g_{LQC}(\underline{u}_{N+1}) = \underline{w}_{N+1}^H P_{N+1} \underline{w}_{N+1} + \sum_{k=1}^{N} \underline{u}_k^H R_{\underline{\mathbf{vv}}} \underline{u}_k + \sum_{k=1}^{N} \underline{d}_k^H R_{\underline{\mathbf{dd}}} \underline{d}_k \tag{10.35}$$

becomes minimal. Newer research is found under the name Model Predictive Control. It can be shown that the substitution of F_k with F_k^*, H_k^* with $-G_k$ and G_k^* with H_k results in the Kalman algorithm as solution. These two problems are said to be dual.

Exercise 10.1 *Derive the Kalman algorithm under the condition*

$$E\left(\begin{bmatrix} \underline{\mathbf{u}}_k \\ \underline{\mathbf{v}}_k \\ \underline{\mathbf{w}}_0 \\ 1 \end{bmatrix} \begin{bmatrix} \underline{\mathbf{u}}_i \\ \underline{\mathbf{v}}_i \\ \underline{\mathbf{w}}_0 \end{bmatrix}^H\right) = \begin{bmatrix} R_{\underline{\mathbf{uu}}}\delta(k-i) & R_{\underline{\mathbf{uv}}}\delta(k-i) & 0 \\ R_{\underline{\mathbf{vu}}}\delta(k-i) & R_{\underline{\mathbf{vv}}}\delta(k-i) & 0 \\ 0 & 0 & P_0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{10.36}$$

Exercise 10.2 *For the special case* $F_k = I, G_k = 0, R_{\mathbf{vv}} = 1$ *and* $X_k = \underline{x}_k^T$ *derive the Kalman equations and compare them with the RLS algorithm.*

Matlab Experiment 5.2 Repeat Matlab Experiment 5.1 and apply now the Kalman algorithm. Compare the results with the previous ones.

10.3 Literature

First publications to the tracking behavior of adaptive algorithms are found [17, 18, 28, 49]. A good introduction to Kalman filters is given in [33] and [38, 71]. The original paper from Kalman is [39]. In [2] Applications of the algorithm are described.

Chapter 11

Unsupervised Classification Algorithms

We will now turn our interest onto the problem of identifying multiple categories. See for example Figure 11.1 in which four categories are shown with attributes "blue" and "red" as well as "round" and "squared". In this example two lines separate nicely all four categories, which can be found with the methods described so far. In general the problem of finding several categories is not as simple and often the number of categories is not even known before hand.

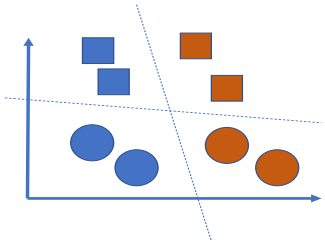

Figure 11.1: *Finding multiple categories.*

11.1 Lloyd's Algorithm

The basic concept of finding categories goes back to Lloyd who worked on optimal quantization. Note that quantization is a special case of classification as you assign a range

of input values to a single output value, typically something like the center value. Lloyd originally formulated the idea back in 1957 when working at Bell labs but as he didn't have a computer at the time he was not able to practically test it. Finally once he was retired he managed to get hands onto a computer and published it in 1982. The basic idea is illustrated in Figure 11.2 below. The input signal x is to divide into $M+1$ subsets each with a lower limitation at t_q and a upper one at t_{q+1}, which serves as the lower limitation for the next one. All x values between t_q and t_{q+1} are represented by a value $\hat{x}_q$. We now need to find the M thresholds $t_1, t_2, ..., t_M$ as well as the optimal quantization levels $\hat{x}_q$. Note that there are M thresholds defining $M-1$ sets in between them plus two subsets that are below the lowest and above the highest threshold, thus $M+1$ quantization levels $\hat{x}_q, q = 0, 1, .., M$.. We can this formulate our problem in a LS form:

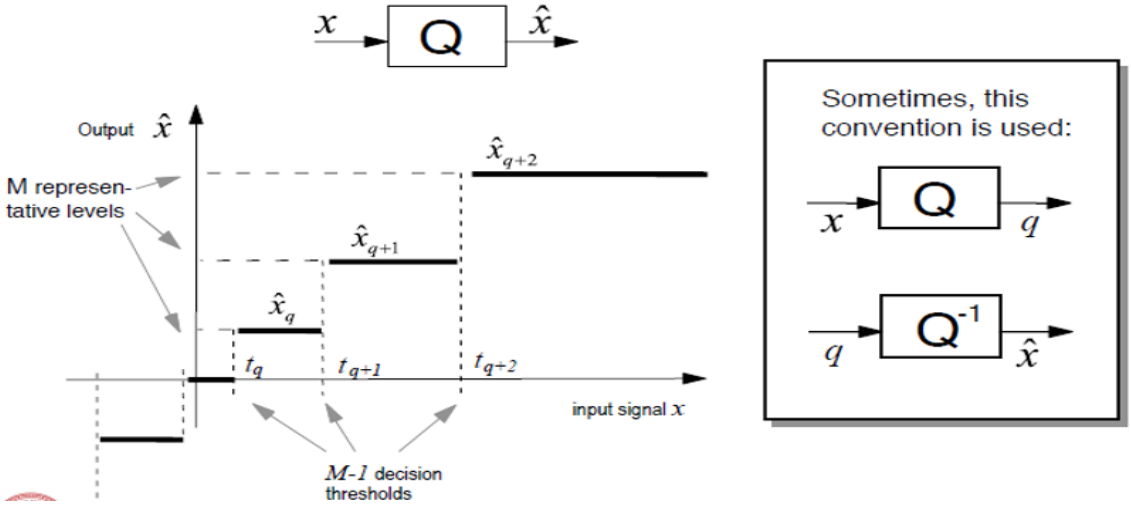

Figure 11.2: *Quantisation device.*

$$\begin{aligned}\min_{\hat{x}_q, t_l} \quad & \int_{-\infty}^{t_1} (x-\hat{x}_0)^2 f_{\mathbf{x}}(x)dx + \\ + \quad & \int_{t_1}^{t_2} (x-\hat{x}_1)^2 f_{\mathbf{x}}(x)dx + \\ & \vdots \\ + \quad & \int_{t_{M-1}}^{t_M} (x-\hat{x}_{M-1})^2 f_{\mathbf{x}}(x)dx + \\ + \quad & \int_{t_M}^{\infty} (x-\hat{x}_M)^2 f_{\mathbf{x}}(x)dx.\end{aligned}$$

Minimizing this expression with respect to the quantization levels x_q is straight forward and thus our next step. We find the conditions:

$$\int_{t_q}^{t_{q+1}} (x - \hat{x}_q) f_{\mathbf{x}}(x) dx = 0; q = 1, 2, ..., M,$$

which immediately delivers

$$\hat{x}_q = \frac{\int_{t_q}^{t_{q+1}} x f_{\mathbf{x}}(x) dx}{\int_{t_q}^{t_{q+1}} f_{\mathbf{x}}(x) dx}; q = 1, 2, ..., M-1.$$

The first and last quantization level can be computed accordingly. This can be interpreted as the ratio of the mean over the probability that that we are in this subsection. Finding the thresholds is much more complicated. Lloyd proposed a necessary condition

$$t_q = \frac{\hat{x}_{q-1} + \hat{x}_q}{2}; q = 1, 2, ..., M,$$

which unfortunately is not sufficient to make it a unique solution. When researchers adopted this idea in image processing they were very happy that the solution is not unique as this provided them with sufficient freedom for their optimization problems. The entire algorithm is then of iterative nature:

1. Guess initial set of representative levels $\hat{x}_q$. (equidistant?)

2. Calculate M decision thresholds

$$t_q = \frac{\hat{x}_{q-1} + \hat{x}_q}{2}; q = 1, 2, ..., M.$$

3. Calculate $M + 1$ new representative levels and probabilities

$$\hat{x}_q = \frac{\int_{t_q}^{t_{q+1}} x f_{\mathbf{x}}(x) dx}{\int_{t_q}^{t_{q+1}} f_{\mathbf{x}}(x) dx}; q = 1, 2, ..., M-1.$$

   as well as the two on the boundary

$$\hat{x}_0 = \frac{\int_{-\infty}^{t_1} x f_{\mathbf{x}}(x) dx}{\int_{-\infty}^{t_1} f_{\mathbf{x}}(x) dx},$$

   and

$$\hat{x}_M = \frac{\int_{t_M}^{\infty} x f_{\mathbf{x}}(x) dx}{\int_{t_M}^{\infty} f_{\mathbf{x}}(x) dx}.$$

4. Compute new MSE.

5. Repeat 2 and 3 until MSE is no longer getting smaller .

Let's look at a simple example. Suppose we have a symmetric pdf, a uniform distribution between -1 and +1. We select $M = 2$ that is two thresholds with three quantization levels. Due to the symmetry we select as a start $\hat{x}_0 = -\hat{x}_2$ and $t_1 = -t_2$. This simplifies the problem so that we can compute the solution explicitly. The MSE reads

$$\int_{-1}^{t_1}(x-\hat{x}_o)^2\frac{1}{2}dx + \int_{t_1}^{-t_1}(x-\hat{x}_1)^2\frac{1}{2}dx + \int_{-t_1}^{1}(x+\hat{x}_o)^2\frac{1}{2}dx$$

which we find to be proportional to

$$(t_1-\hat{x}_0)^3 + (1+\hat{x}_0)^3 - t_1(t_1^2+3\hat{x}_1^2).$$

Differentiating w.r.t. $\hat{x}_1$ directly delivers us $\hat{x}_1 = 0$ which simplifies the expression to

$$(t_1-\hat{x}_0)^3 + (1+\hat{x}_0)^3 - t_1^3.$$

Further differentiating with respect to t_1 delivers that $\hat{x}_o = 2t_1$ and finally we find $t_1 = -\frac{1}{3}$, so that $\hat{x}_0 = -\frac{2}{3}$.

11.2 K Means Algorithm

The term "k-means" was first used by James MacQueen in 1967, though the idea goes back to Hugo Steinhaus in 1957. The standard algorithm was first proposed by Stuart Lloyd in 1957 as a technique for pulse-code modulation, though it wasn't published outside of Bell Labs until 1982. Llyods algorithm provides the basic concept not only for quantization but also for classification. It can be extended from one dimensional ranges into multiple. In 1965, E. W. Forgy published essentially the same method, which is why it is sometimes referred to as Lloyd-Forgy. The problem is to find k clusters, defined by their means (centroids), given a large amount of N data points (feature vectors). In that way, the entire set S of data points is split into k disjoint subsets, with not necessarily identical amounts of elements. Although the algorithm converges fast, it is not necessarily finding the optimal solution. Its solution depends on the starting values A good starting set is to select k points with relatively large distance from each other. We can formulate the algorithm as

1. Find subsets $\mathcal{S}_i$ with means $\underline{\bar{x}}_i; i = 1, 2, ..., k$ in set $\mathcal{S}$:

$$\arg\min_{\mathcal{S}} \sum_{i=1}^{k} \sum_{\underline{x}_j \in \mathcal{S}_i} ||\underline{x}_j - \underline{\bar{x}}_i||^2 = \arg\min_{\mathcal{S}} \sum_{i=1}^{k} |\mathcal{S}_i| \mathrm{var}(\mathcal{S}_i).$$

2. Simple search algorithm by testing all data points $\underline{x}_j; j = 1, 2, ..., N$

3. Test for all clusters $i = 1, 2, ..., k$ and all data points $\underline{x}_j; j = 1, 2, ..., N$.
   $\underline{x}_j$ belongs to $\mathcal{S}_i$ if

$$||\underline{x}_j - \bar{\underline{x}}_i||^2 < ||\underline{x}_j - \bar{\underline{x}}_m||^2; m = 1, 2, ..., k, m \neq i.$$

   Update mean values according to new assignments.
   Continue doing so.

11.3 Derivatives

The algorithm has many derivatives. The most important ones are k-medians clustering and k nearest neighbors. k-medians clustering uses the median in each dimension instead of the mean, and this way minimizes the L1 norm.

A different algorithm though is the k nearest neighbors algorithm Here, clusters are already defined and new data points arrive. By a simple majority vote ($k = 2m - 1$) of the k neighbors, one can assign the new data point. Example: ask $k = 5$ neighbors, $m = 3$ belong to a particular cluster, making the decision.

