Master's Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Radioelectronics

Machine Learning Algorithms in Wireless Physical Layer Network Coding

Bc. Jakub Kolář
Open Electronic Systems

Supervisor: Prof. Ing. Jan Sýkora, CSc.
Field of study: Communications and Signal Processing
January 2020

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Kolář Jakub**	Personal ID number:	**434776**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Radioelectronics**		
Study program:	**Open Electronic Systems**		
Branch of study:	**Communications and Signal Processing**		

II. Master’s thesis details

Master’s thesis title in English:

Machine Learning Algorithms in Wireless Physical Layer Network Coding

Master’s thesis title in Czech:

Algoritmy strojového učení pro bezdrátové síťové kódování fyzické vrstvy

Guidelines:

Student will get acquainted with fundamental principles of machine learning algorithms and with fundamentals of Wireless Physical Layer Network Coding (WPNC). Student should systematically present the background machine learning theory from the particular perspective of digital communication application. The core of the work should focus on the utilisation of learning methods in scenarios with undefined or weakly defined system model. It particularly applies to codebook and constellation used at the transmit side, and the network channel model and topology. Student will first apply (at least at the theoretical concept level) the learning methods on simple point-to-point scenarios (e.g. learning the codebook for unknown channel model), then on simple multi-user and/or WPNC scenarios (e.g. source constellation/modulation classification, learning the relay hierarchical network code maps for end-to-end WPNC solvability, etc). Selected simple algorithms should be implement and verified by a computer simulation.

Bibliography / sources:

[1] J. Sykora, A. Burr: Wireless Physical Layer Network Coding, Cambridge University Press 2018
[2] T. O’Shea and J. Hoydis, “An introduction to deep learning for the physical layer,” IEEE Transactions on Cognitive Communicationsand Networking, vol. 3, pp. 563–575, Dec 2017.
[3] Christopher M. Bishop: Pattern Recognition and Machine Learning, Springer 2006

Name and workplace of master’s thesis supervisor:

prof. Ing. Jan Sýkora, CSc., Department of Radioelectronics, FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **17.09.2019** Deadline for master's thesis submission: ___________

Assignment valid until: **19.02.2021**

prof. Ing. Jan Sýkora, CSc. Supervisor’s signature	doc. Ing. Josef Dobeš, CSc. Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

. Date of assignment receipt	Student’s signature

Acknowledgements

I would like to thank to Prof. Ing. Jan Sýkora, CSc. for the supervision of this thesis. Further, I want to express my sincere gratitude to my family and friends for continuous support during the studies. A very special thanks goes to my girlfriend Eva for being so special to me ♡.

Declaration

I declare that I completed the presented thesis independently and that all used sources are quoted in accordance with the Methodological Instructions that cover the ethical principles for writing an academic thesis.

In Prague, 7. 1. 2020

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 7. 1. 2020

....

Abstract

This thesis deals with an application of machine learning (ML) algorithms in wireless physical layer network coding (WPLNC). An introductory part of the text provides a general overview of ML methods, a motivation for utilization of ML approaches, and a very brief summary of principles of WPLNC. Artificial neural networks (ANN) recently attracted attention in the field of communications, with a broad range of promising applications. Basic principles and backpropagation training procedure of ANN was hereby addressed and implemented. Finally, several exemplary problems in basic WPLNC scenarios were stated, and solutions were proposed and tested as computer simulations. These scenarios focused on an issue of classification of hierarchical symbols in a two-way relay channel with BPSK and QPSK modulations and considered a variable parameter of relative fading. The obtained results showed that the trained systems based on ANN are capable of performing these tasks, and performance was evaluated. An effort was to tune the parameters of the trained system and to provide a clear visual representation of the results.

Keywords: Wireless physical-layer network coding, Machine learning, Artificial neural networks, Backpropagation, Two-way relay channel, Decision regions

Supervisor: Prof. Ing. Jan Sýkora, CSc.

Abstrakt

Tato práce se zabývá aplikacemi algoritmů strojové učení na bezdrátové síťové kódování fyzické vrstvy. Úvodní část textu poskytuje obecný přehled metod strojové učení, motivaci pro využití přístupu strojového učení a velmi stručný přehled principů bezdrátového síťového kódování fyzické vrstvy. Umělé neuronové sítě během posledních letech přitahovaly v oboru komunikace pozornost velkým rozsahem slibných aplikací. Dále jsou zde popsány a implementovány základní principy a metody trénování umělých neuronových sítí. Následně bylo popsáno několik ukázkových úloh v základních scénářích bezdrátového síťového kódování fyzické vrstvy a byla navržena a počítačovými simulacemi testována jejich řešení. Tyto scénáře se zaměřily především na problém klasifikace hierarchického symbolu v dvousměrném reléovém kanálu s modulacemi BPSK a QPSK. Uvažován byl také proměnný parametr relativního tlumení. Dosažené výsledky ukázaly, že natrénovaný systém založený na umělé neuronové síti je schopen tyto úkoly plnit a byla vyhodnocena jejich výkonnost. Byla také vynaložena snaha vyladit parametry natrénovaného systému a poskytnout jasnou vizuální reprezentaci výsledků.

Klíčová slova: Bezdrátové síťové kódování fyzické vrstvy, strojové učení, umělé neuronové sítě, backpropagation, dvousměrný kanál s reléovým uzlem, rozhodovací oblasti

Contents

Figures

Tables

Chapter 1

Introduction

This thesis is about the connection between two topics: Machine Learning (ML) and its possible applications to Wireless Physical Layer Network Coding (WPLNC). Currently, ML is gaining interest in many fields. In the beginning, we shall provide several examples that show how the ML methods might be useful in the broad area of communications. Based on the current literature and providing a brief survey of the state of the art methods, we will try to give a tutorial on a selected topic of ML and connect its applications with the typical scenarios in wireless communications, especially on the physical layer.

Next, the fundamental ideas of WPLNC shall be briefly stated with an aim to select a target field of the applications. This background of WPLNC will be used to state an exemplary problem definition to be solved with a selected ML method. For the applications, focus in this thesis shall be given to provide a clear explanation of the methods and approaches. Note, the performance of the implementations shall not be a key criterion.

The goal of this thesis is to provide a description of the ML algorithms that are suitable to be applied in the scenarios of WPLNC and to implement some of these algorithms as computer simulations.

The theoretical part of the thesis will be focused on the utilization of Artificial Neural Networks. Based on the current literature, a basic description shall be provided together with a detailed explanation of the training procedure.

Finally, the last chapter shall contain several commented examples, implemented as computer simulations.

Chapter 2

Machine Learning: An Overview

2.1 Preface

Machine learning is a broad area of methods, and hereby we will try to provide an introduction to the topic based on a study of several references such as [1] [2] [3]. In an effort to avoid problems with notation, it is adopted mainly from [1]. The title of [2] is "Pattern Recognition and Machine Learning," and let us say, that these topics are strongly connected. Pattern recognition uses concepts of machine learning, and its task is typically to determine regularities in data automatically and, e.g., classify the data to different categories [1]. A very common area of usage of the machine learning is concentrated in the fields of image and speech analysis [1] [2] [3]. Corresponding to this fact, most of the literature takes e.g., a popular picture classification [2] as an introductory tutorial example. Keeping in mind the main topic of this thesis, which concerns wireless digital communication, we will in the text partially adopt also literature from other fields. The justification is that the mathematical formulations are obviously general and blind to specific inputs of the machine learning approaches and algorithms.

In this chapter, we shall provide a background of general machine learning methods.

2.2 Introduction

The task of machine learning is to find an optimized solution of programming tasks based on example data sets or past experience; especially, it is suitable in situations when we can not simply write code to solve the problem because of a lack of knowledge about the situation [1]. The first area of usage is when the model does not exist at all, e.g., converting speech to written text or identifying handwritten digits, when the human can perform this task easily, but can not generally describe the knowledge in order to implement the solution using the computer. The complexity of these tasks is in the diversity of the input data. For example, different styles of the handwriting of letters and digits in optical character recognition, or different parameters of pronunciations of words in speech analysis are the reasons, why these tasks are difficult to be solved in a traditional way – without use of machine learning methods [1]. Then, a method of machine learning is generally to inspect a large amount of data and discover the corresponding mappings that are difficult to be described [1]. Another common assumption of the machine learning paradigm is that the properties of the system might be varying and therefore, the system should be adaptive with respect to the specific conditions – this is different from fixed tuned system [1]. The given examples of speech and image recognition systems are exemplary approaches that are already commercially successful and based on machine learning. Applications of pattern recognition algorithms in the fields of image and speech analysis became very popular in recent years, where the success was driven by the availability of the huge amount of data and large computation power [4].

The field of machine learning developed as a union of many fields, where the methods were discovered for different purposes [1]: statistics, pattern recognition, signal processing, data mining and neural networks, to name few.

Next, let us introduce the suitable use cases of machine learning, because we must not expect that it provides us 'one-fits-all' solution to the problems. Conventional engineering workflow typically means, that the background knowledge of the problem to be solved, e.g., the physical description, is used to produce a mathematical model of the situation, then the algorithm is designed and optimized to find the solution and performance bounds can be then stated to evaluate the results [4]. This is probably the best we can do. On the other side, fundamentally the machine learning approach might allow us to replace the step of designing the model of the system with a possibly easier task of collecting large amount of data, called training set, that is used by the 'machine' to 'learn' the solution of the task [4]. Note, this way is suitable only in case, that the description of the task is not available or is extremely difficult to formulate and this is the fundamental motivation for the topic of this thesis with respect to wireless digital communication. Usage of machine learning is inappropriate in cases that the model of the situation is known and is suitable to find an approximation of the model that corresponds to the data [1].

According to [3], it is advisable to consider the machine learning approach in case that:

- a conventional approach is not desirable from the reasons of an unavailable physical model of the situation or when the algorithmic solutions are too complex to be applied;
- the training sets are available or can be created;
- the task to be learned is stationary and does not change rapidly in time;
- an explanation of the decision is not required, i.e., a black box realization is sufficient;
- clear metric evaluating the solution can be defined.

It is also possible and desirable to choose and design the specific method of machine learning such that it utilizes the background knowledge [4]. Moreover, while the machine learning based on the training data can find a solution without a mathematical model, the solution will be typically found suboptimal with respect to a solution based on a relevant model [3].

Considering a general machine learning task briefly, we are initially given a *training set* of data, that is used to *train* the algorithm to find a possible solution; considering a step called *generalization*, we then use additional *test set* to determine, if the the received solution is suitable [2]. Generalization is fundamental to machine learning because it distinguishes the process from pure memorization of the training set. It assumes that the found model will be valid also for subsequent inputs. The assumptions that allow us to perform the generalization are called *inductive bias* [1]. More details follow in the next sections.

2.3 Basic Terminology

Generally, to perform a solution using a computer, we first need an algorithm, and we want to use the most efficient one, e.g., from the point of view of memory or number of instructions [1]. In machine learning, we want the computer to find the parameters for a method using the training data to compensate for the deficit of our knowledge in cases when we do not have any better approach. Especially, this holds in situations when we suppose that the process is not completely random, i.e., there are some rules or patterns, and therefore we assume that there exists a process that explains the origin of the testing data [1]. In this section, we will try to provide a brief overview of the terminology of machine learning.

There are three main types of machine learning problems to be introduced [3]:

- *Supervised learning*: Let us have N training examples described as $\mathcal{D} = \{(x_i, t_i)\}_{i=1}^{N}$, where x_i stands for data (e.g., picture pixels of digits to be recognized) and t_i for labels corresponding to the x_i (0, 1, 2, . . . , 9). The goal of supervised learning is to assign a label t to input data x based on the training set $\mathcal{D}$. Following the digit recognition example, we train a machine with large number of example digits x_i and correct labels t_i in $\mathcal{D}$ and then we want the machine to output label t to a new picture input x [2].

  Another formulation is to find parameters θ of model $g(\cdot)$ that describes a mapping $t = g(x|\theta)$ [1].

  Classification are called problems with discrete values of t, and this covers applications such as character recognition, speech recognition, face recognition. *Discriminant* is called a function that separates different classes. *Regression* term is used for case of continuous values of t. A basic example of regression is e.g., fitting a polynomial function with noisy observations, as seen in the following section.

- *Unsupervised learning*: Let us have N training examples, but this time we are not provided with the labels, i.e. $\mathcal{D} = \{(x_i)\}_{i=1}^{N}$. Supervised learning aims to reveal some information about the data and generally to discover the mechanism of the data set generation. Typical problems are *clustering* (grouping of similar examples x_i), *density estimation* (determining distribution of x_i) and *dimensionality reduction* [2].

- *Reinforcement learning*: Consider a task, which consists of taking many steps, and based on the steps performed, either rewards or punishments are allocated. The individual steps are not important, but the important thing is to reach the goal based on a correct policy [3]. This policy is good if the steps based on this policy are good to reach the goal and should be found based on the past good steps [1]. The difference from supervised learning is that the labels t are not provided with respect to individual steps; however, some information is provided afterward, and therefore it is not completely unsupervised. The solution is complicated by the fact that an inter-step leading to local maximization of reward might not lead to the overall best policy [2].

Next common terms to be introduced are:

- *Feature extraction* is a term used to address a preprocessing of the data to be used by the machine and includes steps to improve the ability of the machine to determine the correct function.
- Tasks of *predictions* are understood, such that in case of discovering some hypotheses based on the past training data, we can decide, if the current conditions of new data are consistent and eventually predict, that state of the system will correspond to the learned model.
- *Outlier detection* are tasks, when we are afraid, that the given data set might be corrupted by samples, that are not consistent with the remaining samples and could damage further analysis, if not excluded.

2.3.1 Introductory Example: Polynomial Curve Fitting

Let us now describe a simple example of polynomial curve fitting, where the approach of machine learning will be shown practically. The example and discussion are based on [2], and the computation was implemented in Matlab. Note, the approach is based on a well-known least-squares method. However, we might consider it to be a good tutorial example. It is supposed to serve for the introduction of term *over-fitting* and to repeat the basic overview of the machine learning terminology.

Training data $\mathbf{t}_{\text{training}} = (t_1, \ldots, t_N), N = 10$, were obtained as noisy observations of function $f(x) = \sin(2\pi x)$. The samples are shown in Figure 2.1 together with function $f(x)$. The task is to solely based on the training set $\mathbf{t}_{\text{training}}$ to estimate the *unknown* function $\hat{t} = f(x)$ to assign for any value of x a predicted value of $\hat{t}$. Here, as an introductory example, it is our convenience to know the model of the situation, which could be described as $t = \sin(2\pi x) + n$, where n is noise term drawn from Normal distribution with zero mean and some variance σ^2, denoted $n \sim \mathcal{N}(0, \sigma^2)$. As stated earlier, in machine learning tasks we are asked to investigate the training data in order to avoid investigating of the specific observation model.

In this example, we shall use the polynomial approximation of form

$$\hat{t}(x, \mathbf{w}) = w_0 + w_1 x + \cdots + w_m x^m = \sum_{i=0}^{m} w_i x^i, \tag{2.1}$$

where $\mathbf{w} = (w_0, \ldots, w_m)$ are coefficients, that uniquely identify a polynomial of order m, fitting the training data $\mathbf{t}_{\text{training}}$. A performance metric to be used shall be *square-error* function evaluated in N points and defined as

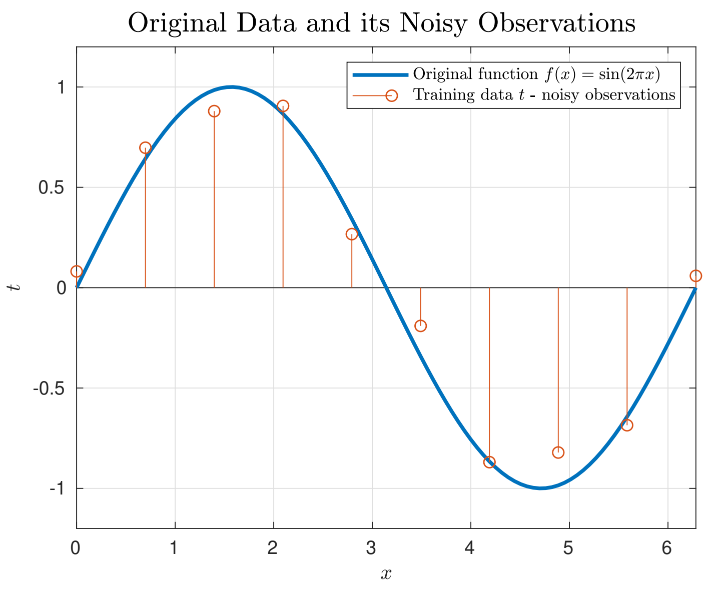


Figure 2.1: Figure of training data, $\mathbf{t}_{\text{training}}$, and original function $f(x) = \sin(2\pi x)$.

$$E(x, \mathbf{w}) = \sum_{n=1}^{N} [\hat{t}(x, \mathbf{w}) - t_n]^2. \tag{2.2}$$

Note that such a definition is opted because the metric is to be minimized using differentiation, and taking the square ensures the first derivative to be continuous, which eases the minimization process. Further, observe that function $E(x, \mathbf{w})$ is linear with respect to the coefficients $\mathbf{w}$. Next step in *model selection* is the choice of polynomial order m. Then, we evaluate the function in training points $\mathbf{t}_{\text{training}}$, and, by standard differentiating and comparing the results with zero, and we minimize the result, i.e., we obtain a set of *linear* equations in the form

$$\frac{\partial E(x, \mathbf{w})}{\partial w_0} = 0, \ldots, \frac{\partial E(x, \mathbf{w})}{\partial w_m} = 0, \tag{2.3}$$

and solve the system. We solved the set of Equations 2.3 for $m = 1, \ldots, 9$ and the resulting approximations can be seen in Figure 2.2 for selected values of $m = \{1; 3; 5; 9\}$. Visually, we can see that the obtained linear function for $m = 1$ gives a very poor result in fitting the training set, $\mathbf{t}_{\text{training}}$. This is not surprising, given the background knowledge of sinusoidal model. A situation, when the model in not capable to follow the data is called *under-fitting*. Next the polynomials of order $m = 3$ and $m = 5$ have quite a good ability to approximate the $\sin(2\pi x)$ and the polynomial of order $m = 9$ can, of course, go through all the ten points of the training set, $\mathbf{t}_{\text{training}}$.

To evaluate the *generalization* ability of our approximation, let us define *root-mean-square error* E_{RMS} as

$$E_{\text{RMS}} = \sqrt{\frac{E(x, \mathbf{w})}{N}}, \tag{2.4}$$

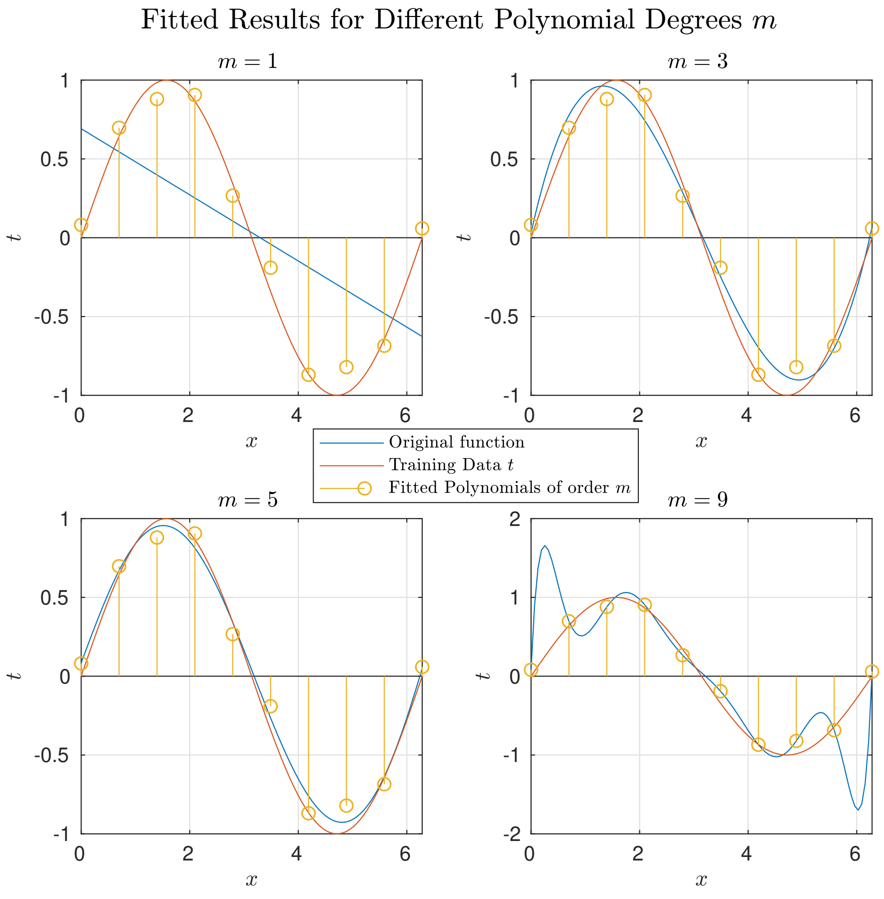


Figure 2.2: Figures of polynomial curves received after minimization of error function; given are examples of the first (i.e., a linear function), the third, the fifth and the ninth order polynomials, respectively. The ninth order polynomial perfectly approximates training data but overall gives poor performance, because it has only a limited ability to assign an appropriate value of $\hat{t}$ to new values of x outside the training set.

which has an ability to "fair and square" evaluate the error without dependence on the size of the test data set. The set, $\mathbf{t}_{\text{test}}$, consists of 100 samples that were generated in the same way as $\mathbf{t}_{\text{training}}$ and are shown in Figure 2.3. Note, we assume in this example, that $\mathbf{t}_{\text{test}}$ was not available before.

Let us evaluate how good the approximation is. The resulting plot of errors is shown in Figure 2.4. We can see, that evaluating the error function E_{RMS} with respect to training set, $\mathbf{t}_{\text{training}}$, the function E_{RMS} is for increasing value of m monotonically decreasing and this confirms the observation from Figure 2.2, where the error is 0 for $m = 9$. On the other side, evaluating E_{RMS} with respect to test set, $\mathbf{t}_{\text{test}}$, shows a decrease for $m = 1, \ldots, 8$ and then a steep increase for $m = 9$, which can be easily understood reviewing the Figure 2.2. The situation, when the model is over-trained on the training data and then gives poor performance on test data is called *over-fitting* and should be avoided by a proper *model selection* - in this case, it means not to select a value of m to large [2].

We can inspect, that our approach leads for increasing values of m to increasing values of coefficients w_i. Let us now concentrate on the case of $m = 9$. We can try to train a machine, that will prefer smaller coefficients w_i.

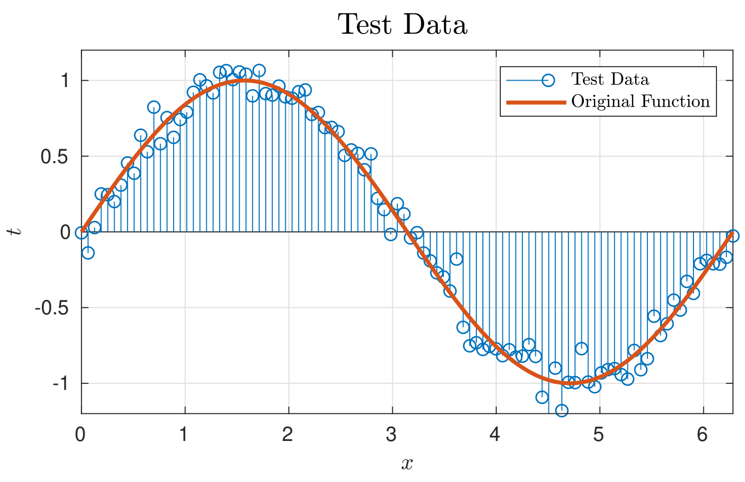


Figure 2.3: A hundred samples of test data to check the ability of generalization; the samples were obtained by the same process as training data in Figure 2.1. It is a common practice to evaluate the resulting performance of the machine using a data set, which was not used during the training phase.

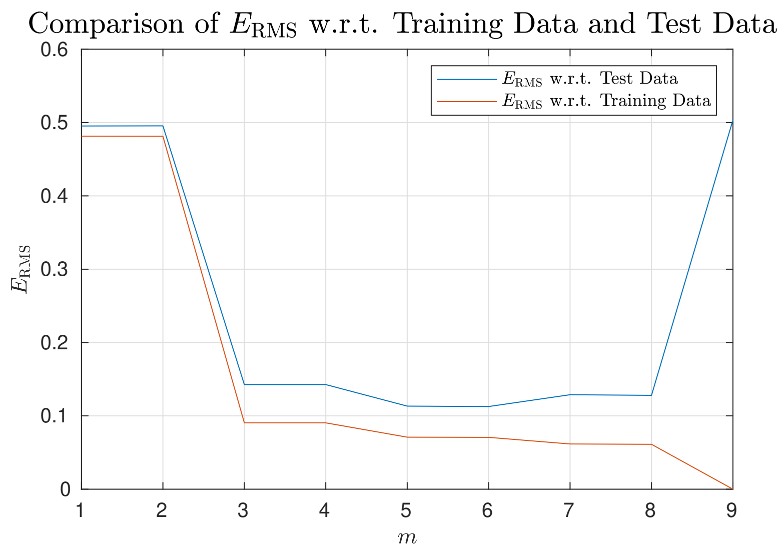


Figure 2.4: Figure compares calculated E_{RMS} error with respect to training data and test data sets; with polynomial of the ninth order the machine learns to perfectly fit the training data but gives very poor result to fit the test data.

This is motivated by an intention to reduce large oscilations of the resulting polynomial, as seen in Figure 2.2 in case of $m = 9$. To do so, let us change the performance metric to take form of [2]

$$\tilde{E}(x, \mathbf{w}) = \sum_{n=1}^{N} [\hat{t}(x, \mathbf{w}) - t_n]^2 + \lambda \sum_{i=0}^{9} w_i^2. \tag{2.5}$$

When minimizing the metric in Equation 2.5, we penalize larger magnitude of coefficients w_i. This form of *regularization* is used to train the machine with smaller coefficients w_i, which leads to less rapid changes in the values of the resulting polynomials.

Again, this approach was implemented in Matlab and the results are provided in Figure 2.5 for different values of λ. For the selected values of λ, we can observe that the rate of changes is decreased as $\ln \lambda$ increases for

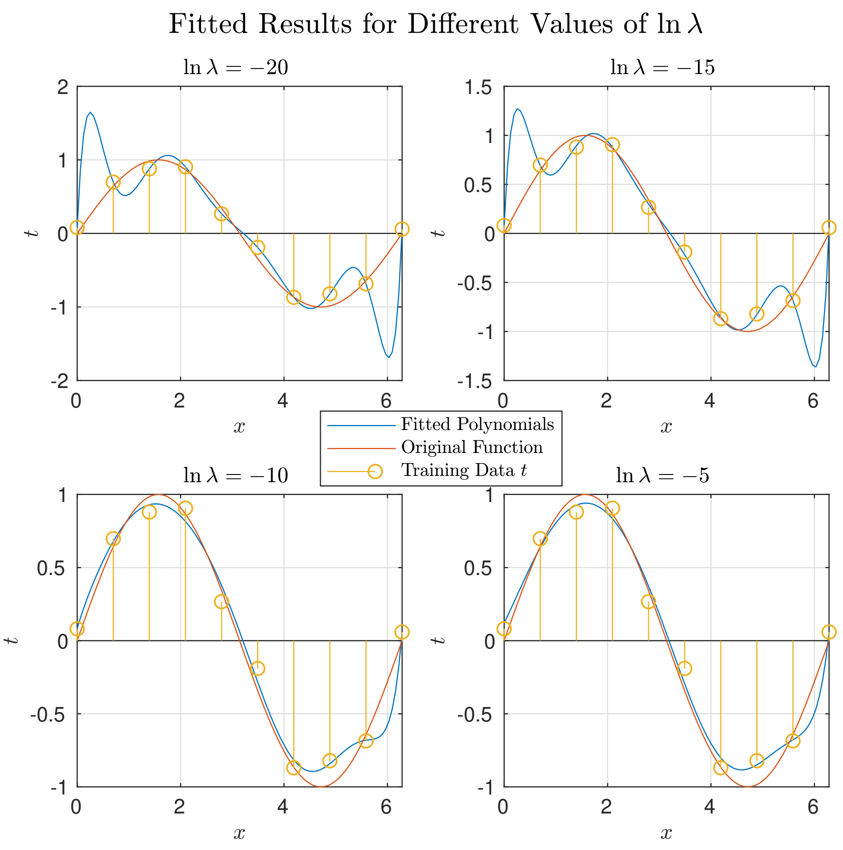


Figure 2.5: We can see how choice of regression parameter λ affects the resulting approximation. Small values of $\ln\lambda = \{-20; -15\}$ do not much suppress the oscillations of the ninth order polynomial; the exemplary values of $\ln\lambda = \{-10; -5\}$ show that the problem of malicious oscillations, i.e. over-fitting, is suppressed. Notice, that with the regularization involved, the approximation polynomial curve does not go through all the training samples any more.

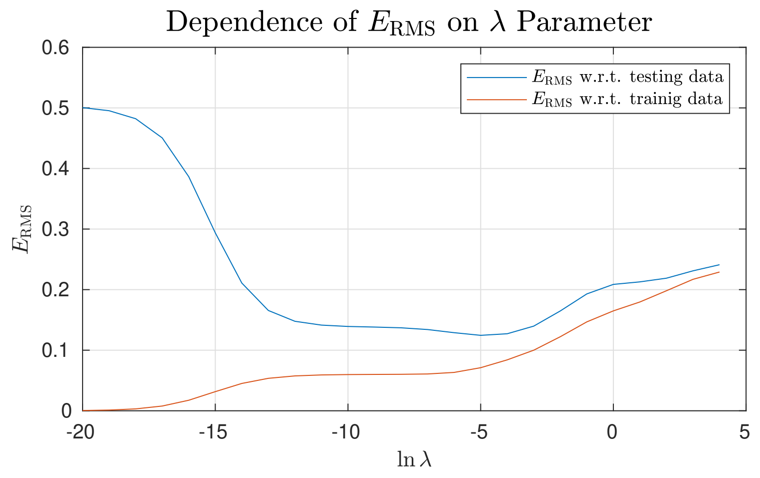


Figure 2.6: Figure analogical to Figure 2.4. We can compare values of E_{RMS} calculated from the sets of training data $\mathbf{t}_{\mathrm{training}}$ and test data $\mathbf{t}_{\mathrm{test}}$. As $\ln\lambda$ increases, the obtained approximation is less successful in matching the training data $\mathbf{t}_{\mathrm{training}}$ and, at the same time, error with respect to test data $\mathbf{t}_{\mathrm{test}}$ is reduced. Nevertheless, for $\ln\lambda > -5$, the obtained polynomial loses ability to follow original function, because the coefficients are forced to be too small, see Figure 2.7.

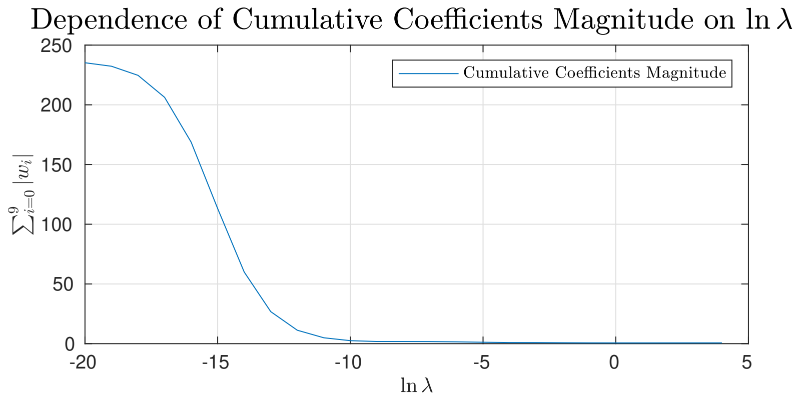


Figure 2.7: In this figure we can observe how the choice of parameter λ in metric defined in Equation 2.5 affects the resulting polynomials: increasing value of λ in the metric causes as result of minimization a significant decrease of the polynomial coefficients $w_0, \dots, w_9$.

$m = 9$. In Figure 2.6, we can again compare behaviour of E_{RMS} with respect to $\mathbf{t}_{\text{training}}$ and $\mathbf{t}_{\text{test}}$, respectively. While the regularization causes increasing deviation from the training data $\mathbf{t}_{\text{training}}$, it reduces an error evaluated with respect to test data $\mathbf{t}_{\text{test}}$.

We can roughly observe that optimum value of λ is for $\ln \lambda \in (-10; -5)$, and then the suppression of magnitude of the coefficients is too high; see Figure 2.6. We can also compare the results from Figure 2.6 and Figure 2.4 to see, that by applying the regularization process, we managed to obtain similar results as when selecting smaller polynomial order.

Finally, let us have a look at Figure 2.7, where we can see how our regularization affected the size of the approximation polynomials. We can observe that the magnitude of the coefficients decreases very rapidly with an increasing value of $\ln \lambda$.

Summarizing this subsection, we have seen a trivial polynomial curve fitting example based on [2] and, e.g., seen also in [4], where a problem of polynomial fitting was solved with a very traditional approach of least squares minimization. The desired outcome is to provide a basic example, where the terminology was revisited, as well as to introduce the problematics of model selection related to under-fitting and over-fitting issues.

2.4 Current Trends of Machine Learning in Communications

Let us now provide an overview of machine learning approaches that are according to the recent literature perspective in the context of wireless communication systems. Specifically, we shall be interested in the methods suitable for applications on the physical layer. As discussed in the Introduction section above, we shall try to concentrate on methods of machine learning, which might be useful in digital communication problems in situations, when alternative solutions are not fully satisfactory. Previously in the text, we provided notes about evaluating the suitability of machine learning approaches for given tasks. Reviewing that, either algorithmic or model deficit is typical to be solved, and the situation should be considered to choose proper solution

[4]. Next, let us address few current trends, separately for the supervised, unsupervised, and reinforcement learning, and also to mention few published applications of machine learning to WPLNC.

2.4.1 Supervised Learning in Wireless Systems

Recently, machine learning approach was adopted to many problems in wireless communication. At the receiver site, the tasks of channel detection and decoding can be described as classification of the label of the transmitted message from baseband signal [4]. Methods of machine learning were adopted, e.g., in cases where no channel models exist or existing solutions are computationally very complex [4]. An issue that might occur in the case of channel decoding is the fast rate of changes in the channel state [4].

As an algorithmic deficit situation might be considered a broad task of modulation classifications [4]. Recent examples of deep learning solutions are [5] and [6], where the authors developed a system to extract image data from received I/Q parts of the signal and then applied pattern recognition techniques to train modulation recognition systems.

Neural networks are also widely used for enhancement of indoor positioning, e.g. in [7] and [8]; other recent application is to cancel self-interference of full-duplex nodes [9] [4]. Caching of popular data content is used on the application layer, and it is supported by machine learning to decide, what content might be required by users in specific locality [4].

Note, that recently amount of survey literature appeared, e.g. [10], [11], and recent trends are captured at [12].

2.4.2 Unsupervised Learning in Wireless Systems

Recently, auto-encoders attracted attention, see e.g. [13] and [14]. Traditionally, auto-encoder consists of two parts: (1) encoder processes input data to some other representation, (2) decoder operates with an output of the encoder to retrieve original input data [4]. In [13], the authors presented an idea to model the whole wireless communication chain as auto-encoder with motivation to simultaneously optimize the design of all parts of the system, i.e., modulator, source coding, channel coding, demodulator, and decoder. From the point of view of machine learning, deficient knowledge of the channel model was addressed in [14]. The auto-encoders are designed as deep neural networks.

Next, clustering is in communication systems used to allocate resources or to initially group the nodes and then perform, e.g., routing with respect to created clusters, rather than single nodes [4]. The ability of self-organization using clustering seems to be a very natural tool to be used in ad-hoc networks.

2.4.3 Reinforcement Learning in Wireless Systems

Interpretation of communication using reinforcement learning is another recent machine learning topic in wireless communication. For example, in [15], the authors illustratively present, how traditional parts of communication chain can be represented by machine learning tools (such as FIR filter $\leftrightarrow$ convolutional neural network layer, IIR filter $\leftrightarrow$ recurrent cell in artificial neural network, and source coding $\leftrightarrow$ auto-encoder). The main contribution of [15] is the design and training of the system, where modulation is learned between two nodes in a decentralized manner using reinforcement learning problem formulation.

2.4.4 Machine Learning in WPLNC

In [16], the author addressed a problem of detecting parameters of WPLNC network, namely the number of source nodes connected to relays, followed by identification of source nodes connected to the relays using the recognition of random sequences. It is called *Cloud Initialisation Procedure*, and with a focus on saving-resources approach, it is based on unsupervised k-means algorithm, pointing to possibilities of enhancement, using e.g., E-M algorithm [16]. Both, simulation and hardware implementations are provided.

In preprint [17], a deep learning approach was adopted to optimize the constellations of 2WRC. The authors describe communication in 2WRC as a combination of three deep neural networks (DNN): (1) source modulator, (2) relay node, and (3) demodulator. This setting is trained to minimize cross-entropy loss between the source and destination nodes, and simulation results are provided for different SNR values [17]. (Note, this seems to be analogical to approach of auto-encoder.)

In paper [18], the authors describe solution called WPLNC random access. It utilizes principles of WPLNC to allow a non-guaranteed exchange of messages in the environment, where interference intentionally occurs, and successful communication is reached when the destination node is able to decode (in two phases) desired messages from received set of equations on given finite field [18]. A deep neural network is in [18] designed to make decisions about sets of equations to be solved in the given step, in order to reach minimal error.

2.5 List of Machine Learning Approaches

In the literature, e.g. [1] [3], these are common general topics of machine learning:

- Bayesian Decision Theory
- Parametric Methods
- Dimensionality Reduction
- clustering
- Decision Trees
- Multilayer Perceptrons

Chapter 3

Brief Fundamentals of Wireless Physical Layer Network Coding

In this chapter, let us briefly provide an introduction to the topic of Wireless Physical Layer Network Coding (WPLNC). We will focus on comparison with traditional approaches. This chapter is based on a very recent book [19] and introductory parts of recent dissertations [20] and [16]. The goal is to provide an introduction in such a way that we can later possibly identify the target applications of machine learning algorithms in WPLNC systems.