Appendix A

Differentiation with Respect to Vectors

The rules to differentiate with respect to vectors as shown on Chapter 1 may appear ad hoc. Let us consider a real-valued column vector $\underline{v}$ with elements

$$\underline{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_M \end{bmatrix}.$$

Differentiating with respect to entry v_i results in:

$$\frac{\partial \underline{v}}{\partial v_i} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

with a single unit entry at position i. If we consider a linear combination of the vector $\underline{v}$ and differentiate, we obtain

$$\frac{\partial}{\partial v_i} \underline{a}^T \underline{v} = \frac{\partial}{\partial v_i} \sum_{k=1}^{M} a_k v_k = a_i.$$

If we build a new vector with all elements that result after differentiating with respect to component v_i from $i = 1$ to M:

$$\frac{\partial}{\partial \underline{v}} \underline{a}^T \underline{v} = \begin{bmatrix} \frac{\partial}{\partial v_1} \\ \frac{\partial}{\partial v_2} \\ \vdots \\ \frac{\partial}{\partial v_M} \end{bmatrix} \underline{a}^T \underline{v} = \underline{a}^T.$$

Note that the ordering in a row vector has been selected arbitrarily. As we continuously use row vectors as gradients this choice simplifies matters.

Let us now consider complex-valued signals. Let $g(z) = z = x + jy$ be a complex valued number. Its derivative is given by:

$$\begin{aligned}\frac{\partial g(z)}{\partial z} &\triangleq \frac{1}{2}\left[\frac{\partial g}{\partial x} - j\frac{\partial g}{\partial y}\right],\\ \frac{\partial g(z)}{\partial z^*} &\triangleq \frac{1}{2}\left[\frac{\partial g}{\partial x} + j\frac{\partial g}{\partial y}\right].\end{aligned}$$

After the mathematician Wirtinger [87], these operations are also-called the Wirtinger differential-operators.

Consider for example the term $g(z) = |z|^2 = zz^* = x^2 + y^2$. Differentiating with respect to x and y we obtain

$$\frac{\partial g(z)}{\partial x} = 2x$$

and

$$\frac{\partial g(z)}{\partial y} = 2y$$

and thus

$$\frac{\partial g(z)}{\partial z} = x - jy = z^*.$$

The operation thus does not effect the conjugate complex value at all. Analogously we can differentiate with respect to z^* and obtain in the above example z. The differentiating rules for complex valued vectors are obtained by combination of the rules shown here. For example, we find

$$\begin{aligned}\frac{\partial \|\underline{z}\|_2^2}{\partial \underline{z}} &= \frac{\partial\left(\underline{x}^T\underline{x} + \underline{y}^T\underline{y}\right)}{\partial \underline{z}} = \underline{x}^T - j\underline{y}^T = \underline{z}^H,\\ \frac{\partial \|\underline{z}\|_2^2}{\partial \underline{z}^H} &= \frac{\partial\left(\underline{x}^T\underline{x} + \underline{y}^T\underline{y}\right)}{\partial \underline{z}^H} = \underline{x} + j\underline{y} = \underline{z},\\ \frac{\partial \underline{z}^H A\underline{z}}{\partial \underline{z}} &= \frac{\partial\left(\underline{x}^T A\underline{x} - j\underline{y}^T A\underline{x} + j\underline{x}^T A\underline{y} + \underline{y}^T A\underline{y}\right)}{\partial \underline{z}} = \underline{x}^T A - j\underline{y}^T A = \underline{z}^H A.\end{aligned}$$

Appendix B

Convergence of Random Series

Definition **[Convergence]**: If there exists to each distance δ a number n_o such that $\text{dist}(x_n, y) < \delta$ for each $n > n_o$ at a given value y, then we call the series x_n convergent towards y:

$$\begin{aligned} x_n &\rightarrow y \\ y &= \lim_{n\to\infty} x_n. \end{aligned}$$

Here $\text{dist}(\cdot)$ is a metric. The point y towards the series x_n moves to, is called the limit. If there is more then one limit value, for example in oscillators, we call these points limit points.

Such definition of convergence is not very practically as it requires that the limit point y is a-priori known. Only then one can check whether convergence is given. Such form of convergence is also-called **convergence everywhere**. It is true for series as well as random series or random processes. For random processes $\mathbf{x}_n(\zeta)$ the limit may depend on the result of a random experiment ζ.

Definition **[Convergence with probability one]**: If a random process $\mathbf{x}_n(\zeta)$ converges to a limit $\mathbf{x}(\zeta)$, such that:

$$P\{\mathbf{x}_n \rightarrow \mathbf{x}\} = 1,$$

then we call it convergence with probability one or **convergent almost everywhere**.

Similar is the following term:
Definition **[Convergence in probability]**: If for each $\delta > 0$ the following is true

$$P(|\mathbf{x}_n - \mathbf{x}| > \delta) \rightarrow 0; \text{ for } n \rightarrow \infty$$

then we call this convergence in probability or **stochastic convergence**.

There are further definitions in this context:
Definition **[Convergence in the mean]**: If the following is true for a random process

$$\lim E[\mathbf{x}_n - \mathbf{x}] = 0,$$

then we call it convergence in the mean which is not to be confused with the next expression.

Definition **[Convergence in the mean square]**: If the following is true for a random process

$$\lim E[|\mathbf{x}_n - \mathbf{x}|^2] = 0,$$

then we call it convergence in the mean square sense or **limit in the mean**.

Definition **[Convergence in distribution]**: Let the distribution $F_{\mathbf{x}_n}(x)$ be of a random process $\mathbf{x}_n$. If this distribution moves to a fixed distribution $F_{\mathbf{x}}(x)$,

$$\lim F_{\mathbf{x}_n}(x) \to F_{\mathbf{x}}(x); \text{ for } n \to \infty$$

then we say that the random process converges in distribution.

There is obviously a multitude of convergence forms as it is often difficult to prove a particular convergence. On the other hand some convergence properties are related to others. For example the convergence in the mean is the weakest property. Is this satisfied we cannot conclude any other property. On the other extreme is convergence everywhere as it includes all other convergence properties. Figure B.1 displays the relations graphically. Here the notation is e: convergence everywhere, a.e.: convergence almost everywhere, p: convergence in probability, d: convergence in distribution, MS: convergence in the Mean Square Sense.

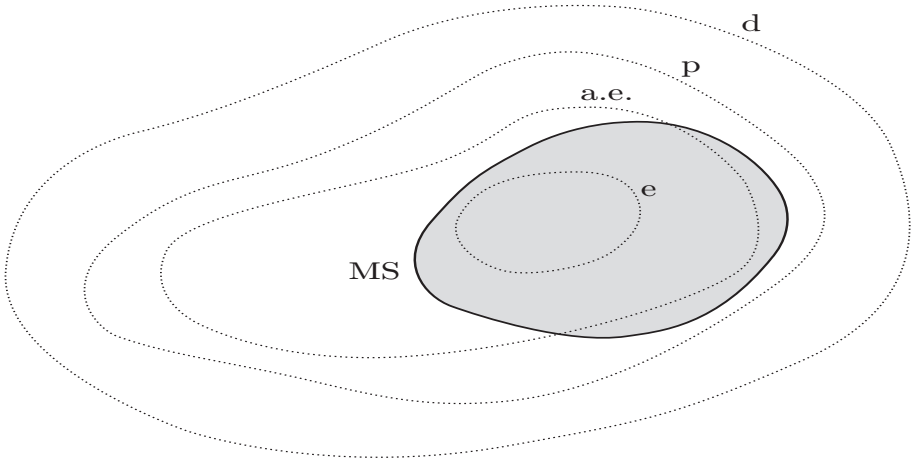

Figure B.1: *Convergence properties of random processes, e: convergence everywhere, a.e.: convergence almost everywhere, p: convergence in probability, d: convergence in distribution, MS: convergence in the Mean Square Sense.*

Appendix C

Spherically Invariant Random Processes

Definition [Two random variables]: Let us first consider two random variables $\mathbf{x}$ and $\mathbf{y}$. They are called spherically invariant if their joint density function $f_{\mathbf{xy}}(x, y)$ can be written as

$$f_{\mathbf{xy}}(x, y) = g(\sqrt{x^2 + y^2}) = g(r).$$

The following theorem holds.

Theorem C.1 *If two random variables* $\mathbf{x}$ *and* $\mathbf{y}$ *are spherically invariant and statistically independent then they are Gaussian distributed, zero-mean and of identical variance.*

Proof: Statistical independence means:

$$g(r) = g(\sqrt{x^2 + y^2}) = f_{\mathbf{x}}(x) f_{\mathbf{y}}(y).$$

Differentiating with respect to x results in:

$$\begin{aligned} \frac{\partial g(r)}{\partial x} &= \frac{\partial g(r)}{\partial r}\frac{\partial r}{\partial x} = \frac{\partial g(r)}{\partial r}\frac{r}{x} \\ \frac{x}{r} g^{(r)}(r) &= f_{\mathbf{x}}^{(x)}(x) f_{\mathbf{y}}(y). \end{aligned}$$

We divide the last equation by $xg(r)$ and obtain:

$$\frac{1}{r}\frac{g^{(r)}(r)}{r} = \frac{1}{x}\frac{f_{\mathbf{x}}^{(x)}(x)}{f_{\mathbf{x}}(x)} = \text{const.}$$

This must be constant as only then the function can only be dependent of x and only of r. The left hand side is a well-known differential equation whose solution is:

$$g(r) = A \exp\left(\alpha r^2/2\right).$$

We thus find that

$$f_{\mathbf{xy}}(x,y) = g(\sqrt{x^2+y^2}) = A\exp\left(\alpha(x^2+y^2)^2/2\right),$$

which allows for interpreting $\sigma_x^2 = \sigma_y^2 = -1/\alpha$.$\square$

The theorem is thus proven. Note, however, that this does not mean that all spherically invariant random variables are Gaussian distributed. It means very much that those that are not Gaussian distributed are not statistically independent. In those cases we do not have $f_{\mathbf{xy}}(x,y) = f_{\mathbf{x}}(x)f_{\mathbf{y}}(y)$.

Graphical Interpretation: A graphical interpretation of two random variables being spherically invariant is obtained when considering $g(r) = const$. For s specific value of the radius r this means that all pairs $(x,y)-$ with this radius need to be on a circle $r = \sqrt{x^2+y^2}$. Cutting the density function horizontally leads to circles on which the same probability for all pairs $(x,y)-$ exists.

We can now relax the condition of circles and extend the space towards **elliptically invariant random processes**:

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0. \tag{C.1}$$

We distinguish the three cases

1. $B^2 - 4AC > 0$:Hyperbola equation
2. $B^2 - 4AC = 0$:Parabola equation
3. $B^2 - 4AC < 0$:Ellipsoid equation

For two random variables, a more practical description is given by:

$$\begin{pmatrix} x-x_0 \\ y-y_0 \end{pmatrix}^T \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}^{-1} \begin{pmatrix} x-x_0 \\ y-y_0 \end{pmatrix} = const \tag{C.2}$$

$$\begin{pmatrix} x-x_0 \\ y-y_0 \end{pmatrix}^T \begin{pmatrix} a & b \\ b & c \end{pmatrix} \begin{pmatrix} x-x_0 \\ y-y_0 \end{pmatrix} = const. \tag{C.3}$$

A comparison with (C.1) delivers:

$$A = a, B = 2b, C = c, D = -2ax_0 - 2by_0, E = -2cy_0 - 2bx_0$$

and thus the condition for an ellipsoid: $B^2 - 4AC = 4b^2 - 4ac < 0$, thus $b^2 < ac$, which is given as the correlation coefficient ρ is bounded by $-1 \leq \rho \leq 1$ (proof by Cauchy-Schwarz-Inequality). The form in $C.3$ is much easier to handle as we can immediately recognize the

translational parameters (x_0, y_0). Also the variances and the correlation factor are given by:

$$\sigma_x^2 = \frac{c}{ac-b^2}; \sigma_y^2 = \frac{a}{ac-b^2}; \rho = -\frac{b}{\sqrt{ac}}.$$

The form in (C.2) comprises of a matrix inversion but on the other hand allows for relatively quick reading of desired parameters. With this form we can easily extend the description towards more dimensions than two, for example three:

$$\begin{pmatrix} x-x_0 \\ y-y_0 \\ z-z_0 \end{pmatrix}^T \begin{pmatrix} \sigma_x^2 & \rho_{xy}\sigma_x\sigma_y & \rho_{xz}\sigma_x\sigma_z \\ \rho_{xy}\sigma_x\sigma_y & \sigma_y^2 & \rho_{yz}\sigma_y\sigma_z \\ \rho_{xz}\sigma_x\sigma_z & \rho_{yz}\sigma_y\sigma_z & \sigma_z^2 \end{pmatrix}^{-1} \begin{pmatrix} x-x_0 \\ y-y_0 \\ z-z_0 \end{pmatrix} = const. \quad \text{(C.4)}$$

More generally we can describe an elliptical equation by

$$\underline{\mathbf{x}}^T R_{\underline{\mathbf{xx}}}^{-1} \underline{\mathbf{x}} = const \quad \text{(C.5)}$$

with $R_{\underline{\mathbf{xx}}}$ being the autocorrelation matrix of the random vector $\underline{\mathbf{x}}$.