3.1 Introduction

Let us first summarize a few generally known facts. In the past 20 years, wireless communication systems became ubiquitous and worldwide penetrated to everyday life. A fundamental restriction factor of wireless communication is in the limited amount of frequency spectrum, where the transmitted signals are conveniently separated. This separation allows the official authorities to control the frequency allocations used for many purposes by different types of organizations or individuals. The necessity of such control comes from the nature of wireless communication. Since electromagnetic waves radiated by antennas of different transmitters operating in the same frequency bands superpose on the receiver antennas, this causes a phenomenon called *interference*. Possibly with the exception of radio jamming, traditionally such interference is considered to be malicious because it damages the desired shape of the waveform at the receiver side and effectively reduces signal-to-interference-plus-noise ratio (SNIR). Informally, reduced SNIR causes the wireless communication to be vulnerable to errors and outages.

In the concept of wireless communication networks and multi-user systems, this fact gave rise to traditional methods of avoiding interference by separating the signals with different frequency channels, i.e., frequency division duplex (FDD); or with different time slots, i.e., time division duplex (TDD); or coding schemes. Corresponding to these exist the well known multiple access methods, such as TDMA, FDMA, and CDMA. Note that combinations of these schemes are commonly used. In cellular systems, these multiple access methods are efficiently utilized to allow communication of many users, while reusing the radio resources in a non-overlapping manner to avoid interference. A trend reaching its limit is reducing the size of the cells [19]. Recently, we might also observe an effort to use higher frequency bands, e.g., as in proposed mmWave communication [21]. An extension of usable communication spectrum is clearly favorable, however, new drawbacks appear at these high frequencies, such as high signal attenuation, which causes a small coverage area [21].

WPLNC paradigm offers a possibility to design a system where the intended cooperative interference of wireless network users is used to increase network capacity and, hence, use it constructively [19].

3.2 Network Coding

Traditional multi-hop networks route the messages such that intermediate devices duplicate input messages to the outputs in order to send them to the desired destinations. A concept of *network coding* was introduced with a key advance, where the intermediate device is allowed to perform operations with input messages, such that some *function* of inputs is used as the output [16].

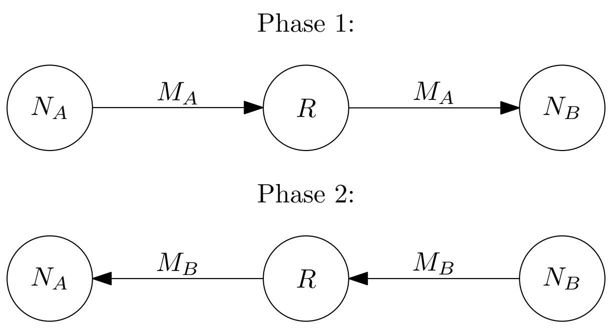


Figure 3.1: Conventional relay scheme.

Let us present a simple example shown in [19] and [20] to exemplify the network coding using Figures 3.1 and 3.2. Note, the network is referred to as a 2-way relay channel (2WRC) [19]. A description follows: Nodes N_A and N_B wish to exchange binary messages M_A and M_B via an intermediate node R, referred to as relay. The first obvious solution is to split the process into two phases, as shown in Figure 3.1. In Phase 1, N_A sends message M_A to relay R; then relay R sends the message to node N_B. In Phase 2, the same is done for message M_B originating in N_B, and the task is done, the nodes exchanged the messages.

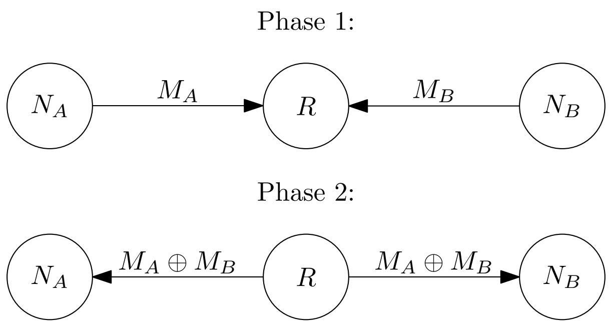


Figure 3.2: Network coding relay scheme.

Now, we revisit the same task, but demonstratively utilizing approach of network coding. In Phase 1 in Figure 3.2, messages M_A and M_B are sent to relay R. Then, node R internally performs bitwise XOR operation of the received messages, denoted $M_A \oplus M_B$, and sends the result to both nodes,

N_A and N_B. Since N_A and N_B have the knowledge of their original messages, they can process calculations to obtain the unknown message. Specifically, node N_A performs XOR again to obtain M_B, since

$$(M_A \oplus M_B) \oplus M_A = (M_A \oplus M_A) \oplus M_B = 0 \oplus M_B = M_B, \tag{3.1}$$

vice versa for N_B to obtain M_A, and a bit elaborately the task of messages exchange is completed, too.

As seen in [16], a general description of network coding, expressed as output function of nth node N_n with input messages $(M_A, M_B, \dots)$, might be stated as

$$y_{N_n} = f_{N_n}(M_A, M_B, \dots), \tag{3.2}$$

and at the destination node N_D, a reconstruction function, denoted e.g. $y_{N_D}^{-1}$, should be defined to recover the messages $(M_A, M_B, \dots)$ from possibly multiple inputs $(y_{N_1}, \dots)$, written as

$$(M_A, M_B, \dots) = f_{N_D}^{-1}(y_{N_1}, \dots). \tag{3.3}$$

Before moving to WPLNC, let us explicitly state, that in the network coding paradigm, the forwarded messages are logically computed in the corresponding devices according to predetermined functions, as in Equation 3.2, and that this high-level approach is used for both, wired and wireless networks, when appropriate multiple-access schemes are applied [19].

3.3 Wireless Physical Layer Network Coding

Concept of WPLNC might be understood as a modification of network coding, tailored to the usage in the wireless environment, where the physical properties of wireless channels are considered and utilized. Firstly, *half-duplex constraint* is a common assumption in wireless communication that must be respected, meaning that the network nodes are not capable of transmitting and receiving signals at the same time using identical radio resources [19]. In the paradigm of WPLNC, the devices are aware of the network topology, and the operations of the network, such as routing, are adapted to it, which consequently improves the theoretic capabilities of the network, compared to the traditional approach of point-to-point links in networks [19]. Knowledge of the topology can be then exploited to transmit signals that are on the physical layer intentionally combined and received in predetermined combinations [19].

The fundamental difference between network coding and WPLNC is the way of combining signals. Previously in case of network coding, relay performed a calculation of specific function based on the inputs and transmitted a result; in WPLNC, the combination of signals reaching the relay originates as a superposition of the electromagnetic waves, i.e., interference at the relay receiver antenna [19] [16].

Revisiting the previous example of 2WRC, we will introduce the most fundamental terms in WPLNC terminology. In the example, we consider that nodes N_A and N_B wish to exchange single bits b_A and b_B, modulated using uncoded BPSK transmission. The half-duplex constraint results in time-domain separation of the process into two phases (see Figure 3.3):

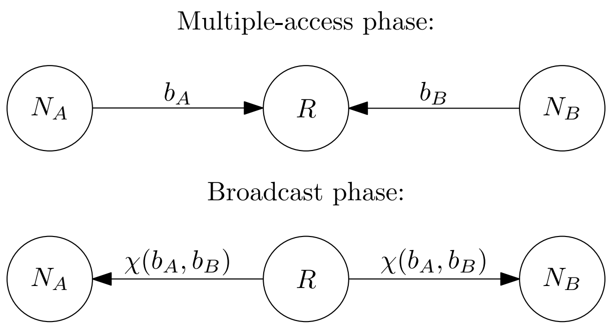


Figure 3.3: WPLNC example: 2-way relay channel. Multiple-access phase and broadcast phase are illustrated.

- *Multiple-access phase* covers the synchronized transmission of the modulated signals received in superposition at the receiver antenna of relay node R;
- *Broadcast phase* refers to (further specified) retransmission of the received superposed information according to mapping $\chi(b_A, b_B)$ [19].

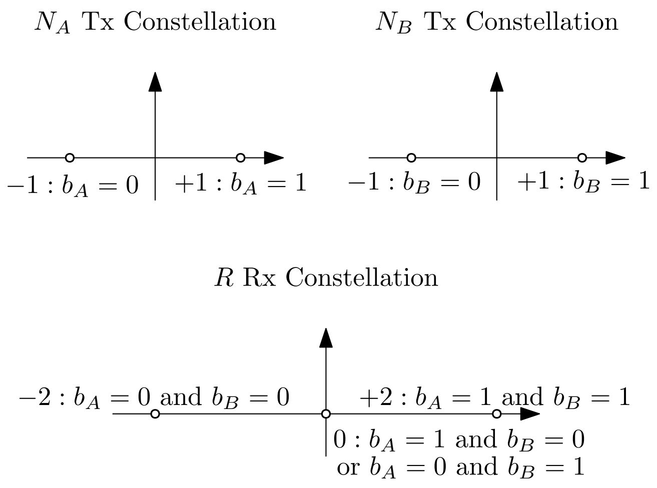


Figure 3.4: BPSK constellations in WPLNC setup describing multiple-access phase. Top: standard BPSK constellations transmitted by nodes N_A and N_B; Bottom: superposed constellation received at relay R, neglecting channel effects and assuming synchronization.

Let us assume a unit gain of the the wireless channels between all nodes. In Figure 3.4, we can see an illustration of standard BPSK constellations transmitted by nodes N_A and N_B, and constellation received by relay R. Note, relay R does not observe mappings of individual bits b_A and b_B, but their *network coded* signals, represented e.g. in the constellation space. Considering the superposed constellation points, if bit value 0 is by BPSK represented as -1 and bit 1 as $+1$, simultaneous transmissions result clearly in 3 possible results: $(b_A = 0, b_B = 0) \to -2$, $(b_A = 1, b_B = 1) \to 2$, and $(b_A = 0, b_B = 1)$ or $(b_A = 1, b_B = 0) \to 0$. Graphically this is illustrated in bottom of Figure 3.4. Note, this superposed modulation is predetermined by the system design. A many-to-one function of the input data resulting to specific value at the relay is called *hierachical network code map* (HNC map). Thus the relay does not access the specific data bits b_A, b_B, but only needs to distinguish between the values $s(b_A, b_B) \in \{-2; 0; +2\}$. Based on values $s(b_A, b_B)$, values of $b = \{0; 1\}$ are assigned as XOR of original data bits b_A, b_B.

Variable b is referred to as *hierachical symbol*. When hierarchical symbols are received by nodes N_A, N_B, these need to utilize previous knowledge of transmitted symbols, i.e. b_A value for N_A and b_B for N_B, in order to be able to decode the required bit values. This background knowledge necessary for recovery of the information is called *hierachical side information* (HSI).

Assuming wireless transfer and considering half-duplex constraint, we can compare number of time slots required to perform the task of message exchange examples depicted in Figures 3.1, 3.2 and 3.3 to demonstrate the efficiency of WPLNC scheme. Conventional approach requires overall 4 time slots for transfers (1: $N_A \rightarrow R$, 2: $R \rightarrow N_B$, 3: $N_B \rightarrow R$, 4: $R \rightarrow N_A$); Network coded solution requires 3 time slots (1: $N_A \rightarrow R$, 2: $N_B \rightarrow R$, 3: $R \rightarrow N_A$ and N_B) and, finally, WPLNC scheme can perform this task using only 2 time slots (1: N_A and $N_B \rightarrow R$,2: $R \rightarrow N_A$ and N_B), which is a remarkable result.

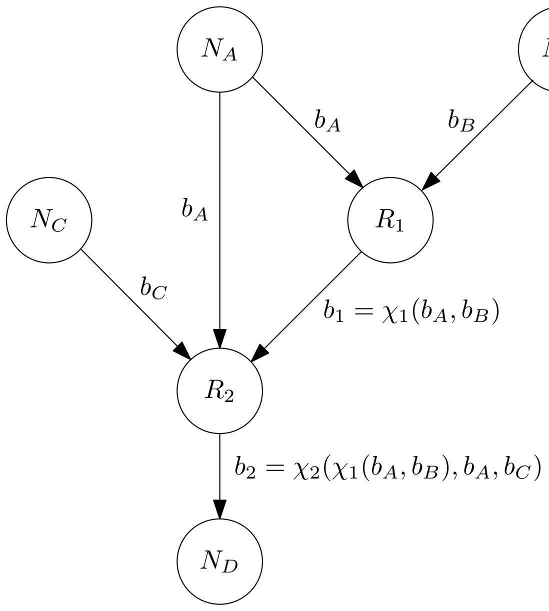


Figure 3.5: WPLNC hierarchical structure demonstration. Considering more complicated network topology, inputs of the HNC map producing hierarchical symbols can be also other hierarchical symbols.

Note, that in the terms above, the attribute "hierarchical" is more illustrative, when the topology is more complicated and refers to, so-called, *hierarchical principle*, which is essential for WPLNC. Hierarchical principle corresponds to a fact that in general network the relays perform the following tasks: (1) receive information from multiple network predecessors (front-end processing), (2) process this information to forward it (relay stage) and (3) forward some function of the received information (back-end processing) [19]. These three steps, considered in multiple stages create *hierarchical encapsulation* [19]. Let us see Figure 3.5 for better understanding. In there, nodes N_A and N_B produce symbols b_A and b_B; these are delivered to relay R_1, which applies its specific HNC map χ_1 and produces hierarchical symbol $b_1 = \chi_1(b_A, b_B)$. So far, this is identical to the 2WRC. Next, node N_C produces symbols b_C and it reaches relay R_2 together with hierarchical symbol b_1. Relay R_2 applies its HNC map χ_2 and produces hierarchical symbol $b_2 = \chi_2(\chi_1(b_A, b_B), b_A, b_C)$, understood as composition. Finally, destination node N_D receives hierarchical symbol b_2 and uses it to recover original information symbols b_A, b_B, b_C. Informally, a mapping used at the destination node to retrieve desired information from all received hierarchical symbols, is referred to as *global HNC map* [19].

Finally, let us state that the fundamental requirement laid on the WPLNC system is to enable end-to-end communication, meaning that all required input messages can be in destination nodes unambiguously decoded using appropriate global HNC maps. This is formally addressed by *generalized exclusive law* in [19] and informally states, that all different sets of input symbols transmitted by source nodes need to be processed by the hierarchical network, such that global HNC map decodes the received hierarchical symbols uniquely.

As described so far, the WPLNC concept offers the possibility to improve the capacity of relay networks with multiple information flows at the cost of tailored design respecting the topology [19]. Hereby, we have tried to provide a brief introduction to the topic of WPLNC, and for comprehensive and formal description, we refer to [19].

Chapter 4

Artificial Neural Networks

In recent literature on usage of machine learning in communications, e.g. [4] [13] [18], artificial neural networks (ANN) are very often utilized for different purposes. In this chapter, we shall discuss an introduction to the ANN topic, motivated by a wide range of possible applications in communication systems. In a tutorial way, we shall train a simple ANN system and focus on the training description. The following is based on [1] and [22]. According to [22], characteristics of the typical use-cases of ANN are:

- Target function is defined with training data set consisting of a vector of features $\boldsymbol{x}$ and desired output t;
- Training data can contain errors (noise);
- Long training times are acceptable;
- Fast evaluation using a trained machine is possible;
- Black-box decision is made with (often) difficult interpretability.

4.1 Introduction

The basic idea of ANN is that a weighted combination of input values is thresholded using a non-linear function to provide the desired output. Basic unit is called *perceptron* and is shown in Figure 4.1.

Inputs $x_i \in \mathbb{R}, i = 0, 1, \ldots, n$, might come either from the environment or can be outputs of other perceptrons; weights $w_i \in \mathbb{R}, i = 0, 1, \ldots, n$, are called *synaptic weights*, associated to the inputs x_i. x_0 is called *bias unit* and is always equal to one [1].

To begin with a purely linear system, we can write output y as

$$y = \sum_{i=0}^{n} x_i w_i, \tag{4.1}$$

or equivalently as a dot product

$$y = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}, \tag{4.2}$$

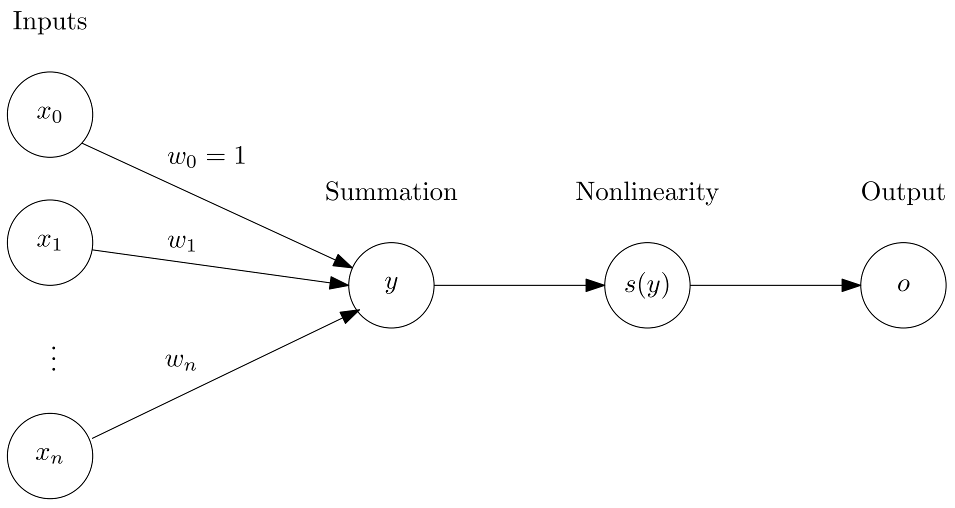


Figure 4.1: Single ANN perceptron: Inputs $x_0, x_1, \ldots, x_n$ are with appropriate weights $w_0, w_1, \ldots, w_n$ summed up to variable y, which is thresholded using non-linear function $s()$ to output o.

where $\boldsymbol{x} = (1, x_1, \ldots, x_n)^{\mathrm{T}}$ and $\boldsymbol{w} = (w_0, w_1, \ldots, w_n)^{\mathrm{T}}$.

In this setup, given data $\boldsymbol{x}$, values of weights $\boldsymbol{w}$ should be found to perform desired task. For $d = 1$, the model reduces to

$$y = x_1 w_1 + w_0, \tag{4.3}$$

that is capable to perform linear fit with slope w_1 and intercept w_0 (see Eq. 2.1). Generally, for $n > 1$ it defines plane in n-dimensional space, which can be used to linearly separate two classes, C_1 and C_2 for positive and negative values, respectively [1]. Formulating it mathematically, we use *threshold function*, $s(y)$, to make the decision. In this case, if the threshold function is defined as

$$s(y) = \begin{cases} +1 & \text{for } y > 0 \\ -1 & \text{otherways} \end{cases}, \tag{4.4}$$

then we select C_1 for $s(y) = 1$ and C_2 for $s(y) = -1$. A possible interpretation of this, is that our designed weights $\boldsymbol{w}$ allow the input values $x_1, \ldots, x_d$ to "overpressure" the bias unit x_0 .

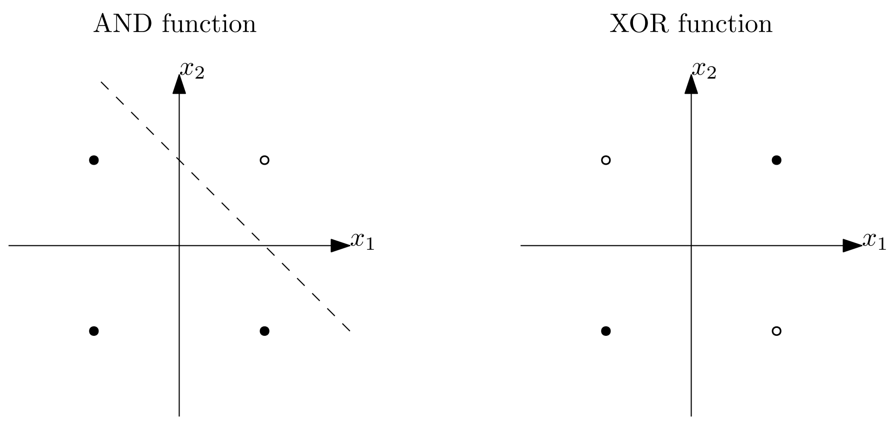


Figure 4.2: Linear separability demonstration: AND function is linearly separable; XOR is not linearly separable.

In Figure 4.2, let us demonstrate the linear separability principle on boolean functions: with x_1 and x_2 representing inputs (± 1) of boolean functions outputting $+1$ (circle) and -1 (disk), we can easily find (e.g.) weights $w_0 = 1, w_1 = -1$, defining a line (dashed), that might be understood as discriminant function realizing function x_1 AND x_2; for function XOR, however, the task is not solvable using a single perceptron, because we can not find a single line, separating the values correclty as for AND function [22].

4.2 Gradient Descent

Let us now discuss, based on [22], how to determine the weight vector $\boldsymbol{w}$. To begin with, we omit the tresholding operation and investigate first the situation for output y as shown in Eq. 4.3. The discussed approach is called *gradient descent*. For this purpose, assume that we want to train the perceptron with training set $\mathcal{D} = \{(\boldsymbol{x}_d, t_d)\}_{d=1}^{D}$.

Training error, $E(\boldsymbol{w})$, shall be measured using squared-error function

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{d=1}^{D}(t_d - y_d)^2, \tag{4.5}$$

where t_d is target training value for dth training example $\boldsymbol{x}_d$ and $y_d = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_d$ is actual output of the trained perceptron with given weights $\boldsymbol{w}$. Note, that Eq. 4.5 defines a quadratic function of weights $\boldsymbol{w}$ (determined by the training data) and we desire to find minimum of this function.

It is generally known that the gradient of function points to the direction of the steepest ascent. Therefore "minus gradient" defines the direction of steepest descent. Simply, gradient descent is the iterative method, where we are moving along the error surface defined by Eq. 4.5 in the direction against the gradient, which is here defined as

$$\nabla E(\boldsymbol{w}) = \left[\frac{\partial E(\boldsymbol{w})}{\partial w_0}, \frac{\partial E(\boldsymbol{w})}{\partial w_1}, \dots, \frac{\partial E(\boldsymbol{w})}{\partial w_n}\right]. \tag{4.6}$$

To state the training rule, we first initialize the weight vector $\boldsymbol{w}$ with small random values, e.g. uniformly distributed values from interval $\left(-\frac{1}{100}; \frac{1}{100}\right)$ [1]. The training rule can be then written as

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla E(\boldsymbol{w}), \tag{4.7}$$

where η is positive constant called *learning rate* and needs to be of appropriate size: too small value makes the learning last unnecessarily long time; too big can prevent the training process from converging [22].

To implement this training rule practically, we need to evaluate the gradient in Eq. 4.6. Because we omitted the non-linear function, the situation is similar to that in Section 2.3.1, and we can evaluate elements of the gradient analytically as

$$
\begin{aligned}
\frac{\partial E(\boldsymbol{w})}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d=1}^{D}(t_d - y_d)^2 \\
&= \frac{1}{2}\sum_{d=1}^{D}\frac{\partial}{\partial w_i}(t_d - y_d)^2 \\
&= \frac{1}{2}2\sum_{d=1}^{D}(t_d - y_d)\frac{\partial}{\partial w_i}(t_d - y_d) \\
&= \sum_{d=1}^{D}(t_d - y_d)\frac{\partial}{\partial w_i}(t_d - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}) \\
&= \sum_{d=1}^{D}(t_d - y_d)(-x_{id}),
\end{aligned}
\tag{4.8}
$$

where x_{id} is ith component of dth training example [22]. We can now express update rule for a single single weight value w_i as

$$
w_i \leftarrow w_i + \eta\sum_{d=1}^{D}(t_d - y_d)x_{id},
\tag{4.9}
$$

and because our simplified model has only one global minimum, this approach is guaranteed to converge, given sufficiently small value of learning rate η.

Gradient descent is a general technique, which can be used to search hypotheses space, parameterized by continuous variable (weights), provided that error function is differentiable with respect to these parameters [22]. General problems of this simple approach are possibly slow convergence and a fact that in case of the error function with multiple local minima, it does not guarantee to reach a global minimum [22].

4.2.1 Example: Gradient Descent Training

Let us now provide an implementation example of gradient descent training. Given training data $\mathcal{D} = \{(\boldsymbol{x}_d, t_d)\}_{d=1}^{1000}$, where $\boldsymbol{x}_d = (x_1, x_2)$ represent two-dimensional data and $t_d \in \{-1, +1\}$ defines two classes, the task is to use gradient descent to find weights $\boldsymbol{w} = (w_0, w_1, w_2)$, that perform linear separation of these classes. Schematically, it is shown in Figure 4.3.

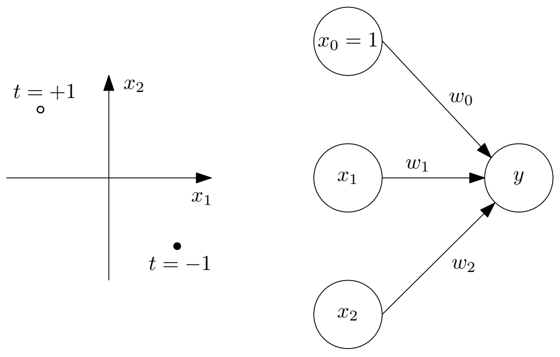


Figure 4.3: Gradient descent - example organization: two-dimensional data (x_1, x_2) are labeled with value t and perceptron is trained to separate these classes using linear discriminant function.

Equation 4.9 was used to find values w_0, w_1, w_2 and the training was performed in Matlab. Evolution of squared error function, defined in Eq. 4.5 is shown in Figure 4.4. As expected, the error decreases uniformly because, in our simplified case, there is only one global minimum. Note, the learning rate η was chosen heuristically.

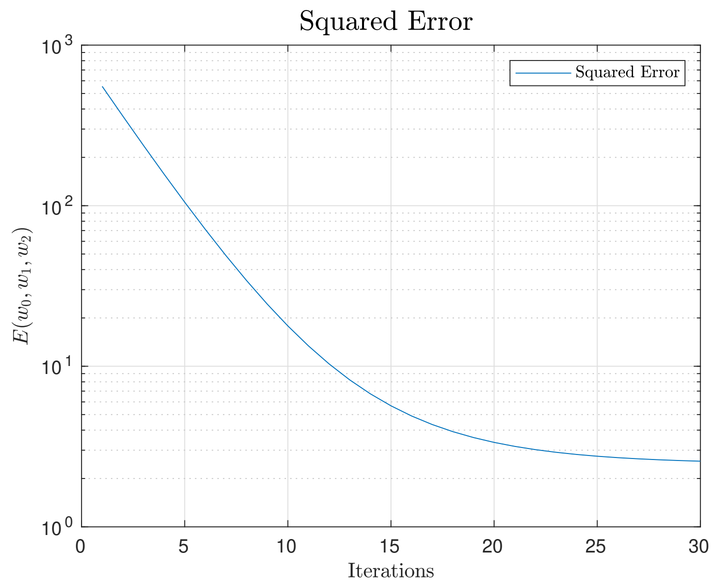


Figure 4.4: Squared error function evolution during gradient descent training: we can observe uniform convergence of the training algorithm.

In Figure 4.5, we can have a look at values of descending weights during single iterations. We can observe that the algorithm converges within 25 iterations for all weights.

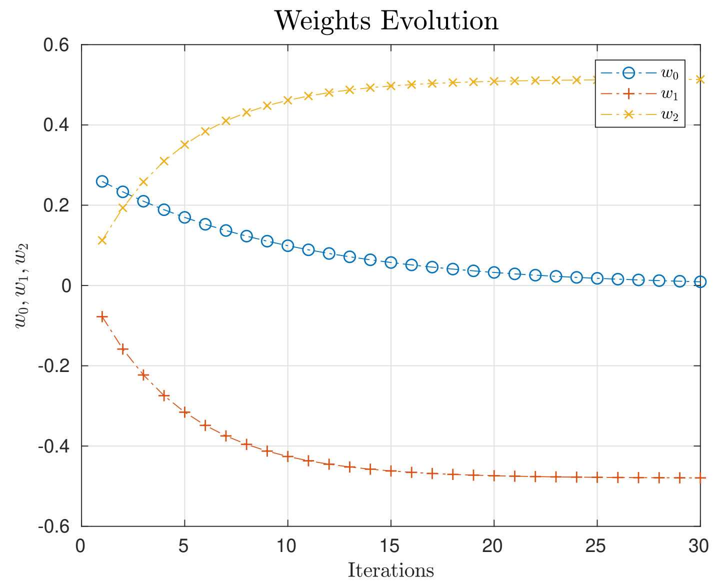


Figure 4.5: Gradient Descent: convergence of individual weights. This figure is dual to Figure 4.4, but provides in detail observation of how the individual weights contribute to the overall error.

We might also be interested in plotting a decision boundary that the system learns to perform the classification. To do so, let us have a look at result of the summation, i.e. $w_0 + w_1x_1 + w_2x_2$. We compare this sum with zero and express x_2 as function of x_1, i.e.

$$x_2 = -x_1\frac{w_1}{w_2} - \frac{w_0}{w_2}. \tag{4.10}$$

In Figure 4.6, we observe the training data of separated classes and decision boundaries. We can see that the system learned a decision line approximately perpendicular to the connection of means of the two classes.

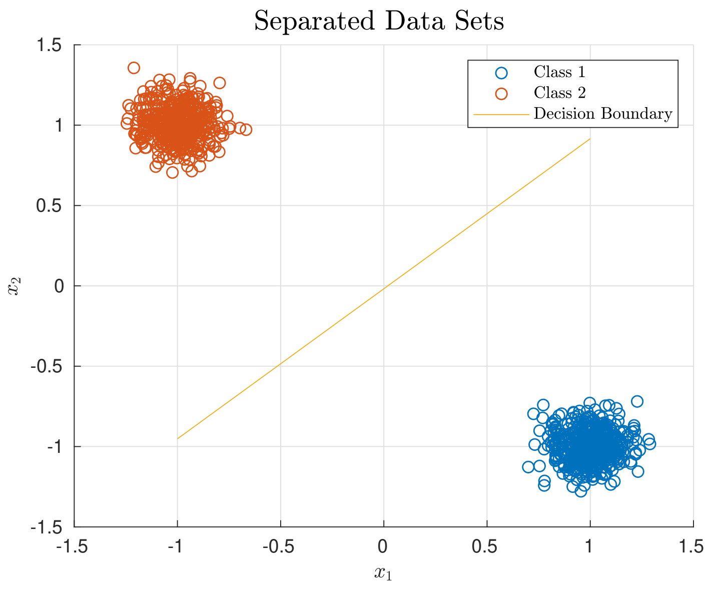


Figure 4.6: Gradient descent: separation of the classes and learned decision boundary.

Finally, in Figure 4.7, we demonstrate the robustness of the gradient descent algorithm. The data sets, in this case, are not linearly separable. Nevertheless, the training procedure still converged to an acceptable solution.

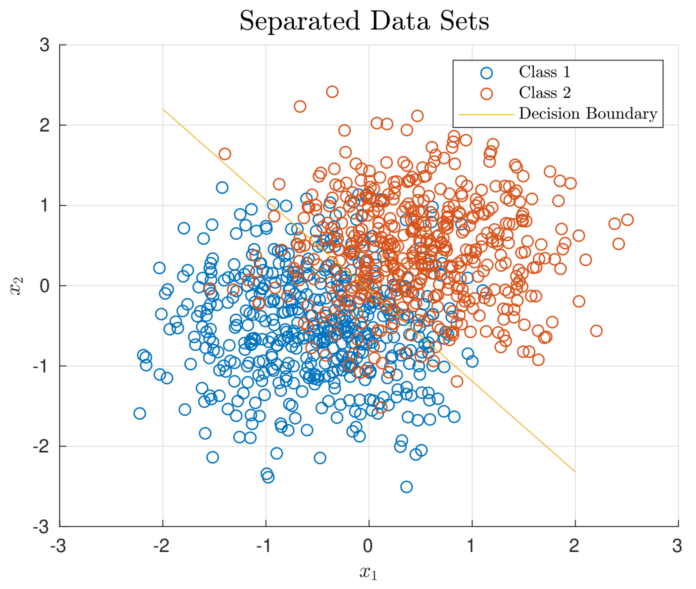


Figure 4.7: Gradient descent: learned separation of linearly non-separable classes.

4.3 Stochastic Gradient Descent

Stochastic gradient descent (often abbreviated as SGD) is modification of gradient descent algorithm, described in the section above. In the gradient descent approach, the training is performed over all training samples within one moment. This is sometimes called *batch learning*. SGD *approximates* this computation using iterative updates, based on individual training samples [22]. The modified error function is given by the expression

$$E_d(\boldsymbol{w}) = \frac{1}{2}(t_d - y_d)^2, \tag{4.11}$$

and resulting learning rule, based on minimization of Eq. 4.11 is

$$w_i \leftarrow w_i + \eta(t_d - y_d)x_{id}, \tag{4.12}$$

where the learning iterations are with respect to training data index d [22]. In every iteration of d out of D, the weights are modified according to gradient computed from single training samples. This is its main advantage over the previous solution because it helps to avoid falling to the local minimum of the minimized error function.

SGD is a very generic algorithm and is broadly used. Sometimes, it is referred to as Least Mean Square (LMS) algorithm and is used, e.g., for adaptive filter design [22].

4.3.1 Example: Stochastic Gradient Descent Training

SGD training was applied to an example described in Subsection 4.2.1, with only modification in the expansion of training set to $D = 3000$, for a purpose to intuitively demonstrate convergence properties. Reached results are very similar, and therefore identical outputs are here omitted.

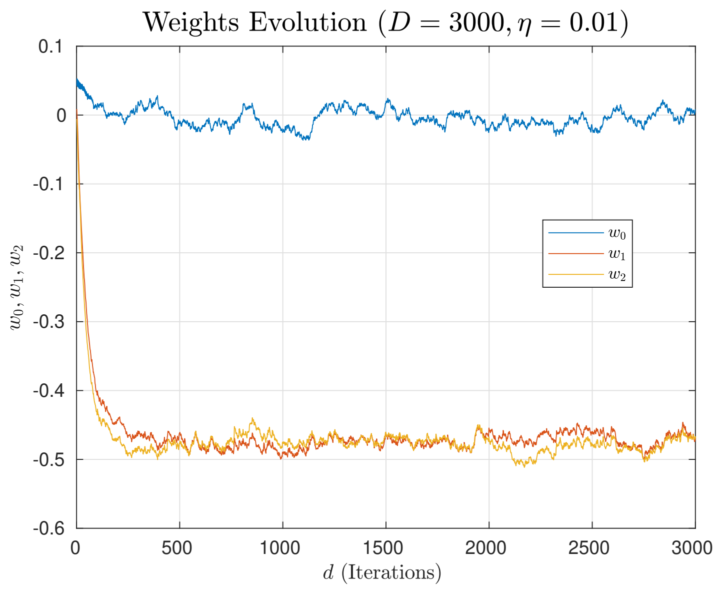


Figure 4.8: SGD training: weights evolution for $\eta = 0.01$. The convergence is disturbed by noise artefacts.

However, we shall present here the impact of the proper choice of learning parameter η. In Figure 4.8, the chosen learning parameter was $\eta = \frac{1}{100}$, and we can observe, that convergence of values of the weights is not uniform and suffers from significant noisy oscillations. In Figure 4.9, the learning rate value was decreased to $\eta = \frac{1}{1000}$, and we can observe, that the convergence process is much smoother, but also significantly slower.

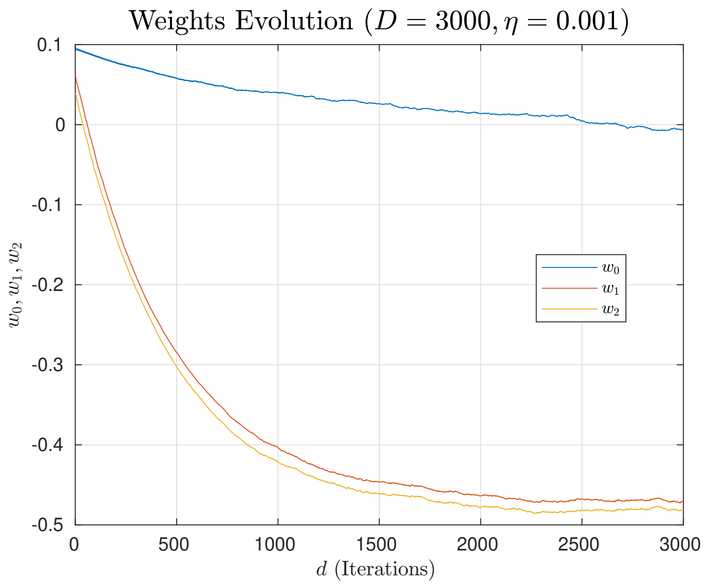


Figure 4.9: SGD training: weights evolution for $\eta = 0,001$. The noise artefacts were suppressed.

This illustrates the trade-off between accuracy and time of convergence. A common technique to smooth the training process and still keep it sufficiently fast is to decrease the learning rate of η gradually [22].

4.4 Backpropagation Algorithm

As stated above, single perceptrons are capable of describing only linearly separable situations [22]. More complicated, non-linear functions are learned by *multilayer networks*, and in this section, we shall briefly provide their description and also a description of the associated training algorithm called *backpropagation*.

It seems natural that to describe non-linear behavior; we need to give up the previously used linear model of the perceptron. Note, a cascade of multiple purely linear layers could be described as a single linear layer. Therefore, we shall incorporate a non-linear function to the output of perceptron, as shown in Figure 4.1. Choice of the non-linear output function is not unique [22], however very commonly is used *sigmoid function*, traditionally denoted σ, and defined as

$$s(y) = \sigma(y) = \frac{1}{1 + \mathrm{e}^{-y}}. \tag{4.13}$$

Convenient property of sigmoid function, is that its derivative can be expressed as product of its outputs, specifically

$$\frac{d\sigma(y)}{dy} = \sigma(y)(1 - \sigma(y)), \tag{4.14}$$

and this property is utilized in description of backpropagation algorithm. Shape of sigmoid function together with its derivative are shown in Figure 4.10. Other output function with similar properties is e.g. $\tanh(y)$ [2].