Theorem C.2 *Elliptical random processes maintain such property under linear transformation. There exists at least one linear transformation that transforms elliptical random variables into spherical ones.*

Proof: Simply by substitution. Assume we have an elliptical random process $\underline{\mathbf{x}}$ with autocorrelation matrix $R_{\underline{\mathbf{xx}}}$. By a linear invertible transformation A we generate $\underline{\mathbf{y}} = A\underline{\mathbf{x}}$ or $\underline{\mathbf{x}} = A^{-1}\underline{\mathbf{y}}$. For the autocorrelation matrix of $\underline{\mathbf{y}}$ we find:

$$R_{\underline{\mathbf{yy}}} = E[\underline{\mathbf{y}}\underline{\mathbf{y}}^T] = E[A\underline{\mathbf{x}}\underline{\mathbf{x}}^T A^T] = AR_{\underline{\mathbf{xx}}}A^T$$

and

$$R_{\underline{\mathbf{xx}}} = A^{-1} R_{\underline{\mathbf{yy}}} A^{-T}.$$

Thus $\underline{\mathbf{y}}^T R_{\underline{\mathbf{yy}}}^{-1} \underline{\mathbf{y}} = \underline{\mathbf{x}}^T A^T R_{\underline{\mathbf{yy}}}^{-1} A\underline{\mathbf{x}} = \underline{\mathbf{x}}^T R_{\underline{\mathbf{xx}}}^{-1} \underline{\mathbf{x}}$. For density functions under linear transformation we have

$$f_{\underline{\mathbf{y}}}(\underline{y}) = \frac{f_{\underline{\mathbf{x}}}(\underline{x})}{|\det A|} = \frac{g(\underline{\mathbf{x}}^T R_{\underline{\mathbf{xx}}}^{-1} \underline{\mathbf{x}})}{|\det A|} = \frac{g(\underline{\mathbf{y}}^T R_{\underline{\mathbf{yy}}}^{-1} \underline{\mathbf{y}})}{|\det A|}.$$

Thus, the first part of the theorem is shown. Let us now select a particular $A = R_{\underline{\mathbf{xx}}}^{-1/2}$, then we have

$$R_{\underline{\mathbf{yy}}} = E[\underline{\mathbf{y}}\underline{\mathbf{y}}^T] = AR_{\underline{\mathbf{xx}}}A^T = I.$$

The in this way linearly transformed random vector $\underline{\mathbf{y}}$ is thus uncorrelated. The particular choice of A is always possible as the autocorrelation matrix is always Hermitian and

positive. Note that with this choice of A we do not only guarantee decorrelation but also identical variance of all entries in $\underline{\mathbf{y}}$. If there would be required only decorrelation we can satisfy this for all diagonal matrices D with $A = DR_{\underline{\mathbf{xx}}}^{-1/2}$.□

Polar coordinates: No matter if we have an elliptical distribution with or without mean the basic distribution is spherically invariant and by adding mean and introducing linear transformations corresponding autocorrelation matrices can be introduced. It thus makes sense to investigate this process more and to introduce polar coordinates. Let us first consider two spherically invariant random variables $\mathbf{x}$ and $\mathbf{y}$ in polar form:

$$f_{\mathbf{xy}}(x,y) = g(\sqrt{x^2+y^2}) = g(r) = \frac{f_{\mathbf{r}}(r)}{2\pi} = \frac{1}{\pi} f(r^2, 2). \tag{C.6}$$

Such formulation is independent of the radial density- Note that the angle component ϕ is always constant between $[-\pi, \pi]$. Furthermore we immediately conclude statistical independence of the two random variables r and ϕ in polar form

$$f_{\mathbf{r}\phi}(r,\phi) = f_{\mathbf{r}}(r) f_{\phi}(\phi),$$

while the Cartesian counterparts $\mathbf{x}$ and $\mathbf{y}$ are not necessarily statistical independent. We have already shown that this is only the case for Gaussian distributions. Such formulations in polar coordinates can be extended to more dimensions. If done so we recognize that all angle densities are statistically independent to each other and to the radial density. For all higher angles we always find a constant distribution in $[0, \pi]$.

In (C.6) we already introduced a particular description for the radial density in this case for the two dimensional density function $\frac{1}{\pi} f(r^2, 2)$ or more general $\frac{1}{\pi^{M/2}} f(r^2, M)$ for M dimensions. To emphasize the dimension M we took it on as a parameter of the density function. The term $\frac{1}{\pi^{M/2}}$ turns out to be quite practical for compact writing. As marginal densities can be computed by integration out of joint densities we find:

$$\frac{1}{\pi^{\frac{M}{2}}} f(r^2, M) = \int_{-\infty}^{\infty} \frac{1}{\pi^{\frac{M+1}{2}}} f(r^2 + s_{M+1}^2, M+1) ds_{M+1}.$$

Here the joint density $\frac{1}{\pi^{\frac{M}{2}}} f(r^2, M)$ consists of M Cartesian components $s_1, s_2, ...s_M$ with condition $r^2 = s_1^2 + s_2^2 + ... + s_M^2$.

For spherically invariant process even the converse is true or in other words, if the radial density is known in one dimension, it can be computed for all dimensions. This can be easily

shown when moving from dimension $M+2$ to M:

$$\begin{aligned}\frac{1}{\pi^{\frac{M}{2}}}f(r^2,M) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{1}{\pi^{\frac{M+2}{2}}}f(r^2+s_{M+1}^2+s_{M+2}^2,M+2)ds_{M+1}ds_{M+2}\\ &= \int_0^{\infty}\int_0^{2\pi}\frac{1}{\pi\pi^{\frac{M}{2}}}\rho f(r^2+\rho^2,M+2)d\rho d\phi\\ &= \int_0^{\infty}\frac{1}{\pi^{\frac{M}{2}}}2\rho f(r^2+\rho^2,M+2)d\rho.\end{aligned}$$

With an additional substitution $r^2=s, s+\rho^2=x, 2rdr=ds$ we now obtain

$$\frac{1}{\pi^{\frac{M}{2}}}f(s,M)=\frac{1}{\pi^{\frac{M}{2}}}\int_s^{\infty}f(x,M+2)dx \tag{C.7}$$

or in other words:

$$f(s,M+2)=-\frac{\partial}{\partial s}f(s,M). \tag{C.8}$$

There is of course also a description from $f(s,M+1)$ to $f(s,M)$, however somewhat more involved. Details can be found in [8].

Example C.1 *Consider a Gaussian distribution:*

$$\frac{1}{\pi}f(r^2,2)=\frac{1}{2\pi}e^{-r^2/2}.$$

Thus all even numbered higher dimensional density functions are given by

$$f(s,2M)=\frac{1}{2^M}e^{-s/2}.$$

We also know the one-dimensional density function

$$\frac{1}{\pi^{1/2}}f(r^2,1)=\sqrt{\frac{2}{\pi}}e^{-r^2/2}.$$

Thus all odd numbered higher dimensional density functions are given by

$$f(s,2M+1)=\frac{1}{2^M}2^{-1/2}e^{-s/2}.$$

Random process: So far we described a finite amount of M components in a random vector $\underline{\mathbf{x}}$ by its joint density $\pi^{-M/2}f(r^2,M)$. If we move M towards infinity the vector describes a random process. Finite values of M can thus be interpreted as part of a random process, explaining thus that there is always a vector with higher dimension whose components may or may not be statistically dependent. We can thus interpret a

vector as the presence of a random process of which we always see a fraction of length M. Independent of the time we always obtain the same joint density function for M components. We thus conclude that we have stationary processes.

Radial density function: Transforming polar coordinates into Cartesian is given by

$$\begin{aligned} s_1 &= r\prod_{j=1}^{M-1}\sin(\phi_j) \\ s_k &= r\cos(\phi_{M+1-k})\prod_{j=1}^{M-k}\sin(\phi_j); k=2,...,M-1 \\ s_M &= r\cos(\phi_1). \end{aligned}$$

Note there that the joint density $f(r^2, M)$ is not the radial density $f_{\mathbf{r}}(r)$.

$$\frac{1}{\pi^{M/2}}f(r^2,M) = f_{\mathbf{r}}(r)f_{\phi_1}(\phi_1)...f_{\phi_{M-1}}(\phi_{M-1})\Delta(\phi_1,..,\phi_{M-1})$$

with the Jacobi determinant for polar coordinates

$$\Delta(\phi_1,..,\phi_{M-1}) = r^{M-1}\prod_{j=1}^{M-2}\left(\sin(\phi_j)\right)^{M-1-j}.$$

In order to obtain the radial density we have to integrate over all angles, thus to compute the marginal density in r

$$\int ...\int \frac{1}{\pi^{M/2}}f(r^2,M)d\phi_1..d\phi_{M-1} = f_{\mathbf{r}}(r)\int ...\int f_{\phi_1}(\phi_1)...f_{\phi_{M-1}}(\phi_{M-1})\Delta(\phi_1,..,\phi_{M-1})d\phi_1..d\phi_{M-1}.$$

Since all angular density functions are known this term can be precomputed for a given dimension M

$$f_{\mathbf{r}_M}(r) = \frac{2}{\Gamma(M/2)}r^{M-1}f(r^2,M). \tag{C.9}$$

Note that the radial component in $\frac{1}{\pi^{\frac{M}{2}}}f(r^2,M)$ has M components while it has $M+1$ in $\frac{1}{\pi^{\frac{M+1}{2}}}f(r^2,M+1)$. The radial density $f_{r_M}(r)$ behaves similar. To emphasize this we now introduce a dimension M in form of an index $\mathbf{r}_M$. Special cases are known for $M=1,2,3$:

$$\begin{aligned} \text{Positive Gaussian} &: f_{\mathbf{r}_1}(r) = \frac{2}{\Gamma(1/2)}f(r^2,1) = \sqrt{\frac{2}{\pi}}e^{-r^2/2} \\ \text{Rayleigh} &: f_{\mathbf{r}_2}(r) = \frac{2}{\Gamma(1)}rf(r^2,2) = re^{-r^2/2} \\ \text{Maxwell} &: f_{\mathbf{r}_3}(r) = \frac{2}{\Gamma(3/2)}r^2f(r^2,3) = \sqrt{\frac{2}{\pi}}r^2e^{-r^2/2} \end{aligned}$$

Note that $r \geq 0$ which is a bit unusual for the Gaussian distribution as they have usually negative arguments as well. In that case the factor is $1/\sqrt{2\pi}$ instead.

Ergodicity: Consider the temporal averaging over the components $\mathbf{s}_i$ of a spherically invariant random process:

$$\sigma_M^2 = \frac{1}{M}\sum_{i=1}^{M} \mathbf{s}_i^2 = \frac{1}{M}\mathbf{r}_M^2.$$

Thus its variance is $\sigma_M = M^{-1/2}\mathbf{r}_M$ and follows the radial density. Let us now grow M to infinity. What happens?

$$\lim_{M\to\infty} \sigma_M = \lim_{M\to\infty} M^{-1/2}\mathbf{r}_M$$

and thus we obtain for the density of σ_M

$$f_\sigma(r) = \lim_{M\to\infty} f_{\sigma_M}(r) = \lim_{M\to\infty} M^{1/2} f_{\mathbf{r}_M}(M^{1/2}r).$$

We find thus that in general in the limit the variance is also a random variable described by the density function $f_\sigma(r)$. We cannot expect that the ensemble average and the temporal average are identical as the temporal average would need to be a constant. We can thus conclude that in general spherically invariant processes are not ergodic although they are stationary.

A particularity is again the Gaussian process. For the Gaussian process we have (just considering even orders)

$$f_{\mathbf{r}_{2M}}(r) = \frac{2}{\Gamma(M)} r^{2M-1} f(r^2, 2M) = \frac{2}{\Gamma(M)}\frac{1}{2^M} r^{2M-1} e^{-r^2/2}. \tag{C.10}$$

The density function follows a χ^2- distribution. The limit of $M^{1/2} f_{\mathbf{r}_M}(M^{1/2}r)$ tends to 1, thus

$$\lim_{M\to\infty} f_{\mathbf{r}_M}(r) = \delta(r-1).$$

The Gaussian process is thus ergodic. The property of the Gaussian process that the radial component tends to be a constant with increasing dimension is often called the hardening phenomenon.