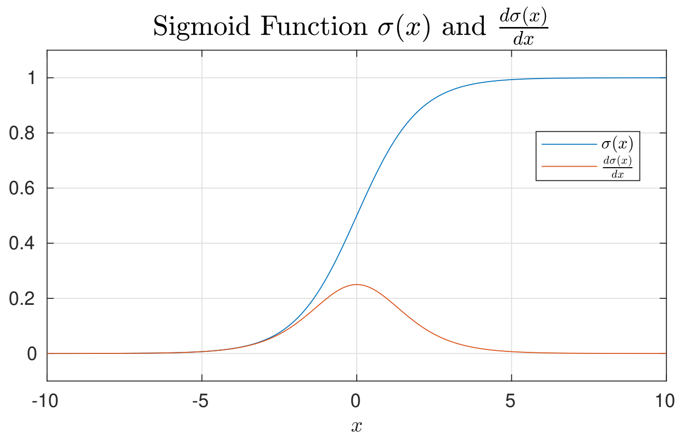


Figure 4.10: Sigmoid function and its derivative.

To present further notation, let us show in Figure 4.11 a simple neural network with one hidden layer, two inputs and one output. When desired to specify weight value in the network, it is common notation to use superscripts to denote number of the layer, e.g. $w_{ji}^{(1)}$ denotes weight in first layer, while subscript ji identifies weight w_{ji} going from ith to jth neuron [22]. The same notation is used to denote x_{ji} the input from ith to jth neuron. Bias entries are in Figure 4.11 represented as w_{i0}, to address them specifically. Note, that often the bias entries are omitted in schematic pictures in the literature. Subscript u is used to address uth output of the network, o_u, and, explicitly, the output is evaluated as non-linearity σ applied to sum of the inputs, i.e. $o_u = \sigma(y_u)$. In Figure 4.11 is only one output, o_1.

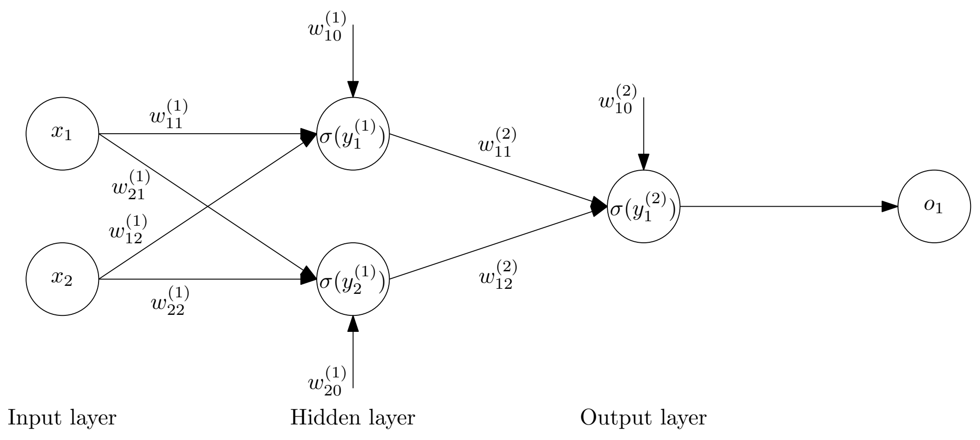


Figure 4.11: Simple neural network with one hidden layer.

Next, we shall provide description of Backpropagation algorithm, based on [22]. Let us first provide description of the variables. Backpropagation algorithm is used to find values of weights in a neural network with n_{in} input entries, n_{hidden} number of hidden neurons and n_{out} outputs. The learning is based on training data set $\mathcal{D} = \{(\boldsymbol{x}_d, \boldsymbol{t}_d)\}_{d=1}^{D}$, where $\boldsymbol{x}_d \in \mathbb{R}^{n_{\text{in}}}$ and $\boldsymbol{t}_d \in \mathbb{R}^{n_{\text{out}}}$. Another input to the algorithm is learning rate η. For clarity, the superscripts are here omitted and variables are distinguished as either "hidden" or "output".

Following derivation is based on the description provided in [22] with slight changes of notation. Consider dth training sample entering the network. Error function to be minimized by Backpropagation is the sum of squares of errors of all k network outputs o_k, i.e.

$$E_d(\boldsymbol{w}) = \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2, \tag{4.15}$$

and the aim is to find update rule of form

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E_d}{\partial w_{ji}}, \tag{4.16}$$

analogically to the situation of SGD. Let us use variable $y_j = \sum_i w_{ji} x_{ji}$ to denote weighted input to the jth neuron. Weight w_{ji} influences the rest of the network only through variable y_j, and therefore chain rule can be used to express derivative $\frac{\partial E_d}{\partial w_{ji}}$ as

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial y_j} \frac{\partial y_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial y_j} x_{ji}, \tag{4.17}$$

where x_{ji} is value passed from ith to jth neuron [22]. Next, we consider separately update rule for the output layer and hidden layer, beginning with the output layer, i.e. when jth neuron in Equation 4.17 is part of the output layer [22]. Similarly to Equation 4.17, value y_j influences only the corresponding output o_j and chain rule can be used to write

$$\frac{\partial E_d}{\partial y_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial y_j}, \tag{4.18}$$

and we can now address the right hand side terms of Equation 4.18 separately [22]. Firstly, using Equation 4.15 in Equation 4.18, we state,

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2, \tag{4.19}$$

and because derivative of right hand side summation is non-zero only when $j = k$, we evaluate it as

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j)^2 = -\frac{1}{2} 2 (t_j - o_j) = -(t_j - o_j). \tag{4.20}$$

Secondly, assuming usage of non-linear output function $\sigma(y_j)$, defined in Equation 4.13, and its derivative, stated in Equation 4.14, the second term of Equation 4.18, i.e. $\frac{\partial o_j}{\partial y_j}$, can be evaluated as

$$\frac{\partial o_j}{\partial y_j} = \frac{\partial \sigma(y_j)}{\partial y_j} = o_j (1 - o_j). \tag{4.21}$$

Substituting Equations 4.20 and 4.21 into Equation 4.18, we get

$$\frac{\partial E_d}{\partial y_j} = -(t_j - o_j) o_j (1 - o_j), \tag{4.22}$$

and using Equation 4.16, the update rule for output weights can be stated as

$$w_{ji} \leftarrow w_{ji} + \eta x_{ji} o_j (1 - o_j)(t_j - o_j). \tag{4.23}$$

In the remainder, let $\delta_k = -\frac{\partial E_d}{\partial y_k}$ for arbitrary neuron k with input y_k [22].

Further, to derive training rule for hidden neurons, we note that this time weight w_{ji} influences (in case of considered, fully connected, neural network) all the inputs y_j in the subsequent layer. Let this set of neurons with direct inputs y_j, be denoted as Successors(j) [22]. Derivative $\frac{\partial E_d}{\partial w_{ji}}$ in Equation 4.16 takes form

$$\begin{aligned}
\frac{\partial E_d}{\partial y_j} &= \sum_{k \in \text{Successors}(j)} \frac{\partial E_d}{\partial y_k} \frac{\partial y_k}{\partial y_j} \\
&= \sum_{k \in \text{Successors}(j)} -\delta_k \frac{\partial y_k}{\partial y_j} \\
&= \sum_{k \in \text{Successors}(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial y_j} \\
&= \sum_{k \in \text{Successors}(j)} -\delta_k w_{kj} o_j (1 - o_j) \\
&= -o_j (1 - o_j) \sum_{k \in \text{Successors}(j)} w_{kj} \delta_k,
\end{aligned} \tag{4.24}$$

and the update rule from Equation 4.16 can be then for the hidden neurons specified as

$$w_{ji} \leftarrow w_{ji} + \eta x_{ji} o_j (1 - o_j) \sum_{k \in \text{Successors}(j)} w_{kj} \delta_k. \tag{4.25}$$

The above description of Backpropagation is summarized as pseudocode in Algorithm 1. Note the following:

- The *stopping criterion* on line 3 might be of different forms. One possibility is to stop when some defined error floor is reached (e.g., factor $(t_k - o_k)$ is sufficiently small). Another choice is to train the network for a fixed number of *epochs*, which is the number of expositions of the whole training set to the network.
- The factors of the form $o_k(1-o_k)$ on lines 7 and 10, respectively, originates from the derivative of the sigmoid function, see Equation 4.13.
- Factor δ_k on line 7 is based on Equation 4.22. Similarly, value δ_h is based on result in Equation 4.25.
- Line 12 is analogical to Equation 4.12, where correction of each weight w_{ji} is proportional to value x_{ji}, that was previously multiplied by this weight.
- Many different modifications of Backpropagation exist. One of most common, mentioned e.g. in [1] and [22], is so-called *adding momentum*. Essentially, it means that the change of weight on line 12 takes into account also the value of change in the past iteration. More formally,

if $\Delta w_{ji}(n)$ denotes change of weight between neurons j and i in nth iteration, line 12 might be written as

$$w_{ji} \leftarrow w_{ji} + \eta\delta_j x_{ji} + \alpha\Delta w_{ji}(n-1),$$

where value $\alpha \in (0,1)$ controls how significant is influence of past change. The motivation for this modification is to avoid convergence of the training process towards local minimum instead of global minimum [22]. Effectively, this introduces memory to the algorithm, which bears information about the prevailing gradient descent vector. This *momentum* modification was also implemented in the shown examples.

Algorithm 1: Backpropagation Algorithm

```
   Data: $\mathcal{D} = \{(\boldsymbol{x}_d, \boldsymbol{t}_d)\}_{d=1}^{D}, n_{\text{in}}, n_{\text{hidden}}, n_{\text{out}}, \eta$;
   Result: Neural network weights $w_{ji}$;
 1 Create network with parameters $n_{\text{in}}$, $n_{\text{hidden}}$,$n_{\text{out}}$;
 2 Initialize all network weights to small random numbers;
 3 while Stopping criterion is not met do
 4     foreach $(\boldsymbol{x}_d, \boldsymbol{t}_d) \in \mathcal{D}$ do
 5         Input $\boldsymbol{x}_d$ to the network and compute all outputs $o_u$;
 6         foreach Network output neuron $k$ do
 7             Compute
                   $\delta_k = o_k(1-o_k)(t_k-o_k)$;
 8         end
 9         foreach Hidden neuron $h$ do
10             Compute
                   $\delta_h = o_h(1-o_h) \sum_{k \in \text{Successors}(h)} w_{kh}\delta_k$;
11         end
12         Update each weight as
                   $w_{ji} \leftarrow w_{ji} + \eta\delta_j x_{ji}$;
13     end
14 end
```

The backpropagation algorithm is widely used to train different structures of ANN. Its main advantage is simplicity. Further details of Backpropagation algorithm might be found e.g. in [1], [2] or [22].

4.4.1 Example: Backpropagation Algorithm to Learn XOR

ANN shown in Figure 4.11 was trained using Backpropagation Algorithm, as described in previous section in Algorithm 1. To illustrate ability of multilayer network to learn linearly non-separable function, let us present training of XOR function. Training data set $\mathcal{D} = \{(\boldsymbol{x}_d, t_d)\}_{d=1}^{4}$ is shown in Table 4.1

d	x_1	x_2	t_d
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

Table 4.1: List of XOR training data.

Computation was performed in Matlab, selected stopping criterion was 5 000 epochs, and the opted learning parameter was $\eta = 1$. In every training epoch, every training sample (x_1, x_2) was fed to the network, output o of the network was evaluated and compared to the desired training set value t. Figure 4.12 is provided to demonstrate the convergence of the training process. Therein we can see, how the modification of the weights over all epochs suppressed error term $(t - o)$. This term is a natural choice to measure the accuracy of the training since it is proportional to the deviation term δ_k shown on line 7 in Algorithm 1. It is interesting to see that the convergence process is not uniform.

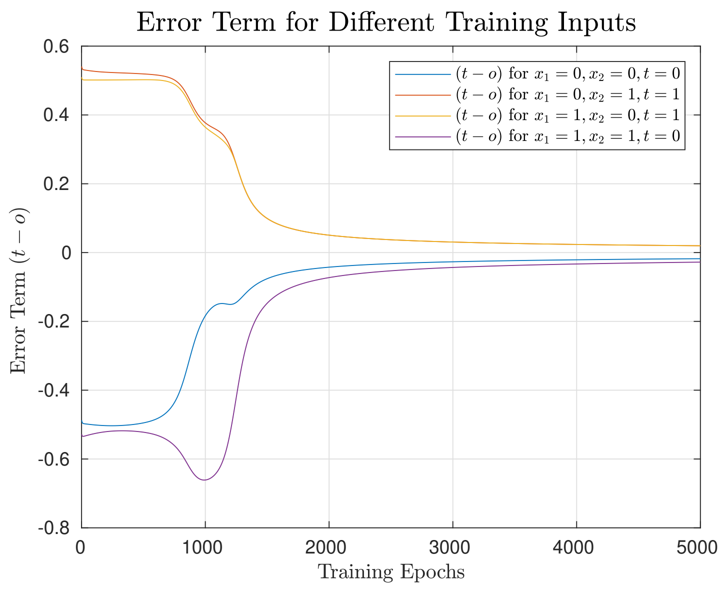


Figure 4.12: Demonstration of XOR training convergence: we can observe decreasing error values of the error terms during the training epochs.

In Figure 4.13 is shown a plot of original training data together with lines, which might be interpreted as decision boundaries. These lines were determined based on weight coefficients of the hidden neurons, by the same computation as shown in Equation 4.10. Zero values of XOR function are identified in the outer space of these lines, and unit values are captured between these lines.

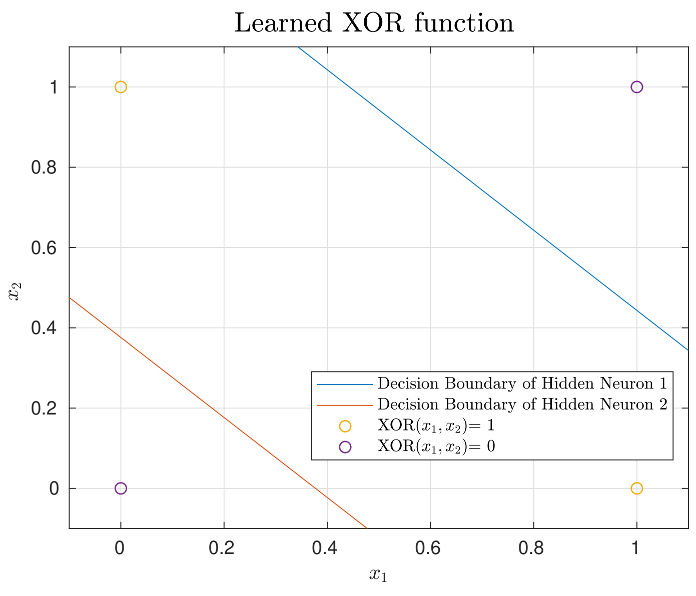


Figure 4.13: Trained XOR function demonstration: decision boundaries were determined based on the trained hidden layer weights.

4.4.2 Example: Auto-encoder

This example is based on [22]. In Figure 4.14 is shown simple structure of ANN called auto-encoder. In this example, the parameters are $n_{\text{in}} = 8$, $n_{\text{hidden}} = 3$ and $n_{\text{out}} = 8$, with training data given in Table 4.2.

d	$\boldsymbol{x}_d$	$\boldsymbol{t}_d$
1	0000 0001	0000 0001
2	0000 0010	0000 0010
3	0000 0100	0000 0100
4	0000 1000	0000 1000
5	0001 0000	0001 0000
6	0010 0000	0010 0000
7	0100 0000	0100 0000
8	1000 0000	1000 0000

Table 4.2: List of auto-encoder training data.

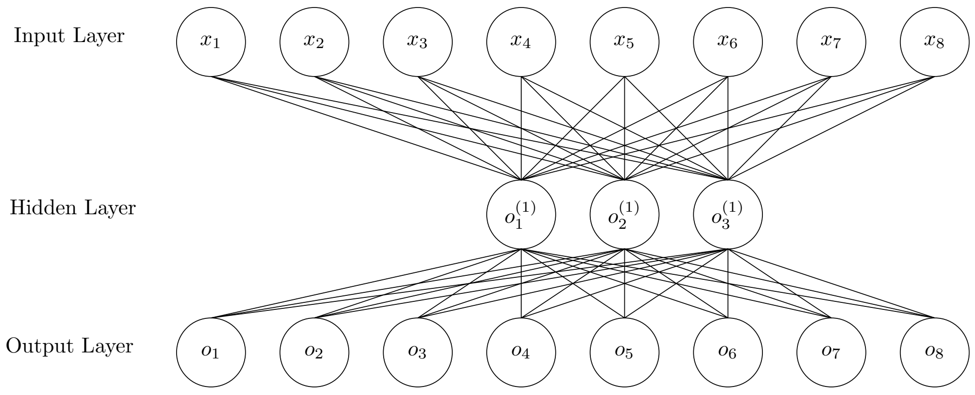


Figure 4.14: Structure of auto-encoder. Auto-encoders are trained to duplicate inputs to the output layer with a bottle-neck in the hidden layer(s).

Interpretation of the training data set in Table 4.2 is that auto-encoder is trained to copy the input of the network to the output. However, in this example, the hidden layer consists of only 3 neurons, and therefore the trained weights need to perform some form of compression. This example was implemented in Matlab. In Figure 4.15 is demonstrated convergence of the training process. The error of the output layer is shown for $d = 8$ different training inputs within 10 000 epochs.

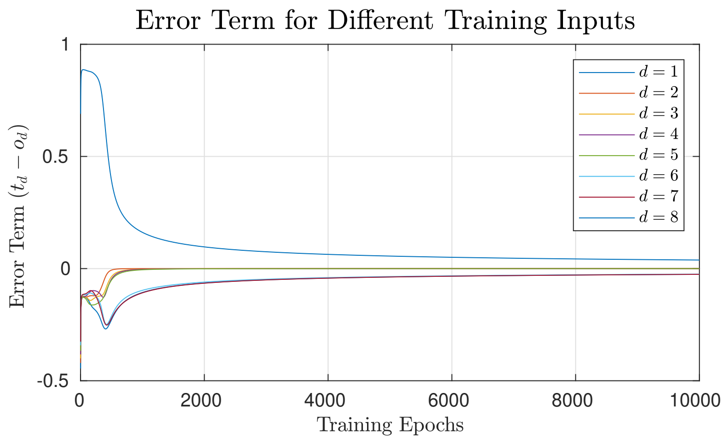


Figure 4.15: Demonstration of convergence of auto-encoder training. Having a hidden layer, the convergence process is not uniform and contains abrupt changes. In the figure, especially the inputs of the training sample denoted $d = 8$ converge very slowly.

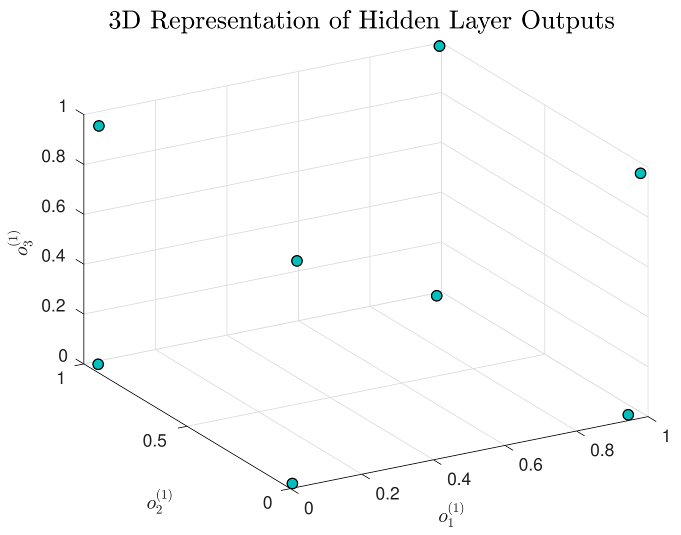


Figure 4.16: Representation of the auto-encoder hidden output values: variables $o_i^{(1)}$, $i = 1, 2, 3$, are shown as representation of 3-dimensional space, where 8 different outputs are described as 8 different combinations of 3 values that converge to binary-like representation.

Let us have a look at the outputs of the hidden layer. Weights of the hidden layer $w_{ji}^{(1)}$ were trained, such that 8 different training samples result in 8 different combinations of the hidden layer outputs $o_i^{(1)}$. Visualization of the situation is provided in Figure 4.16, where the 3 dimensions correspond to 3 hidden outputs. Different inputs are by the hidden layer represented almost as corners of a cube, that effectively perform some binary-like representation of the inputs.

Chapter 5

Exemplary Applications of Machine Learning Algorithms in WPLNC

5.1 Introduction

In this chapter are provided solutions to several exemplary problems that were recognized as suitable to be solved using a machine learning approach. All these examples are solved by training of designed ANN using the Backpropagation algorithm described in Algorithm 1. To avoid possible misinterpretations of the implemented setups of the training, we provide Figure 5.1, which illustrates the utilization of Algorithm 1.

Let us describe Figure 5.1. Firstly, the training data set of D samples is addressed. Previously, it was denoted as $\mathcal{D} = \{(\boldsymbol{x}_d, t_d)\}_{d=1}^{D}$. We emphasize that throughout the whole chapter, these samples correspond to received symbols in the constellation space. With the exception of Section 5.6, where the procedure is addressed in detail, we assume that all samples are received *before* the training process is started. Moreover, these samples *are fixed* during the whole training process (observation is not updated). This means that the training process is in this chapter always performed solely based on a single realization of $\mathcal{D}$. Further, all weights of the ANN are initialized with small random values. It is a typical approach, recommended in the literature on the topic of ANN, e.g., [22].

Following Algorithm 1, the training process consists of individual epochs. Denote E the number of epochs, let e counts individual epochs, initialized $e = 1$. As explained in Chapter 4, a single epoch corresponds to a *single exposure* of *all training samples.* In Algorithm 1, this corresponds to the cycle on lines 4–13. These samples are individually fed into the ANN. For each of the D samples, all the error terms δ_k are evaluated. Using terms δ_k, all the weights are updated. Quantitatively, within each epoch, e, performed with D training samples, all the weights are updated $D-$times. Expanding this evaluation, within E epochs of the training process with D training samples, all the weights are updated $E \cdot D-$times.

In the context of wireless radio communication, we stated that the training samples correspond to points in the constellation space. This implies that to obtain training data, we need *use the radio channel* $D-$times. The product $E \cdot D$ then corresponds to the *computational complexity* of the training process. Next, looking at Figure 5.1, we distinguished how the weights in all the steps of the training process depend on other variables. The weights are denoted as $w_d^e(A;B)$. Let us explain these indices. Easily, the upper index e indicates actual epoch of the training process; lower index d indicates, which training sample d out of D was in the given step fed into the ANN (i.e., enters the input layer and subsequently caused some output on the output layer).

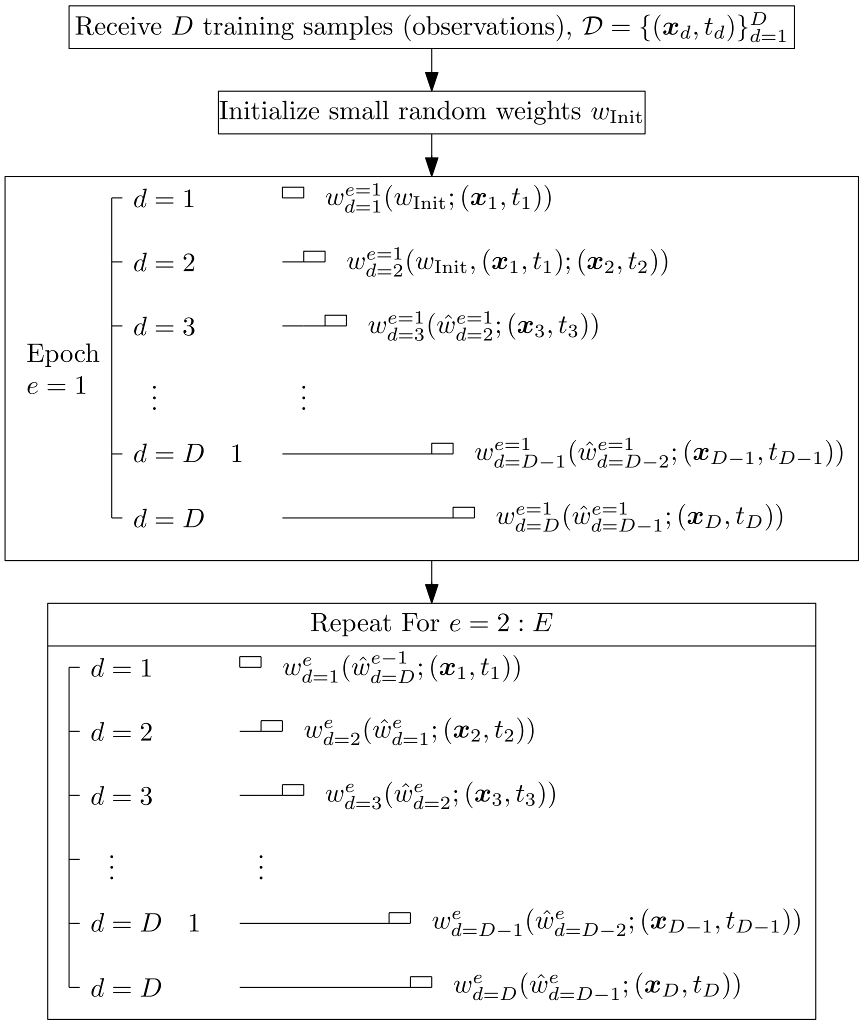

Receive D training samples (observations), $\mathcal{D} = \{(\boldsymbol{x}_d, t_d)\}_{d=1}^{D}$
Initialize small random weights w_{Init}
Epoch $e = 1$
$d = 1$
$w_{d=1}^{e=1}(w_{\text{Init}}; (\boldsymbol{x}_1, t_1))$
$d = 2$
$w_{d=2}^{e=1}(w_{\text{Init}}, (\boldsymbol{x}_1, t_1); (\boldsymbol{x}_2, t_2))$
$d = 3$
$w_{d=3}^{e=1}(\hat{w}_{d=2}^{e=1}; (\boldsymbol{x}_3, t_3))$
$d = D$ 1
$w_{d=D-1}^{e=1}(\hat{w}_{d=D-2}^{e=1}; (\boldsymbol{x}_{D-1}, t_{D-1}))$
$d = D$
$w_{d=D}^{e=1}(\hat{w}_{d=D-1}^{e=1}; (\boldsymbol{x}_D, t_D))$
Repeat For $e = 2 : E$
$d = 1$
$w_{d=1}^{e}(\hat{w}_{d=D}^{e-1}; (\boldsymbol{x}_1, t_1))$
$d = 2$
$w_{d=2}^{e}(\hat{w}_{d=1}^{e}; (\boldsymbol{x}_2, t_2))$
$d = 3$
$w_{d=3}^{e}(\hat{w}_{d=2}^{e}; (\boldsymbol{x}_3, t_3))$
$d = D$ 1
$w_{d=D-1}^{e}(\hat{w}_{d=D-2}^{e}; (\boldsymbol{x}_{D-1}, t_{D-1}))$
$d = D$
$w_{d=D}^{e}(\hat{w}_{d=D-1}^{e}; (\boldsymbol{x}_D, t_D))$

Figure 5.1: Illustration of how training process of ANN is implemented throughout Chapter 5.

Further, parameter A bears information about what *past* variables affected the ANN. For $e = 1, d = 1$, the weights are a function of its randomly initialized values w_{Init}; therefore $A = w_{\text{Init}}$. The variable B then informs, what *actual* variable affects the result of the training. For $e = 1, d = 1$, the first training sample $(\boldsymbol{x}_1, t_1)$ was fed into the ANN, and therefore $B = (\boldsymbol{x}_1, t_1)$. Analogically for $e = 1, d = 2$, the weights will be modified based on the past values $A = \{w_{\text{Init}}, (\boldsymbol{x}_1, t_1)\}$, and the *actual* update of the ANN depends on the second training sample $B = (\boldsymbol{x}_2, t_2)$. Note, that parameters A and B are separated with a semicolon. Subsequent values of parameter A are then denoted with a hat ($\hat{w}_d^e$), to address that the weights were updated based on its previous values (estimates of the weights).

The small rectangles on the horizontal lines in Figure 5.1 visualize what sample d out of D actually entered the ANN. The bottom block of the diagram then denotes that the process repeats $E-$times.

5.1.1 Notes to the Implementation

My own MATLAB implementation of the Backpropagation Algorithm from Chapter 4 was used for all the considered examples. By most, the difference between individual scripts is in parameters of the network and processing of the results for the purpose of visualization. Therefore, this text contains only a single written implementation of the Backpropagation Algorithm and may be found in Appendix A.1. This particular example was developed for an example in Subsection 4.4.1.

Of course, all the implemented MATLAB scripts may be found on the attached CD. Eventually, on the server `https://dspace.cvut.cz/` may be these scripts also found. Therein, all the scripts are for convenience ordered in subfolders according to Sections of this Thesis and typically contain also *.mat files with achieved results for easy reconstruction of output Figures (that are typically also included).

From the perspective of decision theory (see Subsection 1.5.4 in [2]), our implementation of the ANN corresponds to the determination of discriminant function, which is utilized to perform the classification directly, i.e., a probabilistic model is not determined.

Finally, all the tasks were interpreted and solved as classification problems. In the studied literature on the topic of the ANN, no strict rules were determined, concerning the architecture of the network (number of hidden nodes, number of hidden layers, number of output nodes). In principle, the arbitrary value might be achieved on the output of the ANN. Eventually, (in extremal case) this might be used to train the ANN, such that all the classes are indicated using different output values of a single neuron. However, my own implementation utilizes the sigmoid function to threshold the output values of the output layer between 0 and 1. A number of C individual classes are then indicated using C individual outputs, and the maximal value is used for decision. Of course, this corresponds to the training data set, where the classes are indicated as a "one-hot" vector. The training data set is always explained in the subsequent examples. Note, so-called *soft-max* output might be easily implemented to interpret such outputs as a probability of being in the class [1].

5.2 Simplified Scenario: Classification of Transmitters on Relay Node

The following example is in application of ANN to classification of constellation diagrams motivated by [5].

Considering the WPLNC system, all nodes need to be aware of complete topology. Relay nodes decide on how to process received signals based on knowledge of the origin of the received signals. In this section, we shall address the situation when the relay node lacks the confidence of what neighbors are radio-visible. Let us consider only a simplified scenario, where we design ANN to perform the classification of the signal received by the relay node. In Figure 5.2 is shown, how BPSK modulation, received from one to six users, superimposes on the relay node antenna, assuming zero phase shift. With this assumption, the number of possible constellation points equals the number of users, plus one. A similar problem was addressed in [16], in the context of the initialization of the WPLNC system.

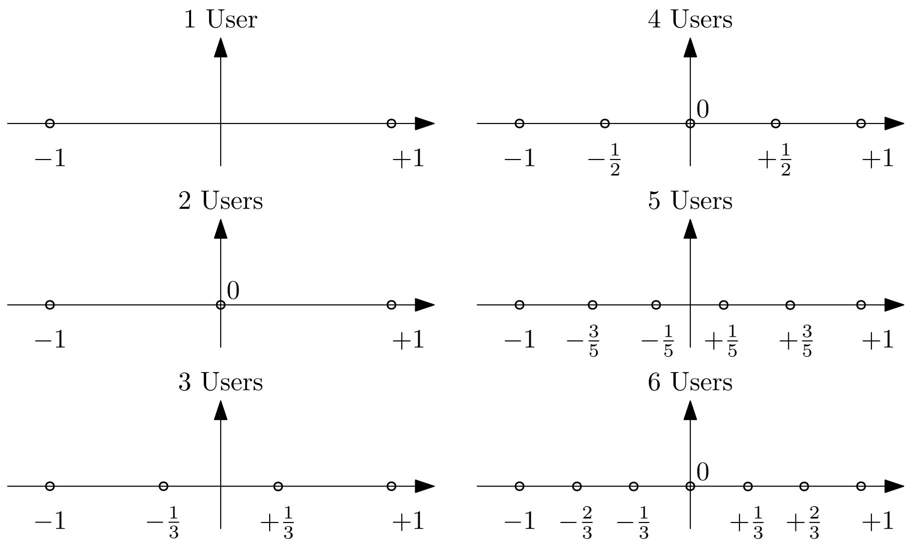


Figure 5.2: Demonstration of how multiple BPSK modulation superimpose on the relay antenna. Possible constellation points are shown. Note, the horizontal axis is normalized to 1.

Further, let us assume perfect time synchronization and knowledge of modulation pulse, such that the relay node can obtain I/Q representation of the received signal. For simplicity, we assume that the only effect of the channel is AWGN. With all these simplifications, the considered relay node shall be able to collect a set of L samples represented in the constellation space. These samples originate by superimposing signals from u transmitters, $u = 1, \dots, U = 6$, on the relay receiver antenna.

We assume that the maximum number of users is $U = 6$. Let us describe how to retrieve the number of users U, that transmit BPSK modulation to the relay node. To do so, we shall use ANN with one hidden layer, as described in the previous chapter. The network inputs shall be denoted $x_1, x_2, \dots, x_N$. As stated, training data to be used are points in constellation space; however, we shall perform *feature extraction* before training the ANN. Let us describe, how the training set $\mathcal{D} = \{(\boldsymbol{x}_d, \boldsymbol{t}_d)\}_{d=1}^{D}$ was created. Note, $\boldsymbol{x}_d \in \mathbb{R}^N$ and $\boldsymbol{t}_d \in \mathbb{R}^U, U = 6$. The ANN desired outputs are for each training sample

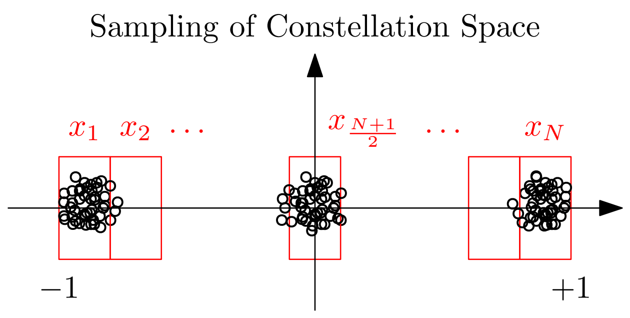


Figure 5.3: Description of the sampling process: In the figure is demonstrated the performed process of feature extraction. The constellation space was along real axis divided into N equally large rectangular areas and counts of received samples within these areas were determined.

provided in the training set as a vector $\boldsymbol{t}_d$. As a result of this, the designed system uses vector $\boldsymbol{t}_d$, where all its values but one are zero and uth position is equal to 1 if the corresponding training input originated as a superposition of transmissions of u users, where $u = 1, \dots, 6$.

The preprocessing is illustrated in Figure 5.3. First of all, the received constellation points were normalized, such that real and imaginary parts take values between -1 and 1. This is necessary to suppress dependence of the classification on received power level and eases the sampling process since we want to obtain a fixed number of features. Next, the constellation space was divided along the real axis into N rectangular areas, where N is chosen odd to reach symmetry, e.g., $N = 19$. The number of constellation points that fall into these rectangles corresponds to the input x_n. This might be understood as sufficient statistics for this task (simple projection). It is recommended to avoid training of ANN with excessively high values, e.g., in [1]. Therefore, the elements of training set $\boldsymbol{x}_d$ were normalized as

$$\boldsymbol{x}_d \leftarrow \frac{\boldsymbol{x}_d - \mu(\boldsymbol{x}_d)}{\delta(\boldsymbol{x}_d)}, \tag{5.1}$$

where $\mu(\boldsymbol{x}_d)$ and $\delta(\boldsymbol{x}_d)$ denote mean value of $\boldsymbol{x}_d$ and standard deviation of $\boldsymbol{x}_d$, respectively. An example of normalized training input is shown in Figure 5.4. We can compare it with Figure 5.2 and observe, that for the provided number of $L = 500$ received superimposed samples, the distribution over constellation points is not uniform. Further, note that examples of training samples in Figure 5.4 are generated for the high value of SNR. Later, results will be provided for different values of SNR. Intuitively, the higher number of inputs N should provide better resolution and lead to better performance with respect to accuracy at lower values of SNR. On the other side, it might also cause the ANN to be more complex with a larger number of parameters that need to be determined.

Note, in this thesis, whenever addressing a scenario with multiple superimposing constellations, SNR is considered with respect to the first user.

Having the preprocessed training data set, the ANN was implemented with number of inputs $n_{\text{in}} = N = 19$, number of hidden neurons $n_{\text{hidden}} = 10$, number of outputs $n_{\text{out}} = U = 6$, and learning rate $\eta = 0,5$. Size of the training set was selected, such that each of 6 considered combinations has 150 realizations, i.e. $D = 6 \cdot 150 = 900$, and determined sufficient number

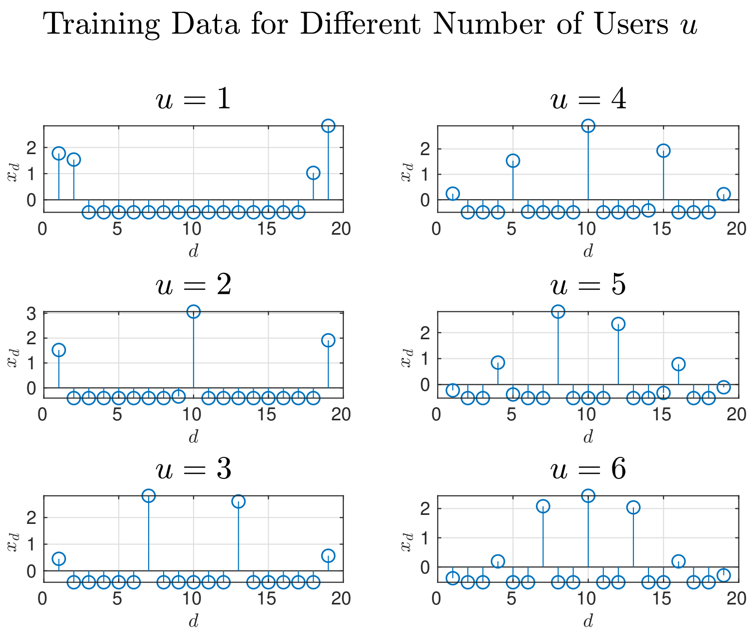


Figure 5.4: Example of training data used to determine number of transmitters u. This figure is analogical to Figure 5.2. Note, however, that number of received constellation points is not distributed uniformly and constellation points with smaller real part were received more often.