Example C.2 *This opens the question how we can generate other spherically invariant processes next to the Gaussian process, possibly with particular radial density. Consider first a multiplication of a Gaussian process* $\mathbf{x}_k$ *by a random variable* $\mathbf{y}$ *to obtain* $\mathbf{z}_k = \mathbf{x}_k\mathbf{y}$. *Let us first consider one element for time instant* k, *thus* $\mathbf{z} = \mathbf{x}\mathbf{y}$. *The density function of* $\mathbf{z}$ *can be obtained by the following two methods:*

Path 1: Consider a fixed value of y. Then we have

$$f_{\mathbf{z}}(z|y) = \frac{1}{\sqrt{2\pi y^2}} e^{-(z/y)^2/2}.$$

The joint density $f_{\mathbf{zy}}(z,y) = f_{\mathbf{z}}(z|y) f_{\mathbf{y}}(y)$. We obtain the desired marginal density by integration

$$f_{\mathbf{z}}(z) = \int f_{\mathbf{z}}(z|y) f_{\mathbf{y}}(y) dy = \int \frac{1}{\sqrt{2\pi y^2}} e^{-(z/y)^2/2} f_{\mathbf{y}}(y) dy.$$

Path 2: Construct the auxiliary variable $\mathbf{w} = \mathbf{y}$ and compute the transformation of $\mathbf{x}, \mathbf{y}$ to $\mathbf{z}, \mathbf{w}$:

$$f_{\mathbf{zw}}(z,w) = \frac{f_{\mathbf{xy}}(z/w, w)}{|w|} = \frac{f_{\mathbf{x}}(z/w) f_{\mathbf{y}}(w)}{|w|}.$$

We obtain the marginal density of $\mathbf{z}$ again by integration

$$f_{\mathbf{z}}(z) = \int \frac{f_{\mathbf{x}}(z/w) f_{\mathbf{y}}(w)}{|w|} dw = \int \frac{1}{\sqrt{2\pi}} \frac{e^{-(z/w)^2/2} f_{\mathbf{y}}(w)}{|w|} dw$$

In both cases we thus obtain the same result. We recall that $r \geq 0$ and identify

$$f(r^2, 1) = 2^{1/2} \int \frac{e^{-(r/w)^2/2} f_{\mathbf{y}}(w)}{|w|} dw. \tag{C.11}$$

Example C.3 *We now want to extend this example. Consider three IID Gaussian distributed random variables $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ being multiplied by a fourth random variable $\mathbf{y}$ of density $f_{\mathbf{y}}(y)$ that is statistically independent of the first three. For the joint densities we have:*

$$f_{\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3,\mathbf{y}}(x_1, x_2, x_3, y) = f_{\mathbf{x}_1}(x_1) f_{\mathbf{x}_2}(x_2) f_{\mathbf{x}_3}(x_3) f_{\mathbf{y}}(y)$$

We thus construct three new random values $\mathbf{z}_1 = \mathbf{x}_1\mathbf{y}, \mathbf{z}_2 = \mathbf{x}_2\mathbf{y}, \mathbf{z}_3 = \mathbf{x}_3\mathbf{y}$. The joint density of the three new values can be computed by the auxiliary variable $\mathbf{w} = \mathbf{y}$:

$$f_{\mathbf{z}_1,\mathbf{z}_2,\mathbf{z}_3}(z_1, z_2, z_3) = \int f_{\mathbf{z}_1,\mathbf{z}_2,\mathbf{z}_3,\mathbf{w}}(z_1, z_2, z_3, w) dw.$$

We obtain

$$f_{\mathbf{z}_1,\mathbf{z}_2,\mathbf{z}_3,\mathbf{w}}(z_1, z_2, z_3, w) = \frac{f_{\mathbf{x}_1}(z_1/w) f_{\mathbf{x}_2}(z_2/w) f_{\mathbf{x}_3}(z_3/w) f_{\mathbf{y}}(w)}{|w|^3}.$$

As all three variables $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ are Gaussian we find:

$$f_{\mathbf{z}_1,\mathbf{z}_2,\mathbf{z}_3,\mathbf{w}}(z_1, z_2, z_3, w) = (2\pi)^{-3/2} \frac{f_{\mathbf{y}}(w) e^{-\frac{z_1^2+z_2^2+z_3^2}{2w^2}}}{|w|^3}.$$

The desired joint density function is thus:

$$f_{\mathbf{z}_1,\mathbf{z}_2,\mathbf{z}_3}(z_1,z_2,z_3) = \int (2\pi)^{-3/2} \frac{f_{\mathbf{y}}(w) e^{-\frac{z_1^2+z_2^2+z_3^2}{2w^2}}}{|w|^3} dw.$$

We again identify for $r \geq 0$:

$$f(r^2,3) = 2^{-1/2} \int \frac{f_{\mathbf{y}}(w) e^{-\frac{r^2}{2w^2}}}{|w|^3} dw. \tag{C.12}$$

For spherically invariant random processes (C.8) must hold, thus:

$$\begin{aligned} \frac{d}{ds} f(s,1) &= \frac{d}{ds} 2^{1/2} \int \frac{e^{-s/w^2/2} f_{\mathbf{y}}(w)}{|w|} dw \\ &= -2^{-1/2} \int \frac{e^{-s/w^2/2} f_{\mathbf{y}}(w)}{|w|^3} dw \\ &= -f(s,3). \end{aligned}$$

In this way we can compute the joint densities of more than three variables. We must have

$$f(s,2n+1) = (-1)^n \frac{d^{(n)}}{ds^n} f(s,1).$$

In our case thus

$$f(s,2n+1) = (-1)^n 2^{-(2n-1)/2} \int \frac{e^{-s/w^2/2} f_{\mathbf{y}}(w)}{|w|^{2n+1}} dw.$$

With this trick we can now compute the radial densities for odd dimensions $M = 2n+1$ (see (C.9))

$$f_{\mathbf{r}_M}(r) = \frac{1}{\Gamma(M/2)} (-1)^{(M-1)/2} 2^{-M/2} \int r^{M-1} \frac{e^{-r^2/w^2/2} f_{\mathbf{y}}(w)}{|w|^M} dw.$$

Remark: The examples here showed a so-called **product process**. It can be shown that all spherically invariant processes are a product process, in particular a product of a Gaussian process with a random variable. (for more details see [8]).

Appendix D

Remarks on Gaussian Processes

Gaussian random variables play an important role in many applications, in particular if the sum of several random variables shows up. Most often the central value theorem is then true which says that loosely speaking a sum of many arbitrary random variables results asymptotically in a Gaussian random variable. For a real-valued vector of p dimensions $\underline{\mathbf{x}}$ with Gaussian entries mean $\underline{\bar{x}}$ and covariance matrix $C_{\underline{\mathbf{xx}}} = \mathrm{E}[(\underline{\mathbf{x}} - \underline{\bar{x}})(\underline{\mathbf{x}} - \underline{\bar{x}})^T]$ we obtain the following probability density function:

$$f_{\underline{\mathbf{x}}}(\underline{x}) = \frac{1}{\sqrt{(2\pi)^p}} \frac{1}{\sqrt{\det C_{\underline{\mathbf{xx}}}}} \; e^{-\frac{1}{2}[\underline{x} - \underline{\bar{x}}]^T C_{\underline{\mathbf{xx}}}^{-1} [\underline{x} - \underline{\bar{x}}]}. \tag{D.1}$$

For zero-mean random variables the covariance matrix and the autocorrelation matrix $R_{\underline{\mathbf{xx}}} = C_{\underline{\mathbf{xx}}}$ are identical. Correspondingly two joint variables $\underline{\mathbf{x}} \in I\!R^{1\times p}$ and $\underline{\mathbf{y}} \in I\!R^{1\times q}$ can be described by their joint pdf

$$f_{\underline{\mathbf{x}},\underline{\mathbf{y}}}(\underline{x}, \underline{y}) = \frac{1}{\sqrt{(2\pi)^p}} \frac{1}{\sqrt{(2\pi)^q}} \frac{1}{\sqrt{\det C_{\underline{\mathbf{xx}},\underline{\mathbf{yy}}}}} \; e^{-\frac{1}{2}[(\underline{x} - \underline{\bar{x}})^T, (\underline{y} - \underline{\bar{y}})^T] C_{\underline{\mathbf{xx}},\underline{\mathbf{yy}}}^{-1} \begin{bmatrix} \underline{x} - \underline{\bar{x}} \\ \underline{y} - \underline{\bar{y}} \end{bmatrix}}, \tag{D.2}$$

and covariance matrix

$$C_{\underline{\mathbf{xx}},\underline{\mathbf{yy}}} = \begin{bmatrix} C_{\underline{\mathbf{xx}}} & C_{\underline{\mathbf{xy}}} \\ C_{\underline{\mathbf{yx}}} & C_{\underline{\mathbf{yy}}} \end{bmatrix}. \tag{D.3}$$

Obviously the real-valued Gaussian process is defined by its mean and its covariance matrix (first and second order moments) In the following we show that this is also true for complex-valued Gaussian random processes.

We consider a complex-valued random vector $\underline{\mathbf{z}}$, comprising of two real-valued random vectors $\mathbf{z}_i = \mathbf{x}_i + j\mathbf{y}_i$. It is called Gaussian if its real and imaginary parts are called joint

Gaussian. Assume the first two moments of the complex-valued Gaussian process are given, thus $\bar{\underline{z}}$ and $C_{\underline{\mathbf{zz}}}$:

$$C_{\underline{\mathbf{zz}}} = \mathrm{E}[(\underline{\mathbf{z}} - \bar{\underline{z}})(\underline{\mathbf{z}} - \bar{\underline{z}})^*] = C_{\underline{\mathbf{xx}}} + C_{\underline{\mathbf{yy}}} + j(C_{\underline{\mathbf{yx}}} - C_{\underline{\mathbf{xy}}}). \tag{D.4}$$

This information is not sufficient to find uniquely $C_{\underline{\mathbf{xx}}}, C_{\underline{\mathbf{yy}}}, C_{\underline{\mathbf{xy}}}$ as the real part of $C_{\underline{\mathbf{zz}}}$ only defines the sum of $C_{\underline{\mathbf{xx}}} + C_{\underline{\mathbf{yy}}}$ and the imaginary part only the difference of $C_{\underline{\mathbf{yx}}} - C_{\underline{\mathbf{xy}}}$. In order to separate the three parts we need further knowledge, for example

$$C_{\underline{\mathbf{zz}}^*} = \mathrm{E}[(\underline{\mathbf{z}} - \bar{\underline{z}})(\underline{\mathbf{z}} - \bar{\underline{z}})] = C_{\underline{\mathbf{xx}}} - C_{\underline{\mathbf{yy}}} + j\left(C_{\underline{\mathbf{yx}}} + C_{\underline{\mathbf{xy}}}\right). \tag{D.5}$$

With this additional knowledge the three parameter could be defined uniquely.Gaussian processes that are circular (spherically invariant) have the following property[1]:

$$C_{\underline{\mathbf{zz}}^*} = C_{\underline{\mathbf{xx}}} - C_{\underline{\mathbf{yy}}} + j\left(C_{\underline{\mathbf{yx}}} + C_{\underline{\mathbf{xy}}}\right) = 0. \tag{D.6}$$

This is equivalent to

$$C_{\underline{\mathbf{xx}}} = C_{\underline{\mathbf{yy}}} \text{ and } C_{\underline{\mathbf{xy}}} = -C_{\underline{\mathbf{yx}}}. \tag{D.7}$$

Note that the requirement $C_{\underline{\mathbf{xy}}} = -C_{\underline{\mathbf{yx}}}$ is unusual and indeed this requirement assures the circularity of the process. Consider for example a two dimensional random vector. The correlation matrix between $\mathbf{x}$ and $\mathbf{y}$ is given by

$$C_{\mathbf{xy}} = \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix}. \tag{D.8}$$

In order to have $C_{\mathbf{xy}} = -C_{\mathbf{yx}}$, we must have: $\alpha_{11} = \alpha_{22} = 0$ und $\alpha_{12} = -\alpha_{21}$. Thus we obtain the particular form

$$C_{\mathbf{xy}} = \begin{bmatrix} 0 & \alpha_{12} \\ -\alpha_{12} & 0 \end{bmatrix}. \tag{D.9}$$

Is this a contradiction to the statement that quadratic forms need to be positive, thus their imaginary part being negative? The answer is no as we will show on the next example. We consider the quadratic form of a circular complex-valued random vectors $\underline{z}^H C_{\underline{\mathbf{zz}}} \underline{z}$:

$$\frac{1}{2}\underline{z}^H C_{\underline{\mathbf{zz}}} \underline{z} = (\underline{x} - j\underline{y})^T \left(C_{\underline{\mathbf{xx}}} + C_{\underline{\mathbf{yy}}}\right)(\underline{x} + j\underline{y}). \tag{D.10}$$

We obtain for its real part:

$$\frac{1}{2}\Re\underline{z}^H C_{\underline{\mathbf{zz}}} \underline{z} \;\; = \;\; \underline{x}^T[C_{\underline{\mathbf{xx}}} + C_{\underline{\mathbf{yy}}}]\underline{x} + \underline{y}^T[C_{\underline{\mathbf{xx}}} + C_{\underline{\mathbf{yy}}}]\underline{y} \geq 0. \tag{D.11}$$

[1]Note that spherically invariance requires that the joint density function is a function of the radius r, for a Gaussian process it is proportional to $\exp(-r^2)$.