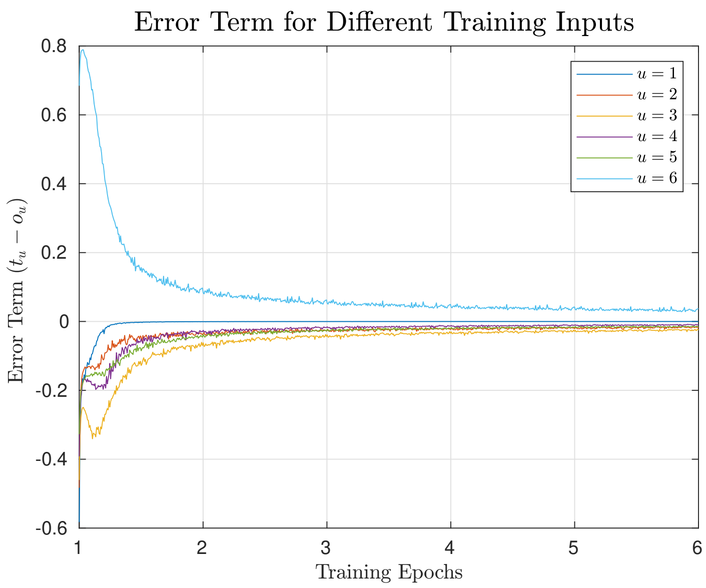


Figure 5.5: Demonstration of convergence of the training process: During 5 epochs, the training set containing 900 training examples was exposed to the network and convergence was successfully reached.

of training epochs was 5. In Figure 5.5 is demonstrated convergence of the training process for high value of SNR.

Further, the training process was verified using additional validation set $\mathcal{V} = \{(\boldsymbol{x}_d, \boldsymbol{t}_d)\}_{d=1}^{D}$, which was generated using exactly the same process as a training set $\mathcal{D}$, using the same parameters, but differs in realizations of AWGN. The weights of the ANN obtained by training of the network with the training set were used to process the validation set $\mathcal{V}$, and the system reached 100% accuracy.

Let us now investigate how the designed algorithm performs for different values of SNR. An initial attempt is shown in Figure 5.6. Accuracy of classification is in the whole chapter defined as

$$\text{Accuracy} = \frac{\text{Number of Correctly Classified Validation Samples}}{\text{Size of Validation Set}} \cdot 100\%. \tag{5.2}$$

Let us emphasize that in Figure 5.6, the accuracy was evaluated, such that training data were generated for the corresponding SNR, the system was then trained, and lastly tested using validation data with the same SNR as training data set. (Specifically, this means that 41 markers in Figure 5.6 correspond to accuracy of 41 separately trained ANN for a single value of SNR.) This evaluation was performed for values between 0 and 20 dB with step 0,5 dB. Moreover, in Figure 5.6 is the accuracy recorded for N=19 and for N=39 inputs of the ANN, corresponding to the granularity of the sampling during the process of feature extraction. We observe that finer sampling results in only a very slight improvement in performance.

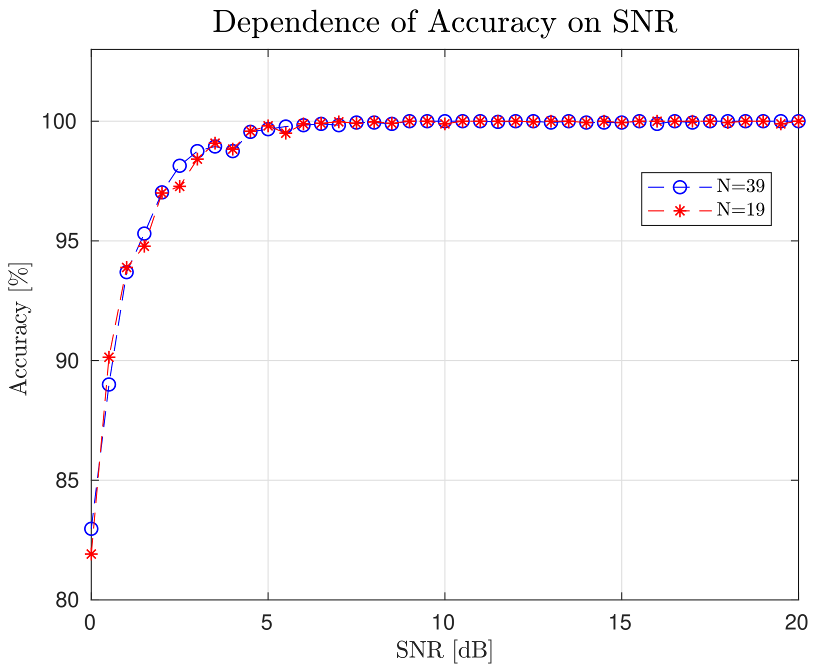


Figure 5.6: Demonstration of the accuracy of the system trained for different levels of SNR. On the interval between 0 dB to 20 dB, the accuracy of the system is excellent. Nevertheless, bellow 5 dB accuracy drops very rapidly, and the number of inputs to the ANN seems not to be of much importance.

The results in Figure 5.6 seem to be promising. Let us now focus on the *generalization* ability of the designed system. We shall investigate how this setup works for unknown values of SNR, which is a slightly more practical approach. For that purpose, we shall use Figure 5.7. In this experiment, all the above-described parameters of ANN were preserved, but the number of training epochs was extended to 10. Three different approaches were tested, and the explanation follows. In the figure, green markers stand for the most naive approach, where testing data were generated with SNR = 15 dB, the ANN was trained, and validation was performed for different values of SNR. Blue markers evaluate training setup, such that training data were generated for values of SNR between 0 dB to 20 dB, and all these training samples were exposed to the ANN during the training epochs. The blue markers perform much better for low SNRs. The green markers showed a similarly good performance above 10 dB. The reason for this might be that the weights of the ANN tried to capture the situation for a too broad range of SNR, and generalization was not possible. If we look back at Figure 5.6, therein the situation of rapid drop-off at 5 dB was similar to that in Figure 5.7. This means that generally, this system has the poor capability to work precisely at low SNRs, more specifically below 5 dB. Therefore, last approach was tested, denoted in Figure 5.7 with red markers. Red markers correspond to training strategy, where we a priori gave up a chance to train the system for values of SNR below 5 dB. The training data set was generated for SNR values between 5 dB to 20 dB, and obviously, it outperformed previous approaches on this interval during testing with the validation data set.

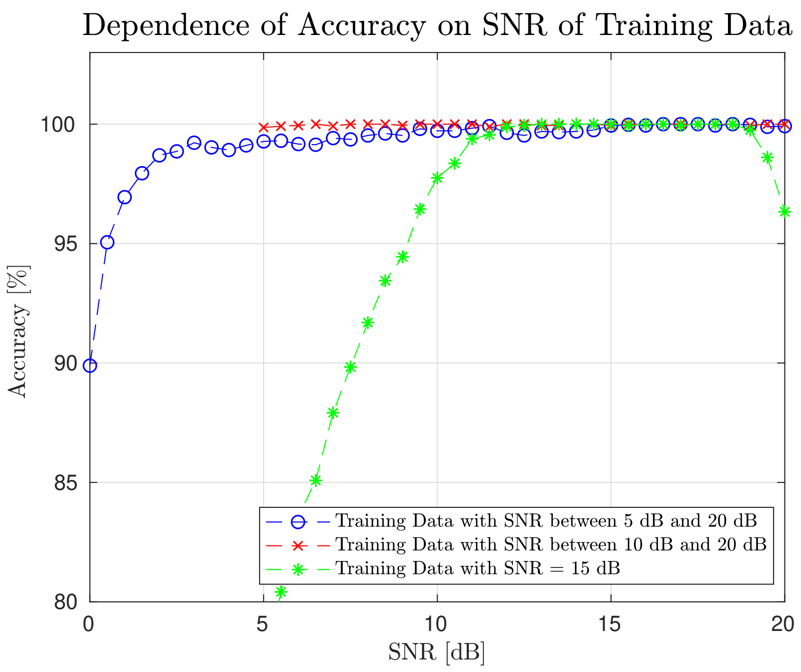


Figure 5.7: Demonstration of generalization issues of the system tested on different values of SNR. Green markers evaluate system trained for SNR = 15 dB and tested with validation data with different SNRs; blue markers describe accuracy of system trained with training data on multiple levels of SNR and tested on whole range of SNRs; red markers show accuracy of system, that was trained and tested only for usage on a subset of SNRs.

5.3 Classification of Hierarchical Symbols in simplified 2WRC with BPSK

5.3.1 Problem Formulation

In this example, let us consider the application of WPLNC on 2WRC with BPSK modulation. We shall assume a situation where the effect of channel fading is not neglected. Specifically, our model shall include flat fading, which affects the amplitude and phase of the received signal and also the effect of AWGN.

First, we shall present the system model. Let the observation of the received signal, $x \in \mathbb{C}$, be described as

$$x = s_A(b_A) + hs_B(b_B) + w, \tag{5.3}$$

where information bits $b_A, b_B \in \{0, 1\}$ are modulated using BPSK to constellation symbols $s_A,\ s_B\ \in\ \{\pm\ 1\}$, $h \in \mathbb{C}$ is relative fading parameter, $w \in \mathbb{C}$ is complex AWGN with variance σ_w^2 and considered HNC map of hierarchical symbol $b \in \{0, 1\}$ is bitwise XOR, i.e.

$$b = \chi(b_A, b_B) = b_A \oplus b_B. \tag{5.4}$$

With the model above, the task is to determine the hierarchical symbol b based on observation x. Given the observation model, this situation was formally addressed in [19], and decision regions were analytically determined. In this section, we revisit this example. We shall try to determine decision regions for hierarchical symbol b using ANN. Initially, we try to determine what is the ability of ANN to reach the same results as in [19]. Firstly, we verify the performance of ANN-based decision regions. An extension of the task might be a usage of higher-order modulation in the system model, e.g., QPSK. More practical utilization of decision regions, determined using machine learning, would be to apply it to a situation where the decision regions are not yet analytically derived due to the complexity of more complicated channel models, e.g., non-linear. This addresses the model deficit, as discussed in Chapter 2.

Again, we assume perfect time synchronization and knowledge of modulation pulse. Observation x can be represented as constellation point. To keep previous notation, let real and imaginary part of complex number x be denoted as x_1 and x_2, i.e. $x_1 = \text{Real}(x)$, $x_2 = \text{Imag}(x)$. To build up a reasonable scenario, let us assume, that training data set $\mathcal{D} = \{(\boldsymbol{x}_d, \boldsymbol{t}_d)\}_{d=1}^{D}$ is provided. As previously, $\boldsymbol{x}_d = (x_1, x_2)_d$ is vector of training samples used as inputs of the ANN. A model situation might be, such that transmitters N_A and N_B modulate and transmit number of D bits b_A and b_B, respectively, that are predetermined and used as pilot signals at the receiver. These bits form pseudo-random sequences $\{b_{A,d}\}_{d=1}^{D}$ and $\{b_{B,d}\}_{d=1}^{D}$. The receiving relay node R is aware of pilot sequences $\{b_{A,d}\}_{d=1}^{D}$ and $\{b_{B,d}\}_{d=1}^{D}$, and utilizing Equation 5.4 is provided with sequence of desired hierarchical symbols $\{b_d\}_{d=1}^{D} = \{b_{A,d} \oplus b_{B,d}\}_{d=1}^{D}$.

Addressing the desired outputs of the trained ANN for dth training sample, denoted as previously by $\boldsymbol{t}_d$, let us use ANN with two outputs, corresponding to two considered hierarchical symbol values $\{0, 1\}$. Let $\boldsymbol{t_d} = (t_1, t_2)_d$, such that t_1 indicates hierarchical symbol 1 and t_2 indicates hierarchical symbol 0. It is summarized in following Table 5.1, and effectively means, that $t_1 = b_A \oplus b_B$ and $t_2 = 1 - t_1$. Of course, in this case, we could use ANN with only one output with non-linear sigmoid function and use its values directly to determine hierarchical symbols; however, we design this example as a classification task, and this setup is more suitable for a possible extension to a system with more hierarchical symbols than only two.

$b_{A,d}$	$b_{B,d}$	b_d	$\boldsymbol{t}_d = (t_1, t_2)_d$
0	0	0	(0, 1)
0	1	1	(1, 0)
1	0	1	(1, 0)
1	1	0	(0, 1)

Table 5.1: Explanation of training input $\boldsymbol{t_d}$.

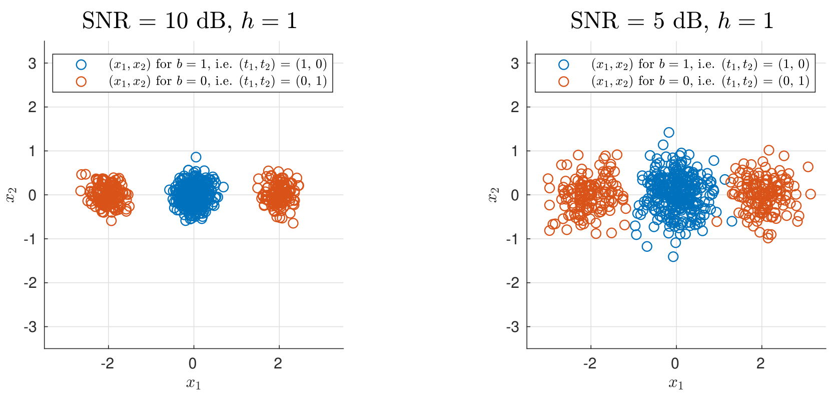


Figure 5.8: Demonstration of different levels of SNR of the training data. Different levels of SNR of training data might impact level of generalization achieved by the trained ANN.

5.3.2 Illustration of Degrees of Freedom of the Problem

Properties of system model, defined in Equation 5.3 are illustrated in following Figures 5.8-5.11, where number of training samples is $D = 300$. Firstly, in Figure 5.8 is shown training data set for different levels of SNR. The effect of spreading the constellation points is trivial. However, we utilize this to mention a regularization technique, used to improve a generalization ability of ANN. It was described, e.g., in [23] or [24], that larger noise of input training data might increase the performance of the trained system.

Later, it will be graphically addressed when illustrating decision regions trained by the ANN. Intuitively, training the ANN with data containing a more significant level of noise forces the weights to be more tolerant of outliers. Eventually, this means that the result of the training process might be improved when artificially corrupting the training data with additional noise. In [23], it is concluded that improvement of the training process might be achieved, when several epochs of the training are performed with training data injected with additional noise and subsequently continue with clean training data.

In further examples, SNR is fixed to 10 dB and results for different values of fading parameters h are shown. The examples are easy to interpret, but are hereby utilized to visually demonstrate degrees of freedom of the desired classification system. In Figure 5.9, the fading parameters are $h = \mathrm{e}^{\pm\frac{\mathrm{j}\pi}{4}}$. Factors $\frac{\pm\pi}{4}$ correspond to angles in constellation space, between constellation points of modulated signals $s_B(b_B)$ and $s_A(b_A)$.

Figure 5.10 extends example from Figure 5.9 and demonstrates, that after change of the angle in exponent by a factor of π, i.e. hereby for fading parameters $h = \mathrm{e}^{\frac{\mathrm{j}\pi}{5}}$ and $h = \mathrm{e}^{\frac{\mathrm{j}\pi}{5}+j\pi} = \mathrm{e}^{\frac{6\mathrm{j}\pi}{5}}$, the groups of constellation points

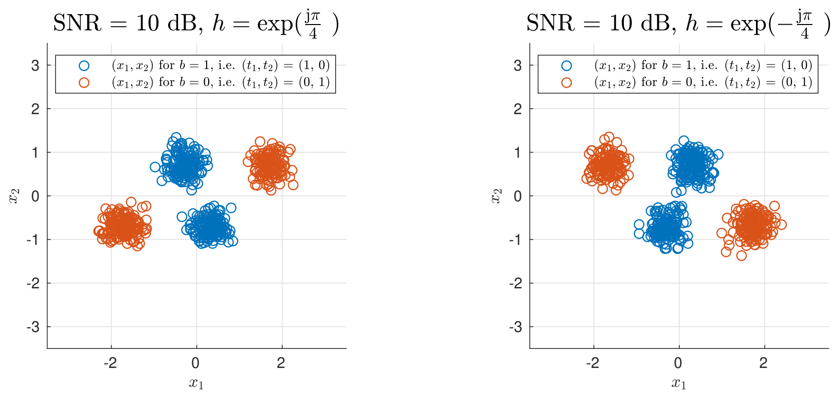


Figure 5.9: Demonstration of effect of different angles of fading parameter h. Positive or negative angle of the complex fading parameter defines, whether the superposition angle is counter-clockwise or clockwise.

corresponding to hierarchical symbols $b = 0$ and $b = 1$ interchange positions.

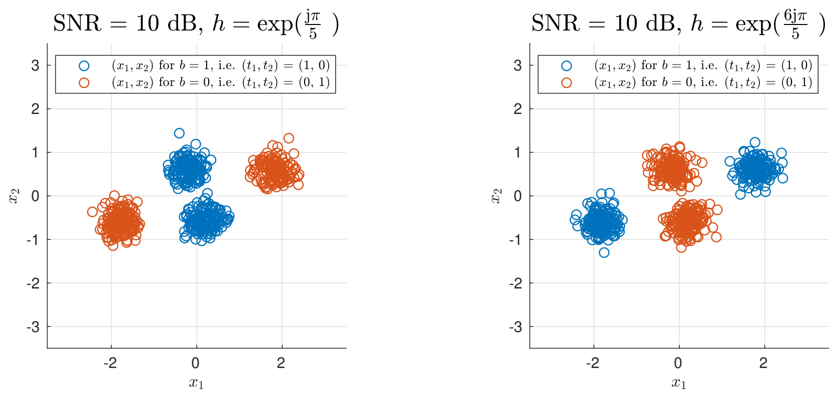


Figure 5.10: Demonstration of effect of π radians difference between angles of two fading parameters h. The groups of constellation points corresponding to different hierarchical symbols interchange position.

Finally, in Figure 5.11 is demonstrated how the magnitude of the fading parameter affects the superimposed constellation. When magnitude $|h|$ decreases from $|h| = 0,9$ to $|h| = 0,5$, it results in a shorter distance between the groups of constellation points for the different hierarchical symbols. Effectively, this should force the classification regions to be sharper.

5.3.3 Parameters of the simulation

Having described the observation model and training set, we can move to the training procedure. Given two dimensions of constellation space, number of inputs to the network is $n_{\text{in}} = 2$ and two possible hierarchical symbols determine number of outputs $n_{\text{out}} = 2$. To the purpose of evaluating the performance of the training, it is necessary to visualize the decision regions. Especially for ANN with a large number of hidden neurons, it is complicated to show determined decision regions based only on the trained weights, as, e.g., in Figure 4.13.

As a result of this, we opted to sample the constellation space, feed the samples to the trained network and, after rounding operation, to show the outputs. Moreover, we need to have some reference solutions to evaluate

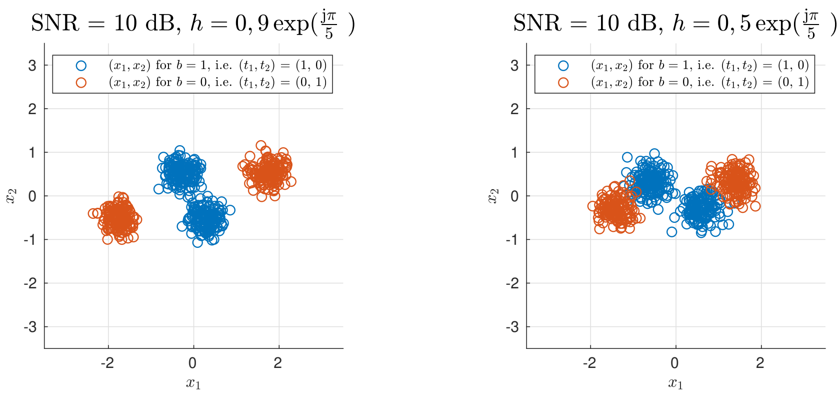


Figure 5.11: Demonstration of effect of different magnitudes of two fading parameters h.

how good the trained model is. Two reference solutions are hereby provided. Firstly, for its simplicity, evaluation is based on a metric, which is Euclidian distance of the tested points from the constellation points, neglecting the noise term w. The second method that is described later is hereby referred to as "True Metric Map" and was shown in [19].