We obtain for the imaginary part:

$$\frac{1}{2}\Im \underline{z}^H C_{\underline{\mathbf{zz}}} \underline{z} \;=\; \underline{x}^T [C_{\underline{\mathbf{xy}}} + C_{\underline{\mathbf{yy}}}] \underline{y} - \underline{y}^T [C_{\underline{\mathbf{xx}}} + C_{\underline{\mathbf{yy}}}.] \underline{x} = 0. \tag{D.12}$$

Equivalently the covariance matrix in (D.3) for a circular complex-valued vector $\underline{\mathbf{z}}$ can be written as

$$C_{\underline{\mathbf{xx}},\underline{\mathbf{yy}}} = \begin{bmatrix} C_{\underline{\mathbf{xx}}} & C_{\underline{\mathbf{xy}}} \\ -C_{\underline{\mathbf{xy}}} & C_{\underline{\mathbf{xx}}} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} \text{Real}\,\{C_{\underline{\mathbf{zz}}}\} & -\text{Imag}\,\{C_{\underline{\mathbf{zz}}}\} \\ \text{Imag}\,\{C_{\underline{\mathbf{zz}}}\} & \text{Real}\,\{C_{\underline{\mathbf{zz}}}\} \end{bmatrix}. \tag{D.13}$$

With this we can write the p-dimensional joint density of a complex-valued random vector $\underline{\mathbf{z}}$ can be compactly described by its covariance matrix $C_{\underline{\mathbf{zz}}}$ (compare to (D.2) with $p = q$):

$$f_{\underline{\mathbf{x}},\underline{\mathbf{y}}}(\underline{x}, \underline{y}) = f_{\underline{\mathbf{z}}}(\underline{z}) = \frac{1}{\pi^p} \frac{1}{|\det C_{\underline{\mathbf{zz}}}|} \; e^{-[\underline{z} - \bar{\underline{z}}]^H C_{\underline{\mathbf{zz}}}^{-1} [\underline{z} - \bar{\underline{z}}]}. \tag{D.14}$$

To prove this we have to show the following two identities

$$\frac{1}{2}[(\underline{x} - \bar{\underline{x}})^T, (\underline{y} - \bar{\underline{y}})^T] C_{\underline{\mathbf{xx}},\underline{\mathbf{yy}}}^{-1} \begin{bmatrix} \underline{x} - \bar{\underline{x}} \\ \underline{y} - \bar{\underline{y}} \end{bmatrix} \;=\; [\underline{z} - \bar{\underline{z}}]^H C_{\underline{\mathbf{zz}}}^{-1} [\underline{z} - \bar{\underline{z}}] \tag{D.15}$$

$$\frac{1}{|\det C_{\underline{\mathbf{zz}}}|} \;=\; \frac{1}{\sqrt{\left|\det C_{\underline{\mathbf{xx}},\underline{\mathbf{yy}}}\right|}} \frac{1}{2^p}. \tag{D.16}$$

The first identity is shown by computing the inverse of the matrix and comparing the individual terms of the $\underline{\mathbf{x}}$ and $\underline{\mathbf{y}}$ pairs.

$$C_{\underline{\mathbf{xx}},\underline{\mathbf{yy}}}^{-1} \;=\; \begin{bmatrix} C_{\underline{\mathbf{xx}}} & C_{\underline{\mathbf{xy}}} \\ -C_{\underline{\mathbf{xy}}} & C_{\underline{\mathbf{xx}}} \end{bmatrix}^{-1} = \begin{bmatrix} \Delta^{-1} & C_{\underline{\mathbf{xx}}}^{-1} C_{\underline{\mathbf{xy}}} \Delta^{-1} \\ -C_{\underline{\mathbf{xx}}}^{-1} C_{\underline{\mathbf{xy}}} \Delta^{-1} & \Delta^{-1} \end{bmatrix} \tag{D.17}$$

$$\Delta \;=\; C_{\underline{\mathbf{xx}}} + C_{\underline{\mathbf{xy}}} C_{\underline{\mathbf{xx}}}^{-1} C_{\underline{\mathbf{xy}}} \tag{D.18}$$

$$\frac{1}{2} C_{\underline{\mathbf{zz}}}^{-1} \;=\; [C_{\underline{\mathbf{xx}}} + j C_{\underline{\mathbf{xy}}}]^{-1} = [I - j C_{\underline{\mathbf{xx}}}^{-1} C_{\underline{\mathbf{xy}}}] \Delta^{-1}. \tag{D.19}$$

Writing both terms in real and imaginary part $\underline{x}$ and $\underline{y}$, we obtain for the second part (D.19)

$$\begin{aligned} \frac{1}{2} \underline{z}^H C_{\underline{\mathbf{zz}}}^{-1} \underline{z} \;&=\; \underline{x}^T \Delta^{-1} \underline{x} + \underline{y}^T \Delta^{-1} \underline{y} \\ &+\; \underline{x}^T C_{\underline{\mathbf{xx}}}^{-1} C_{\underline{\mathbf{xy}}} \Delta^{-1} \underline{y} - \underline{y}^T C_{\underline{\mathbf{xx}}}^{-1} C_{\underline{\mathbf{xy}}} \Delta^{-1} \underline{x} \\ &-\; j \left[\underline{x}^T C_{\underline{\mathbf{xx}}}^{-1} C_{\underline{\mathbf{xy}}} \Delta^{-1} \underline{x} + \underline{y}^T C_{\underline{\mathbf{xx}}}^{-1} C_{\underline{\mathbf{xy}}} \Delta^{-1} \underline{y} \right] \\ &-\; j \left[\underline{y}^T \Delta^{-1} \underline{x} - \underline{x}^T \Delta^{-1} \underline{y} \right], \end{aligned} \tag{D.20}$$

and realize that the real part is indeed identical. Note that the imaginary part becomes zero again. However, this is not obvious and will be considered more closely. The last term in the imaginary part

$$\left[\underline{y}^T\Delta^{-1}\underline{x} - \underline{x}^T\Delta^{-1}\underline{y}\right] = 0.$$

The first term is of more difficult nature:

$$\left[\underline{x}^T C_{\underline{\mathbf{xx}}}^{-1} C_{\underline{\mathbf{xy}}}\Delta^{-1}\underline{x} + \underline{y}^T C_{\underline{\mathbf{xx}}}^{-1} C_{\underline{\mathbf{xy}}}\Delta^{-1}\underline{y}\right].$$

Here we have that $(C_{\underline{\mathbf{xx}}}^{-1}C_{\underline{\mathbf{xy}}}\Delta^{-1})^T = -\Delta^{-1}C_{\underline{\mathbf{xy}}}C_{\underline{\mathbf{xx}}}^{-1}$, and due to symmetry $C_{\underline{\mathbf{xy}}} = -C_{\underline{\mathbf{xy}}}^T$ also: $\Delta^{-1}C_{\underline{\mathbf{xy}}}C_{\underline{\mathbf{xx}}}^{-1} = C_{\underline{\mathbf{xx}}}^{-1}C_{\underline{\mathbf{xy}}}\Delta^{-1}$. And further we find

$$\underline{x}^T C_{\underline{\mathbf{xx}}}^{-1} C_{\underline{\mathbf{xy}}}\Delta^{-1}\underline{x} = 0.$$

The later relation we prove by

$$C_{\underline{\mathbf{xx}}}^{-1}C_{\underline{\mathbf{xy}}}\Delta^{-1} = \Delta^{-1}C_{\underline{\mathbf{xy}}}C_{\underline{\mathbf{xx}}}^{-1}.$$

Multiplying Δ from left and right we obtain:

$$C_{\underline{\mathbf{xy}}}[I + (C_{\underline{\mathbf{xx}}}^{-1}C_{\underline{\mathbf{xy}}})^2] = [I + (C_{\underline{\mathbf{xy}}}C_{\underline{\mathbf{xx}}}^{-1})^2]C_{\underline{\mathbf{xy}}}.$$

Multiplying with $C_{\underline{\mathbf{xy}}}$ finally shows the identity.

To compute the determinant we use the Schur complement. We have:

$$\det\left(\begin{bmatrix} C_{\underline{\mathbf{xx}}} & C_{\underline{\mathbf{xy}}} \\ -C_{\underline{\mathbf{xy}}} & C_{\underline{\mathbf{xx}}} \end{bmatrix}\right) = \det(C_{\underline{\mathbf{xx}}})\det(C_{\underline{\mathbf{xx}}} + C_{\underline{\mathbf{xy}}}C_{\underline{\mathbf{xx}}}^{-1}C_{\underline{\mathbf{xy}}}). \tag{D.21}$$

Note that we also have:

$$\begin{aligned}\det\left(\begin{bmatrix} C_{\underline{\mathbf{xx}}} - jC_{\underline{\mathbf{xy}}} & C_{\underline{\mathbf{xy}}} \\ -C_{\underline{\mathbf{xy}}} - jC_{\underline{\mathbf{xx}}} & C_{\underline{\mathbf{xx}}} \end{bmatrix}\right) &= \det(C_{\underline{\mathbf{xx}}})\det(C_{\underline{\mathbf{xx}}} - jC_{\underline{\mathbf{xy}}} + C_{\underline{\mathbf{xy}}}C_{\underline{\mathbf{xx}}}^{-1}(C_{\underline{\mathbf{xy}}} + jC_{\underline{\mathbf{xx}}})) \\ &= \det(C_{\underline{\mathbf{xx}}})\det(C_{\underline{\mathbf{xx}}} + C_{\underline{\mathbf{xy}}}C_{\underline{\mathbf{xx}}}^{-1}C_{\underline{\mathbf{xy}}}). \end{aligned} \tag{D.22}$$

Thus, the determinant is not negative due to the quadratic term. We have:

$$\det\left(\begin{bmatrix} 2C_{\underline{\mathbf{xx}}} & 2C_{\underline{\mathbf{xy}}} \\ -2C_{\underline{\mathbf{xy}}} & 2C_{\underline{\mathbf{xx}}} \end{bmatrix}\right) = \det\left(\begin{bmatrix} C_{\underline{\mathbf{zz}}}^* & 2C_{\underline{\mathbf{xy}}} \\ -jC_{\underline{\mathbf{zz}}}^* & 2C_{\underline{\mathbf{xx}}} \end{bmatrix}\right). \tag{D.23}$$

Applying the Schur complement a second time and we find the solution:

$$\det\left(\begin{bmatrix} C_{\underline{\mathbf{zz}}}^* & 2C_{\underline{\mathbf{xy}}} \\ -jC_{\underline{\mathbf{zz}}}^* & 2C_{\underline{\mathbf{xx}}} \end{bmatrix}\right) = \det(C_{\underline{\mathbf{zz}}}^*)\det\left(2C_{\underline{\mathbf{xx}}} + jC_{\underline{\mathbf{zz}}}^*(C_{\underline{\mathbf{xy}}}^*)^{-1}2C_{\underline{\mathbf{xy}}}\right) \tag{D.24}$$

$$= \det(C_{\underline{\mathbf{zz}}}^*)\det\left(2C_{\underline{\mathbf{xx}}} + j2C_{\underline{\mathbf{xy}}}\right) \tag{D.25}$$

$$= \det(C_{\underline{\mathbf{zz}}}^*)\det(C_{\underline{\mathbf{zz}}}) = |\det(C_{\underline{\mathbf{zz}}})|^2. \tag{D.26}$$

Fourth order moments: For real-valued, zero-mean Gaussian processes, we find:

$$\mathrm{E}[x_1x_2x_3x_4] = \mathrm{E}[x_1x_2]\mathrm{E}[x_3x_4] + \mathrm{E}[x_1x_3]\mathrm{E}[x_2x_4] + \mathrm{E}[x_1x_4]\mathrm{E}[x_2x_3]$$

For complex-valued, zero-mean Gaussian processes, we find:

$$\mathrm{E}[x_1x_2^*x_3x_4^*] = \mathrm{E}[x_1x_2^*]\mathrm{E}[x_3x_4^*] + \mathrm{E}[x_1x_4^*]\mathrm{E}[x_2^*x_3]$$

In the following we derive the relation for real-valued processes. Consider the joint density function for the four random variables $x_1 ... x_4$:

$$f_{\underline{\mathbf{x}}}(\underline{x}) = \frac{1}{(2\pi)^2}\frac{1}{\sqrt{\det(R_{\underline{\mathbf{xx}}})}} e^{-\frac{1}{2}\underline{x}^T R_{\underline{\mathbf{xx}}}^{-1}\underline{x}}. \tag{D.27}$$

Due to the normalization of densities, we can modify the density function by

$$g(\alpha,\beta) = \frac{1}{(2\pi)^2}\frac{1}{\sqrt{\det(R_{\underline{\mathbf{xx}}})}} \int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} e^{-\frac{1}{2}\underline{x}^T \left[R_{\underline{\mathbf{xx}}}^{-1} + \begin{pmatrix} 0 & \alpha & 0 & 0 \\ \alpha & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta \\ 0 & 0 & \beta & 0 \end{pmatrix} \right] \underline{x}} d\underline{x}. \tag{D.28}$$

Obviously $g(0,0) = 1$ brings back the original density. We further introduce the matrix

$$L = \left[R_{\underline{\mathbf{xx}}}^{-1} + \begin{pmatrix} 0 & \alpha & 0 & 0 \\ \alpha & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta \\ 0 & 0 & \beta & 0 \end{pmatrix} \right]^{-1} \tag{D.29}$$

for which we have

$$g(\alpha,\beta)\sqrt{\frac{\det(R_{\underline{\mathbf{xx}}})}{\det(L)}} = 1 \tag{D.30}$$

independent of α and β. Differentiating $g(\cdot,\cdot)$ with respect to α and β delivers

$$\frac{\partial^2}{\partial\alpha\partial\beta} g(\alpha,\beta) = \int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} x_1x_2x_3x_4 f_{\underline{\mathbf{x}}}(\underline{x}) d\underline{x}, \tag{D.31}$$

thus exactly the desired expression. We thus have

$$\mathrm{E}[x_1x_2x_3x_4] = \frac{\partial^2}{\partial\alpha\partial\beta}\sqrt{\frac{\det(L)}{\det(R_{\underline{\mathbf{xx}}})}} \tag{D.32}$$

$$= \frac{\partial^2}{\partial\alpha\partial\beta}\frac{1}{\sqrt{\det\left(\mathbf{I} + R_{\underline{\mathbf{xx}}}\begin{pmatrix} 0 & \alpha & 0 & 0 \\ \alpha & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta \\ 0 & 0 & \beta & 0 \end{pmatrix}\right)}}. \tag{D.33}$$

We now have to find the determinant of the matrix:

$$\left(\mathbf{I} + R_{\underline{\mathbf{xx}}} \left(\begin{array}{cccc} 0 & \alpha & 0 & 0 \\ \alpha & 0 & 0 & 0 \\ 0 & 0 & 0 & \beta \\ 0 & 0 & \beta & 0 \end{array} \right) \right) = \left[\begin{array}{cccc} 1+\alpha r_{21} & \alpha r_{22} & \beta r_{23} & \beta r_{24} \\ \alpha r_{11} & 1+\alpha r_{12} & \beta r_{13} & \beta r_{14} \\ \alpha r_{41} & \alpha r_{42} & 1+\beta r_{43} & \beta r_{44} \\ \alpha r_{31} & \alpha r_{32} & \beta r_{33} & 1+\beta r_{34} \end{array} \right].$$

Development of the determinant in the first row we find the following terms:

$$\begin{aligned} \det(L) &= (1+\alpha r_{21}) \left| \begin{array}{ccc} (1+\alpha r_{12}) & \beta r_{13} & \beta r_{14} \\ \alpha r_{42} & (1+\beta r_{43}) & \beta r_{44} \\ \alpha r_{32} & \beta r_{33} & (1+\beta r_{34}) \end{array} \right| \\ &- \alpha r_{22} \left| \begin{array}{ccc} \alpha r_{11} & \beta r_{13} & \beta r_{14} \\ \alpha r_{41} & (1+\beta r_{34}) & \beta r_{44} \\ \alpha r_{31} & \beta r_{33} & (1+\beta r_{34}) \end{array} \right| \\ &+ \beta r_{23} \left| \begin{array}{ccc} \alpha r_{11} & 1+\alpha r_{12} & \beta r_{14} \\ \alpha r_{41} & \alpha r_{42} & \beta r_{44} \\ \alpha r_{31} & \alpha r_{32} & (1+\beta r_{34}) \end{array} \right| \\ &- \beta r_{24} \left| \begin{array}{ccc} \alpha r_{11} & 1+\alpha r_{12} & \beta r_{13} \\ \alpha r_{41} & \alpha r_{42} & (1+\beta r_{43}) \\ \alpha r_{31} & \alpha r_{32} & \beta r_{33} \end{array} \right| \\ &= h(\alpha,\beta). \end{aligned}$$

We now take advantage of symmetries $r_{ij} = r_{ji}$ and differentiate $h(\alpha,\beta)$ with respect to α and β:

$$\frac{\partial^2}{\partial\alpha\partial\beta}\frac{1}{\sqrt{h(\alpha,\beta)}} = \frac{3}{4}h(\alpha,\beta)^{-\frac{5}{2}}h_\alpha(\alpha,\beta)h_\beta(\alpha,\beta) - \frac{1}{2}h(\alpha,\beta)^{-\frac{3}{2}}h_{\alpha,\beta}(\alpha,\beta). \tag{D.34}$$

Here the first derivatives with respect to α and β are index accordingly. As we are interested in the result for $\alpha = 0$ and $\beta = 0$, we finally obtain

$$\begin{aligned} \mathrm{E}[x_1x_2x_3x_4] &= \frac{\partial^2}{\partial\alpha\partial\beta}\frac{1}{\sqrt{h(\alpha,\beta)}}\bigg|_{\alpha=0,\beta=0} && \text{(D.35)} \\ &= r_{12}r_{34} + r_{13}r_{24} + r_{14}r_{23}. && \text{(D.36)} \end{aligned}$$

It is worth to read the corresponding section in Papoulis' textbook where a much simpler derivation is shown. Note however that the path here can readily be modified to compute other terms, like six order moments.

Appendix E

Basics of linear Algebra

Consider two sets of vectors

$$\underline{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \underline{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \underline{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \underline{e}_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

and

$$\underline{g}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \underline{g}_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \underline{g}_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \underline{g}_4 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

As the vectors in both sets are linearly independent, each set builds a basis. Constructing a matrix out of the basis vectors

$$F = \begin{bmatrix} \underline{e}_1 & \underline{e}_2 & \underline{e}_3 & \underline{e}_4 \end{bmatrix}$$

and

$$G = \begin{bmatrix} \underline{g}_1 & \underline{g}_2 & \underline{g}_3 & \underline{g}_4 \end{bmatrix},$$

then the rows and columns of such matrices are linearly independent. The number of linear independent rows provides the row rank, the number of linearly independent columns the column rank of the matrix. The matrices F and G have both the rank four.

Both sets of vectors thus build a basis of a linear vector space of dimension four. While the first set $\{\underline{e}_1, \underline{e}_2, \underline{e}_3, \underline{e}_4\}$ is an *orthonormal* (=normalized to unity and orthogonal) basis, the second set $\{\underline{g}_1, \underline{g}_2, \underline{g}_3, \underline{g}_4\}$ is simply a basis of such space.

Let us now consider a matrix $H \in \mathbb{C}^{N \times M}$. Obviously, H describes a mapping of vectors $\underline{x} \in \mathbb{C}^{M \times 1}$ onto vectors $\underline{y} \in \mathbb{C}^{N \times 1}$:

$$\underline{y} = H\underline{x}. \tag{E.1}$$

The range space of the matrix H is that linear vector space that is spanned by his columns, thus

$$\mathcal{R}(H) = \{H\underline{x}|\underline{x} \in \mathbb{C}^{M\times 1}\} \tag{E.2}$$

That $\mathcal{R}(H)$ indeed is a linear space follows from the property that if we have for $\{\underline{x}_1, \underline{x}_2\}$ that $\{H\underline{x}_1, H\underline{x}_2\} \in \mathcal{R}(H)$, then it is also true that $c_1H\underline{x}_1 + c_2H\underline{x}_2 \in \mathcal{R}(H)$. Such properties do not allow to conclude whether the mapping (E.1) is unique that is that there are two different $\underline{x}_1$ and $\underline{x}_2$ that generate the same $\underline{y}$. If this is the case , we have $H[\underline{x}_1 - \underline{x}_2] = \underline{0}$. If such vectors exist, then their differences define the nullspace of H,

$$\mathcal{N} = \left\{\underline{x} \in \mathbb{C}^{M\times 1}|H\underline{x} = \underline{0}\right\}. \tag{E.3}$$

Repeating the above argument, we can show that the nullspace is a linear space too. In summary, we find that the uniqueness of a solution depends on the null space. If it is empty, then the solution is unique.

An orthogonal complement (space) $\mathcal{L}^\perp$ to a linear space $\mathcal{L}$ is given by

$$\mathcal{L}^\perp = \left\{\underline{x} \in \mathbb{C}^{M\times 1}|\underline{x}^H\underline{z} = 0, \text{for all } \underline{z} \in \mathcal{L}\right\}. \tag{E.4}$$

Lemma E.1 (column space and nullspace) *We have the following properties:* $\mathcal{N}(H) = \mathcal{R}(H^H)^\perp, \mathcal{N}(H^H) = \mathcal{R}(H)^\perp, \mathcal{R}(H) = \mathcal{N}(H^H)^\perp, \mathcal{R}(H^H) = \mathcal{N}(H)^\perp$.

Proof: Let us consider the first property. Let be $\underline{x} \in \mathcal{N}(H)$. Then we have also that $H\underline{x} = \underline{0}$. Then the following is also true $\underline{y}^H H\underline{x} = \underline{x}^H H^H\underline{y} = 0$ for all $\underline{y}$. Thus, $\underline{x}$ is orthogonal onto H^H, thus $\underline{x} \in \mathcal{R}(H^H)^\perp$ and as $\underline{x} \in \mathcal{N}(H)$, we must have $\mathcal{N}(H) \subseteq \mathcal{R}(H)^\perp$. Starting the argumentation with $\underline{x} \in \mathcal{R}(H^H)^\perp$, we thus conclude that $\underline{x} \in \mathcal{N}(H)$ and thus $\mathcal{R}(H^H)^\perp \subseteq \mathcal{N}(H)$. Therefore, it must be that $\mathcal{N}(H) = \mathcal{R}(H^H)^\perp$. The remaining properties follow accordingly.□

Lemma E.2 (Column spaces of H^H and H^HH) *We have:*

$$\mathcal{R}(H^H) = (H^HH).$$

Proof: With Lemma E.1 it is sufficient to show that $\mathcal{N}(H) = \mathcal{N}(H^HH)$. Let $\underline{x} \in \mathcal{N}(H^HH)$, then $H^HH\underline{x} = \underline{0}$. Thus, we also have $\underline{x}^HH^HH\underline{x} = \|H\underline{x}\|_2^2 = 0$. $H\underline{x} = \underline{0}$ is equivalent to $\underline{x} \in \mathcal{N}(H)$, and therefore $\mathcal{N}(H^HH) \subseteq \mathcal{N}(H)$. Let's start from the other end: let $\underline{x} \in \mathcal{N}(H)$, then $H\underline{x} = \underline{0}$ and also $H^HH\underline{x} = \underline{0}$. Thus we have $\underline{x} \in \mathcal{N}(H^HH)$ and therefore $\mathcal{N}(H) \subseteq \mathcal{N}(H^HH)$. Then, there is only possible that $\mathcal{N}(H) = \mathcal{N}(H^HH)$ and thus also $\mathcal{R}(H^H) = \mathcal{R}(H^HH)$.□

Appendix F

Method of Lagrange Multipliers

The method of Lagrange multipliers allows to formulating an optimization problem with side constraints into a simper form without side constraints. Let us consider a real-valued function $J(\underline{x})$ of an unknown vector $\underline{x} \in I\!R^{1\times n}$ and $f(\underline{x})$ a real-valued function of the same vector. Then the following optimization problem with constraint is given

$$\min_{\underline{x}} J(\underline{x}) \tag{F.1}$$

$$\text{with constraint} \quad f(\underline{x}) = b. \tag{F.2}$$

A solution $\underline{x}_o$ to this problem, if existent, will in general not be a stationary (or extremal) point of $J(\underline{x})$. It thus does not need to be point (in general it is not a point) for which the gradient of $J(\cdot)$ disappears.

Example F.1 *Let us assume that x is a scalar and that*

$$J(x) = x^2 + x - 2, f(x) = x^2, b = 1.$$

The solutions of $f(x) = 1$ are $x = \pm 1$, while the extrema point of $J(x)$ is at $x = -1/2$. Under the constraint $f(x) = 1$ we obtain two possible solutions $x = \pm 1$ of which $J(1) = 0$ and $J(-1) = -2$ only $x = -1$ minimized the function $J(x)$.

In general such elimination is not necessarily given and thus difficult to obtain. The method of the Lagrangian multipliers that we explain next, offers such a procedure which may lead to the desired result. We fist have to consider the differentials $J(\underline{x})$ and $f(\underline{x})$:

$$dJ(\underline{x}) = \frac{\partial J}{\partial x_1}dx_1 + \frac{\partial J}{\partial x_2}dx_2 + ... + \frac{\partial J}{\partial x_n}dx_n \tag{F.3}$$

$$df(\underline{x}) = \frac{\partial f}{\partial x_1}dx_1 + \frac{\partial f}{\partial x_2}dx_2 + ... + \frac{\partial f}{\partial x_n}dx_n. \tag{F.4}$$

If a solution $\underline{x}_o$ exists to this problem, we must have:

$$df(\underline{x}_o) = 0. \tag{F.5}$$

Also, we must have that

$$dJ(\underline{x}_o) = 0. \tag{F.6}$$

If we select the following linear combination of the two differentials we obtain the required (but not sufficient) condition for the existence of a minimum

$$\left(\frac{\partial J}{\partial x_1}(\underline{x}_o) - \lambda_1 \frac{\partial f}{\partial x_1}(\underline{x}_o)\right) dx_1 + \left(\frac{\partial J}{\partial x_2}(\underline{x}_o) - \lambda_2 \frac{\partial f}{\partial x_2}(\underline{x}_o)\right) dx_2 + ... + \left(\frac{\partial J}{\partial x_n}(\underline{x}_o) - \lambda_n \frac{\partial f}{\partial x_n}(\underline{x}_o)\right) dx_n = 0. \tag{F.7}$$

We thus no longer require that each term $\frac{\partial f}{\partial x_n}(\underline{x}_o)$ becomes zero, but it suffices to select the variables λ_n so that the terms $\left(\frac{\partial J}{\partial x_n}(\underline{x}_o) - \lambda_n \frac{\partial f}{\partial x_n}(\underline{x}_o)\right) dx_n$ become zero.

Example F.2 *Let us assume x to be a scalar and that*

$$J(x) = x^2 + x - 2, f(x) = x^2, b = 1.$$

A necessary condition for the solution is

$$[(2x_o + 1) - 2\lambda x_o]dx = 0$$

for arbitrary λ. We select λ, so that

$$(2x_o + 1) - 2\lambda x_o = 0,$$

for which we obtain by substituting x_0 in $f(x)$

$$\frac{1}{4(\lambda - 1)^2} = 1.$$

This provides two solutions for $\lambda = \{1/2, 3/2\}$ and thus $x = \pm 1$. Trying the possible solutions leads to the single true solution $x_o = -1$.

Summarizing we can say that the method of Lagrangian multipliers transforms an optimization problem $J(\underline{x})$ with constraint $f(\underline{x}) = 0$ into a new optimization problem

$$V(\underline{x}, \underline{\lambda}) = J(\underline{x}) - \sum_{i=1}^{n} \lambda_i [f_i(\underline{x}) - b_i] \tag{F.8}$$

the solutions of which are given by

$$dV(\underline{x}_o) = 0, f_i(\underline{x}_o) = b_i.$$

Obviously this is a necessary but not a sufficient condition.

In case of complex-valued constraints ($f(\underline{x}) \in \mathbb{C}$), it can be substituted by two conditions:

$$\begin{aligned} V(\underline{x}, \lambda_R, \lambda_I) &= J(\underline{x}) - \lambda_R[f_R(\underline{x}) - b_R] - \lambda_I[f_I(\underline{x}) - b_I] && \text{(F.9)} \\ &= J(\underline{x}) - Re\{\lambda^*[f(\underline{x}) - b]\} && \text{(F.10)} \\ &= J(\underline{x}) - \lambda^*[f(\underline{x}) - b] - \lambda[f(\underline{x}) - b]^*, && \text{(F.11)} \end{aligned}$$

using a complex-valued $\lambda = \lambda_R + j\lambda_I$.

Appendix G

State Space Description of Systems

A linear, time-variant system can be described by

$$\underline{x}_{k+1} = F_k \underline{x}_k + G_k \underline{u}_k, \quad \underline{x}_{k_0} = \text{initial value} \tag{G.1}$$

$$\underline{y}_k = H_k \underline{x}_k + K_k \underline{u}_k, \quad k \geq k_0, \tag{G.2}$$

where the matrices $\{F_k, G_k, H_k, K_k\}$ are of dimensions $n \times n, n \times q, p \times n$ and $p \times q$, respectively. Correspondingly, $\underline{u}_k$ is of dimension $q \times 1$ and $\underline{y}_k$ of dimension $p \times 1$. The $n-$ dimensional vector $\underline{x}_k$ is called the state of the system.

State $\underline{x}_k$ can be computed directly by transition matrix $\Phi(k, j)$:

$$\underline{x}_k = \Phi(k, j)\underline{x}_j + \sum_{l=j}^{k-1} \Phi(k, l+1) G_l \underline{u}_l. \tag{G.3}$$

The transition matrix is given by

$$\Phi(k, j) = F_{k-1} F_{k-2} ... F_j, \; \Phi(k, k) = I. \tag{G.4}$$

For the special case of a time-invariant system $\{F, G, H, K\}$, we find for the transition matrix

$$\Phi(k, j) = F^{k-j}, k \geq j. \tag{G.5}$$

Under many circumstances the time-invariant system with transition matrix can be diagonalized.

$$F = \text{diag}\{\lambda_1, \lambda_2, ..\lambda_n\}.$$

Assuming we write G as column vector with elements $\beta_i, i = 1..n$, then the pair $\{F, G\}$ is called controllable if all λ_i are different from each other and all β_i are unequal to zero. By this condition an input signal u_k can impact all states $\underline{x}_k$. Similarly, we like to know if each state $\underline{x}_k$ can impact each output value $\underline{y}_k$. This property is defined by the pair $\{F, H\}$. If $\{F, H\}$ is observable then $\{F^*, H^*\}$ is controllable. Furthermore, the stability of such system is of interest. The stability is guaranteed if all eigenvalues$|\lambda_i| < 1$.

Appendix H

Small-Gain Theorem

The small gain theorem can be interpreted as the generalization of stability in the linear case. It is well-known for linear time-invariant systems that a closed loop system is stable if and only if the open loop system has a gain smaller than one. The small gain theorem extends such statement towards arbitrary nonlinear systems.Consider an input signal $x_k, k = 1..N$ described by the vector $\underline{x}_N$ of dimension $1 \times N$. The response of a system H_N on such a signal is given by

$$\underline{y}_N = H_N \underline{x}_N. \tag{H.1}$$

Definition 1: A mapping H is called l-stable, if two positive constants γ, β exist, such that for all input signals $\underline{x}_N$ the output is upper bounded by:

$$\|\underline{y}_N\| = \|H_N \underline{x}_N\| \leq \gamma \|\underline{x}_N\| + \beta. \tag{H.2}$$

Definition 2: The smallest positive constant γ, which satisfies l-stability is called the gain of the system H_N.

Remark: So-called bounded input-bounded output (BIBO) stability is another expression for l_∞-stability.

Let us consider now a feedback system with two components H_N and G_N of individual gains γ_h and γ_g. We find:

$$\underline{y}_N = H_N \underline{h}_N = H_N[\underline{x}_N - \underline{z}_N] \tag{H.3}$$

$$\underline{z}_N = G_N \underline{g}_N = G_N[\underline{u}_N + \underline{y}_N]. \tag{H.4}$$

Theorem H.1 (Small Gain Theorem) *If the gains γ_h and γ_g are such that*

$$\gamma_h \gamma_g < 1, \tag{H.5}$$

then the signals $\underline{h}_N$ *and* $\underline{g}_N$ *are upper bounded by*

$$\|\underline{h}_N\| \leq \frac{1}{1-\gamma_h\gamma_g}\left[\|\underline{x}_N\| + \gamma_g\|\underline{u}_N\| + \beta_h + \gamma_g\beta_h\right] \tag{H.6}$$

$$\|\underline{g}_N\| \leq \frac{1}{1-\gamma_h\gamma_g}\left[\|\underline{u}_N\| + \gamma_h\|\underline{x}_N\| + \beta_g + \gamma_h\beta_g\right]. \tag{H.7}$$

Proof: We have

$$\underline{h}_N = \underline{x}_N - \underline{z}_N \tag{H.8}$$

$$\underline{g}_N = \underline{u}_N + \underline{y}_N. \tag{H.9}$$

Thus for the norm

$$\|\underline{h}_N\| \leq \|\underline{x}_N\| + \|G_N\underline{g}_N\| \tag{H.10}$$

$$\leq \|\underline{x}_N\| + \gamma_g\|\underline{g}_N\| + \beta_g \tag{H.11}$$

$$\leq \|\underline{x}_N\| + \gamma_g[\|\underline{u}_N\| + \gamma_h\|\underline{h}_N\| + \beta_h] + \beta_g \tag{H.12}$$

$$= \gamma_g\gamma_h\|\underline{h}_N\| + \|\underline{x}_N\| + \gamma_g\|\underline{u}_N\| + \gamma_g\beta_h + \beta_g \tag{H.13}$$

$$= \frac{1}{1-\gamma_g\gamma_h}[\gamma_g\|\underline{u}_N\| + \gamma_g\beta_h + \beta_g]. \tag{H.14}$$

The corresponding relation for $\underline{g}_N$ can be shown in the same way.

Bibliography

[1] Y.Bengio, Y.LeCun, G.Hinton, "Deep Learning," Nature, vol. 521 pp.436-444, 2015.

[2] A.Bahai, M.Rupp, "Training and tracking of adaptive DFE algorithms under IS-136," SPAWC97 in Paris, April 1997.

[3] Behrooz and Parhami , "Computer Arithmetic," Oxford University Press, 2000.

[4] N.J. Bershad, "Analysis of the normalized LMS algorithm with Gaussian inputs," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. ASSP–34, No. 4, pp. 793–806, Aug. 1986.

[5] Neil J. Bershad, "Behavior of the ϵ–normalized LMS algorithm with Gaussian inputs," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP–35, No. 5, pp. 636–644, May 1987.

[6] Dariusz Bismor, Krzysztof Czyz and Zbigniew Ogonowski, "Review and Comparison of Variable Step-Size LMS Algorithms," International Journal of Acoustics and Vibration, Vol. 21, No. 1, pp. 24-39, 2016.

[7] S. Bozinovski, "Teaching space: A representation concept for adaptive pattern classification," COINS Technical Report, University of Massachusetts at Amherst, No. 81-28, 1981. `UM-CS-1981-028.pdf`

[8] Helmut Brehm, "Description of spherically invariant random processes by means of G–functions," Springer lecture Notes, vol. 969, pp. 39–73, 1982.

[9] H.-J. Butterweck, "Iterative analysis of the steady-state weight fluctuations in LMS-type adaptive filters," IEEE Transactions on Signal Processing, vol. 47, pp. 2558-2561, Sept. 1999.

[10] H.-J. Butterweck, "A wave theory of long adaptive filters," IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, vol. 48, pp. 739-747, June 2001.

[11] Theo A. C. M. Claasen, Wolfgang F. G. Mecklenbräuker, "Comparison of the convergence of two algorithms for adaptive FIR digital filters," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP–29, No. 3, pp. 670–678, Juni 1981.

[12] Peter M. Clarkson, Paul R. White, "Simplified analysis of the LMS adaptive filter using a transfer function approximation," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP–35, No. 7, pp. 987–993, Juli 1987.

[13] S. C. Douglas, T. H.–Y. Meng, "Exact expectation analysis of the LMS adaptive filter without the independence assumption," Proc. ICASSP, San Francisco, pp. IV61–IV64, Apr. 1992.

[14] J.C.Doyle, K.Glover, P.Kargonekar, B.Francis, "State-space solutions to standard H_2 and H_∞ control problems, IEEE Transactions on Automatic Control, Vol. 34, No. 8, pp. 831-847,1989.

[15] D.L. Duttweiler,"Adaptive filter performance with nonlinearities in the correlation multiplier," *IEEE Trans. Acoust., Speech, Signal Processing*, vol.ASSP-30, pp. 578-586, Aug. 1982.

[16] D.L. Duttweiler, "Proportionate Normalized Least-Mean-Squares Adaptation in Echo Cancelers," *IEEE Trans. on Speech and Audio Processing,* vol. 8,no.5, pp. 508-518, Sep. 2000.

[17] E. Eleftheriou, D.D. Falconer, "Tracking properties and steady-state performance of RLS adaptive filter algorithms," *IEEE Trans. Acoustics, Speech and Signal Proc.*, vol. ASSP-34, No. 5, pp. 1097-1110, Oct. 1986.

[18] D.C. Farden, "Tracking Properties of adaptive processing algorithms," *IEEE Trans. Acoustics, Speech and Signal Proc.*, vol. ASSP-29, No. 6, pp. 439-446, June 1981.

[19] P. L. Feintuch, "An adaptive recursive LMS filter," Proc. IEEE, vol. 64, No. 11, pp. 1622–1624, Nov. 1976.

[20] A. Feuer and E. Weinstein, "Convergence analysis of LMS filters with uncorrelated Gaussian data," *IEEE Trans. Acoust. Speech and Signal Processing,* vol. ASSP–33, No. 1, pp. 222–230, Feb. 1985.

[21] R. Frenzel, M.E. Hennecke, Using prewhitening and stepsize control to improve the performance of the LMS algorithm for acoustic echo compensation, 1992 IEEE International Symposium on Circuits and Systems, ISCAS '92. Proceedings., Vol. 4 , pp. 1930 -1932, 1992.

[22] W.A Gardner, "Learning characteristics of stochastic gradient descent algorithms: a general study, analysis and critique," Signal Processing, pp. 113-133, Apr. 1984.

[23] S.L.Gay, "An efficient fast converging adaptive filter for network echo cancelling, *Proc. of Asilomar Conference,*, Monterey, Nov. 1998.

[24] John R. Glover, Jr. , "Adaptive noise canceling applied to sinusoidal interferences," IEEE Trans. Acoust., Speech, Signal Processing, vol ASSP–25, no. 6., pp. 484–491, Dec. 1977.

[25] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy, "EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES," Proc. of ICLR, 2015.`https://arxiv.org/pdf/1412.6572.pdf`

[26] R.M. Gray, Toeplitz and Circulant Matrices: A Review, now publisher, 2007.

[27] B. Haetty, "Recursive Least Squares Algorithms using Multirate Systems for Cancellation of Acoustical Echoes," Proc. ICASSP, Albuquerque, New Mexico, USA, pp. 1145-1148, 1990.

[28] M. Hajivandi, W.A. Gardner, "Measures of tracking performance for the LMS algorithm," *IEEE Trans. Acoustics, Speech and Signal Proc.*, vol. ASSP-38, No. 11, pp. 1953-1958, Nov. 1990.

[29] B. Hassibi, A.H. Sayed, and T. Kailath, "LMS and Backpropagation are minimax filters," in *Neural Computation and Learning*, ed. V. Roychowdhury, K. Y. Siu, and A. Orlitsky, Ch. 12, pp. 425–447, Kluwer Academic Publishers, 1994.

[30] B. Hassibi, A.H. Sayed, and T. Kailath, "H^∞ optimality of the LMS algorithm," *IEEE Trans. Signal Processing*, Vol. 44, No. 2, pp. 267–280, Feb. 1996.

[31] S. Haykin, *Neural Networks: A Comprehensive Foundation*, MacMillan Publishing Company, 1994.

[32] Simon Haykin, *Adaptive Filter Theory*, 1. edition, Prentice Hall 1986.

[33] Simon Haykin, *Adaptive Filter Theory*, 3. edition, Prentice Hall 1996.

[34] Simon Haykin, *Adaptive Filter Theory*, 4. edition, Prentice Hall 2002.

[35] L.L.Horowitz and K.D.Senne, "Performance advantage of complex LMS for controlling narrow-band adaptive arrays," IEEE Transactions on Signal Processing, Vol. 29 , No. 3, pp.722-736, June 1981.

[36] S. Hui, S.H. Zak, "The Widrow-Hoff algorithm for McCulloch-Pitts type neurons," *IEEE Transactions on Neural Networks*, Vol. 5, No. 6, pp. 924-929, Nov. 1994.

[37] D.R. Hush, B.G. Horne, "Progress in supervised neural networks," *IEEE Signal Processing Magazine,* Vol. 10, No. 1, pp. 8-39, Jan. 1993.

[38] T.Kailath, A.H.Sayed, B.Hassibi, Linear Estimation, Prentice Hall, 1999.

[39] R. E. Kalman, "Design of a self–optimizing control system," Trans. ASME, Vol. 80, pp. 468–478, 1958.

[40] W. Kellermann, "Analysis and Design of Multirate Systems for Cancellation of Acoustical Echoes," Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, NY, 1988, Vol. 5, pp. 2570-2573.

[41] P.P.Khargonekar, K.M. Nagpal, "Filtering and smooting in an $H^\infty-$ setting, IEEE Transactions on Automatic Control, vol. 36, pp. 151-166, 1991.

[42] H.K. Khalil, *Nonlinear Systems*, Mac Millan, 1992.

[43] R.P. Lippmann, "An introduction to computing with neural nets," *IEEE Acoustics, Speech and Signal Processing Magazine,* Vol. 4, No. 2, pp.4-22, April 1987.

[44] Guozho Long, Fuyun Ling, John A. Proakis, "The LMS algorithm with delayed coefficient adaptation," IEEE Trans. Acoust., Speech, Signal Processing, Vol. ASSP–37, No. 9, pp. 1397–1405, Sep. 1989.

[45] Guozho Long, Fuyun Ling, John A. Proakis, "Corrections to 'The LMS algorithm with delayed coefficient adaptation'," IEEE Trans. Signal Processing, Vol. SP–40, No. 1, pp. 230–232, Jan. 1992.

[46] Vijay K. Madisetti: Editor, *The DSP Handbook*, CRC Press, 1997.

[47] J. Mai, Ali H. Sayed, "A feedback approach to the steady-state performance of fractionally-spaced blind adaptive equalizers, *IEEE Trans. on Signal Processing*, vol. 48, no.1, pp. 80-91, Jan. 2000.

[48] S. Makino, Y. Kaneda and N. Koizumi, "Exponentially weighted step-size NLMS adaptive filter based on the statistics of a room impulse response, *IEEE Trans. on Speech and Audio Processing,* vol. 1, No. 1, pp. 101–108, Jan. 1993.

[49] S. Marcos, O. Macchi, "Tracking capability of the least mean square algorithm: application to an asynchronous echo canceller, *IEEE Trans. Acoustics, Speech and Signal Proc.*, vol. ASSP-35, No. 11, pp. 1570-1578, Nov. 1987.

[50] J. E. Mazo, "On the independence theory of equalizer convergence, *Bell Syst. Tech. J.,* vol. 58, pp. 963–993, 1979.

[51] H. Mohamad, S. Weiss, M. Rupp, L. Hanzo, "A fast converging fractionally spaced equalizer, Asilomar conference, Nov. 2001.

[52] H. Mohamad, S. Weiss, M. Rupp, L. Hanzo, "Fast adaptation of fractionally spaced equalizers," Electronic Letters, vol. 38, no.2, p. 96-98, 17. Jan. 2002.

[53] K.S. Narendra, K. Parthasarathy, "Identification and control of dynamical systems using neural networks, *IEEE Transactions on Neural Networks*, vol. 1, p. 4-27, March 1990.

[54] K.S. Narendra, K. Parthasarathy, "Gradient methods for the optimization of dynamical systems containing neural networks, *IEEE Trans. on Neural Networks*, vol. 2, p. 252-262, March 1991.

[55] R. Nitzberg, "Normalized LMS algorithm degradation due to estimation noise," IEEE Trans. Aerosp. Electron. Syst., Vol. AES–22, No. 6., p. 740–749, Nov. 1986.

[56] K. Ozeki, T. Umeda, "An adaptive filtering algorithm using orthogonal projection to an affine subspace and its properties," Electronics and Communications in Japan, Vol. 67–A, No. 5, pp. 19–27, 1984.

[57] M. Rupp, "The behavior of LMS and NLMS algorithms in the presence of spherically invariant processes," IEEE Trans. Signal Processing, Vol. SP–41, No. 3, pp. 1149-1160, March 1993.

[58] M. Rupp, R. Frenzel "The behavior of LMS and NLMS algorithms with delayed coefficient update in the presence of spherically invariant processes," IEEE Trans. Signal Processing, Vol. SP–42, No. 3, pp. 668-672, March 1994.

[59] M. Rupp, "Bursting in the LMS algorithm," IEEE Transactions on Signal Processing, Vol. 43, No. 10, pp. 2414-2417, Oct. 1995.

[60] M. Rupp, A. H. Sayed, "On the stability and convergence of Feintuch's algorithm for adaptive IIR filtering," *Proc. IEEE Conf. ASSP*, Detroit, MI, May 1995.

[61] M. Rupp, A.H. Sayed, "A robustness analysis of Gauss-Newton recursive methods," Signal Processing, Vol. 50, No. 3, pp. 165-188, June 1996.

[62] M. Rupp, A.H. Sayed, "A time-domain feedback analysis of filtered-error adaptive gradient algorithms," IEEE Transactions on Signal Processing, Vol. 44, No. 6, pp. 1428-1440, June 1996.

[63] M. Rupp, "Saving complexity of modified filtered-X-LMS and delayed update LMS algorithm," IEEE Transactions on Circuits & Systems, pp. 57-59, Jan. 1997.

[64] M.Rupp, A.H. Sayed, "Improved convergence performance for supervised learning of perceptron and recurrent neural networks: a feedback analysis via the small gain theorem," IEEE Transaction on Neural Networks, Vol. 8, No. 3, pp. 612-623, May 1997.

[65] M. Rupp, A.H. Sayed, "Robust FxLMS algorithm with improved convergence performance," IEEE Transactions on Speech and Audio Processing, vol. 6, No. 1, pp. 78-85, Jan. 1998.

[66] M. Rupp, "A family of adaptive filter algorithm with decorrelating properties," IEEE Transactions on Signal Processing, vol. 46, No. 3, pp. 771-775, March 1998.

[67] M. Rupp, "On the learning behavior of decision feedback equalizers," 33rd. Asilomar Conference, Monterey, California, Oct. 1999

[68] M. Rupp, A.H. Sayed, "On the convergence of blind adaptive equalizers for constant modulus signals" IEEE Transactions on Communications, vol. 48, No. 5, pp. 795-803, May 2000.

[69] M. Rupp, J. Cezanne, "Robustness conditions of the LMS algorithm with time-variant matrix step-size," Signal Processing, vol. 80, No. 9, Sept. 2000.

[70] M. Rupp and H.-J. Butterweck, "Overcoming the independence assumption in LMS filtering," in Proc. of Asilomar Conference, pp. 607-611, Nov. 2003.

[71] A.H. Sayed and T. Kailath, "A state-space approach to adaptive RLS filtering," *IEEE Signal Processing Magazine*, vol. 11, No. 3, pp. 18–60, July 1994.

[72] A.H. Sayed, M. Rupp, "Error energy bounds for adaptive gradient algorithms," IEEE Transactions on Signal Processing, vol. 44, No. 8, pp. 1982-1989, Aug. 1996.

[73] A.H.Sayed, M. Rupp, "An l_2-stable feedback structure for nonlinear H^∞-adaptive filtering," Automatica, vol. 33, No.1, pp. 13-30, Jan. 1997.

[74] V.H.Nascimento, A.H.Sayed, "Are ensemble averaging learning curves reliable in evaluating the performance of adaptive filters," Proceedings 32nd Asilomar Conference on Circuits, Systems, and Computers, pp. 1171-1174, Nov. 1998.

[75] A.H. Sayed, "Fundamentals of Adaptive Filtering," Wiley 2003.

[76] Solo, V. and X. Kong, *Adaptive Signal Processing Algorithms: Stability and Performance*, Prentice Hall, New Jersey, 1995.

[77] S.D. Stearns, G. R. Elliot, "On adaptive recursive filtering," Proceedings 10th Asilomar Conference on Circuits, Systems, and Computers, pp. 5–11, Nov. 1976.

[78] Stehman, Stephen V.. "Selecting and interpreting measures of thematic classification accuracy". Remote Sensing of Environment. 62 (1), pp. 77-89, 1997, doi:10.1016/S0034-4257(97)00083-7.

[79] K. Steiglitz, L. E. McBride, "A technique for the identification of linear systems," IEEE Trans. Autom. Control, vol. AC–10, pp. 461–464, 1965.

[80] S.Theodoridis, Machine Learning: A Bayesian and Optimization Perspective, 2nd edition, 2020.

[81] G. Ungerboeck, "Theory on the speed of convergence in adaptive equalizers for digital communication," IBM J. Res. Develop., vol. 16, no. 6, pp. 546–555, 1972.

[82] M. Vidyasagar, *Nonlinear Systems Analysis*, Prentice Hall, New Jersey, second edition, 1993.

[83] S. A. White, "An adaptive recursive digital filter," Proc. 9th Asilomar Conf. Circuits Syst. Comput., pp. 21–25, Nov. 1975.

[84] Bernard Widrow, M. E. Hoff Jr., "Adaptive switching circuits," IRE WESCON conv. Rec., Part 4, pp. 96–104, 1960.

[85] E. Walach and B. Widrow, "The least-mean-fourth (LMF) adaptive algorithm and its family," *IEEE Trans. Inform. Theory*, vol. IT-30, pp.275-283, March 1984.

[86] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*, NY: Prentice-Hall, Inc., 1985.

[87] W. Wirtinger, "Zur formalen Theorie der Funktionen von mehr komplexen Veränderlichen," Mathematische Annalen, vol.. 97, pp. 357-376, 1927.

[88] Watt, Borhani, Katsegallos, "Machine Learning Refined" 2020, (1.st and 2nd edition)

[89] Zoubir, A., Iskander, D. (2004). Bootstrap Techniques for Signal Processing. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511536717