Further, let us note that this task was handled experimentally. The reason is, there are not any general rules on how to design the parameters of ANN. Fixing the number of inputs and outputs, degrees of freedom of the ANN to be designed are: (1) number of hidden layers, (2) number of neurons in each hidden layer, and (3) learning rate. Additionally, concerning the transmission, we need to determine: (4) the number of training samples D, and (5) number of training epochs, such that desired generalization is reached. Finally, as number (6), it comes out that the critical parameter is hereby SNR of training data.

All of these parameters have a very significant impact on the properties of convergence and, specifically, on time of training. While the overall number of neurons in ANN determines the number of operations performed during training, it also affects its approximation abilities. Consequences of improperly chosen learning rate η are either extremely long convergence time or losing the ability to minimize the error function. The number of training samples D obviously affects the computational time required per epoch, and, together with its SNR, it rules the ability of generalization. An exhaustive number of computer simulations was performed to map these effects, and, in the sequel of this section, let us provide a summary of the results.

5.3.4 Introductory Example

First, let us start with an illustrative example. In MATLAB was implemented ANN with two hidden layers, having $n^{(1)}_{\text{hidden}} = 10$ and $n^{(2)}_{\text{hidden}} = 10$ neurons in hidden layers. Learning rate was selected as $\eta = 0,5$. Let $D = 300$ and SNR $= 10$ dB and the training data set $\mathcal{D} = \{(\boldsymbol{x}_d, \boldsymbol{t}_d)\}_{d=1}^{D}$ was generated with fading parameter $h = \mathrm{e}^{\frac{j\pi}{4}}$. Further, the trained network was tested using validation set with $2 \cdot D$ samples, i.e. $\mathcal{V} = \{(\boldsymbol{x}_d, \boldsymbol{t}_d)\}_{d=1}^{2D}$, preserving the fading parameter h and with fixed SNR $= 15$ dB. Training lasted 2 000 training epochs.

As mentioned, according to [19], two possible reference maps are offered.

The first, hereby denoted as *Distance-Based Map* is determined purely based on shortest Euclidean distance in constellation space between coordinated of classified points (i.e., (x_1, x_2) and true coordinated of constellation points corresponding to specific hierarchical symbols, determined with knowledge of fading parameter h and neglecting noise term.

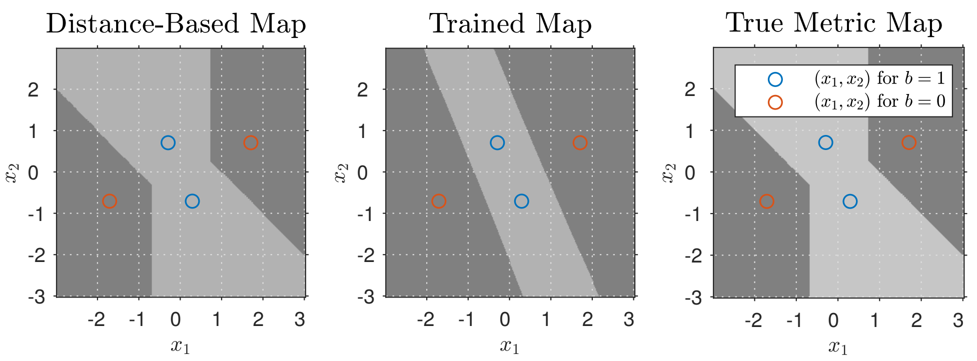


Figure 5.12: Comparison of reference decision maps and trained decision map in small-scale example with SNR = 15 dB.

The second reference map denoted in the sequel as *True Metric Map* was derived in [19]. It takes into account the specific variance of AWGN and the contribution of all constellation points, corresponding to given hierarchical symbols. Note, Distance-Based Map is an approximation of True Metric Map, where only the greatest operand is counted.

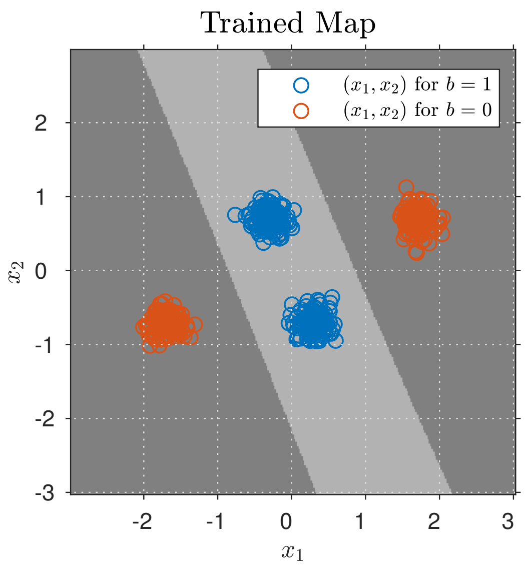


Figure 5.13: Introductory example shows, that though all samples of validation set generated with SNR = 15 dB were correctly classified, generalization ability of the system is very low and does not come too close to the reference solutions.

Result of training process is shown in Figure 5.12. Two levels of gray color are used to distinguish the decision regions corresponding to the hierarchical symbols, and results are shown for approximated Distance-Based Map (left-hand side), determined Trained Map (middle), and True Metric map (right-hand side). Constellation points corresponding to the hierarchical symbols are also shown for reference.

Note, the shaded decision regions were determined, such that the constellation space was sampled, and individual samples were processed. For the trained decision map, this means that samples of constellation space were subsequently fed to the trained ANN, and rounded outputs, corresponding to specific hierarchical symbols, were recorded.

Though we can see in Figure 5.13, that all samples of the validation set $\mathcal{V}$ were correctly classified, the shape of the trained decision boundary mostly consists of two lines and does not come very close to the reference maps. This means that reached generalization ability of the trained system is terrible, and effectively is equivalent to the simple XOR-example, shown in Figure 4.13.

5.3.5 Regularization through Noise

Since the illustrative results shown in the previous subsection were not very satisfactory, an effort was made to tune the parameters of the simulation. A significant improvement of the results was observed when significantly decreasing SNR of training data. This was recognized as a regularization technique, as addressed previously, and formally described, e.g., in [23] and [24].

Further, it is shown that training the ANN with training data set with very low SNR improves the performance of training. Further, let us provide the results of simulations in two setups. The following parameters were heuristically determined based on a large number of experiments: SNR and size of training data set, number of neurons in hidden layers, learning rate η. The previous example was extended, and as a result of this, we present obtained results.

Though this form of regularization is later shown to improve the trained result significantly, it also causes a problem. The issue is that the ANN is successfully trained using a training data set with some specific SNR; of course, the reference solution is obtained using the same SNR. However, the trained ANN is then tested with data sets generated with a wide range of SNRs.

Therefore, the problem is that different SNR might be used for successful training than for the real application of the system. Later, the corresponding reference solution is therefore provided based on the actual SNR, while a single trained system is used to evaluate performance for all values of SNR. Eventually, an artificial noise might be added to the received "pilot" samples to reduce the SNR.

In the provided results, this problem is not further addressed.

Setup 1

Parameters of simulation in Setup 1 are shown in Table 5.2. The first exemplary result is for value of relative fading $h = \exp\left(\frac{\mathrm{j}\pi}{4}\right)$. In Figure 5.14, we can see visualization of training data set.

In Figure 5.15 are shown resulting decision maps. Though the result is not perfect, we can see that the shape of trained decision maps approximately preserves the shape of reference solutions. Note that the trained Map is not symmetric as it is supposed to be. Further, True Metric Map is seen to be very smooth, while Distance-Based Map is not. It is caused by a fact, that True Metric Map takes into account variance of the noise.

Parameter	Value
η	0,05
$n_{\text{hidden}}^{(1)}$	40
$n_{\text{hidden}}^{(2)}$	40
SNR of training data set	-1 dB
D	7 000
Number of training epochs	500

Table 5.2: Parameters of simulation in Setup 1.

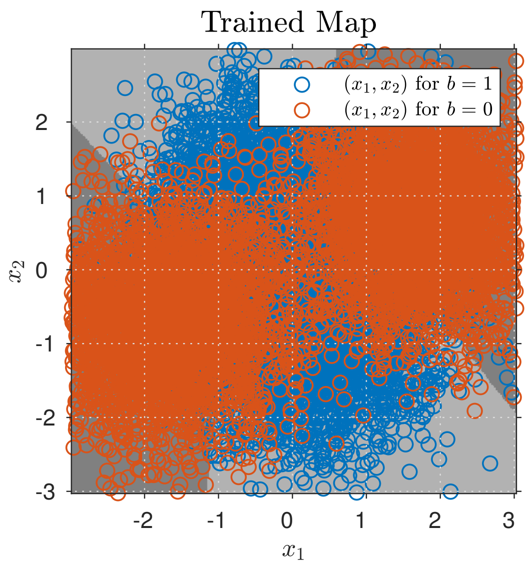


Figure 5.14: Setup 1: Illustration of training data set for $h = \exp\left(\frac{j\pi}{4}\right)$.

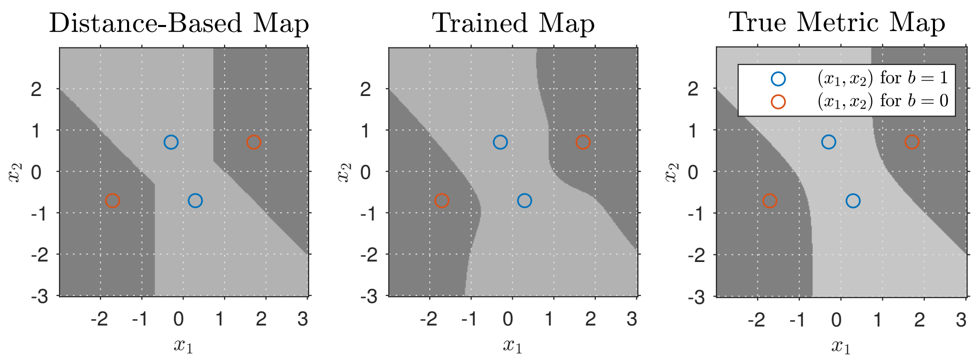


Figure 5.15: Setup 1: Comparison of decision maps for $h = \exp\left(\frac{j\pi}{4}\right)$.

To evaluate performance of trained system, we use Figure 5.16 , that shows accuracy, evaluated in % and Figure 5.17, that shows evaluation of Bit Error Rate (BER) as a function of SNR. Note, for evaluation of BER was used validation set of 10^5 samples, and at this moment, it is understood as the rate of wrong hierarchical symbol classification. We can observe that for values of SNR above 8 dB, the performance is overall outstanding. From both figures, we can read out that both reference solutions slightly outperform the trained system. Also, we emphasize that the True Metric Map was computed for each SNR separately. Of course, the Distance-Based Map does not change with SNR at all, and Trained Map was determined for SNR $=-1$ dB.

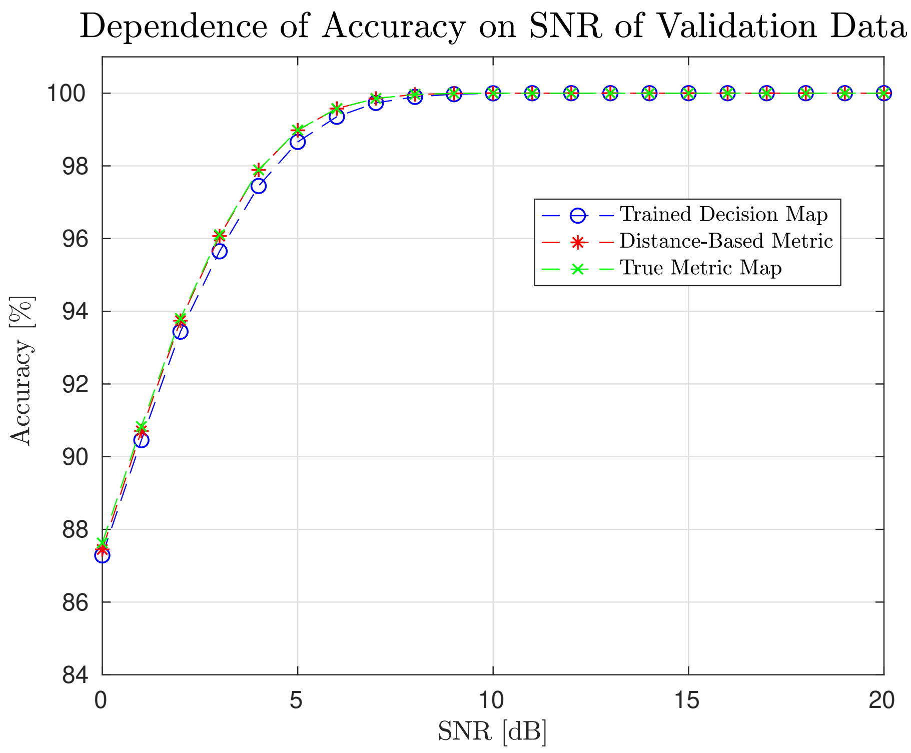


Figure 5.16: Setup 1: Comparison of accuracy of determined maps for $h = \exp\left(\frac{j\pi}{4}\right)$.

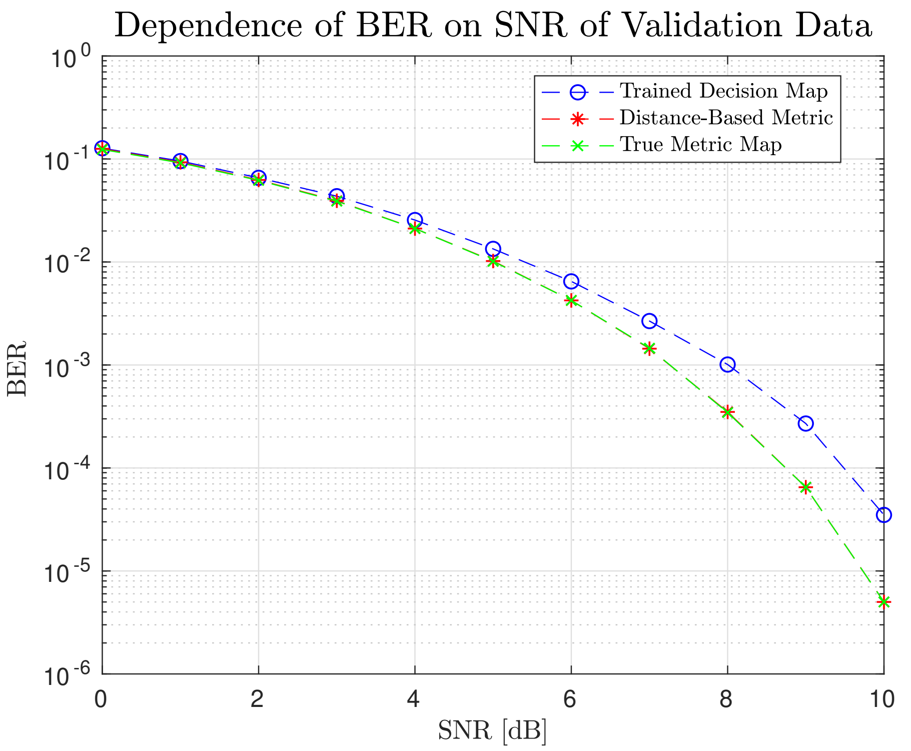


Figure 5.17: Setup 1: Comparison of BER of determined maps for $h = \exp\left(\frac{j\pi}{4}\right)$.

Let us provide a few more examples. Firstly, for value of fading parameter $h = \exp\left(\frac{\mathrm{j}\pi}{2}\right)$, the reference decision map splits constellation space to 4 equal areas. It seems to be suitable for benchmark purposes since we can observe how precisely is the trained ANN able to approximate the central area, where the reference decision boundaries are orthogonal. Of course, better results might be achieved when using larger ANN, an increasing number of training epochs, or when the system was fine-tuned in other ways. Once more, training data set is illustrated in Figure 5.18.

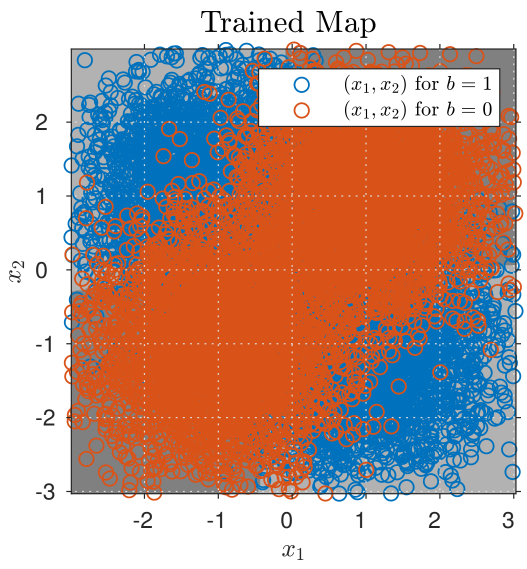


Figure 5.18: Setup 1: Illustration of training data set for $h = \exp\left(\frac{\mathrm{j}\pi}{2}\right)$.

Comparison of reference maps and trained decision map can be seen in Figure 5.19. Again, the approximation is not perfect; however, focusing, e.g., on the upper half-plane of the trained decision map, the decision regions are well-separated along the real and imaginary axis. We can observe that though the setup is symmetric, the trained decision maps are not.

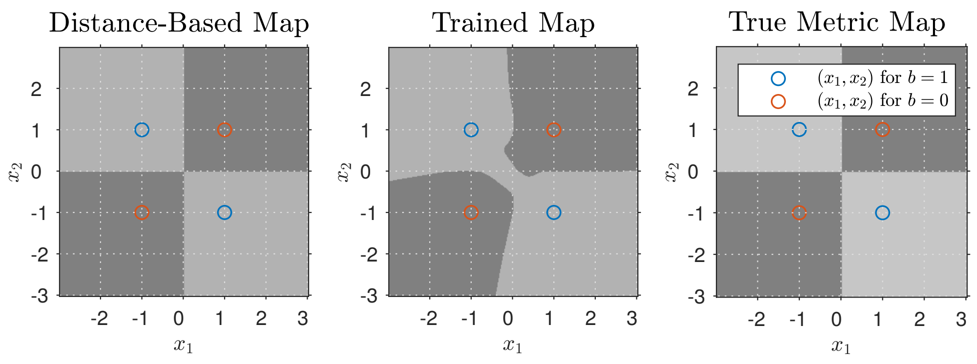


Figure 5.19: Setup 1: Comparison of decision maps for $h = \exp\left(\frac{\mathrm{j}\pi}{2}\right)$.

Comparison of accuracy of reference and trained decision maps over different SNR is once again shown in Figure 5.20 in % and in Figure 5.21 as BER.

Further examples are shown, preserving $|h| = 1$. Specifically, $h = 1$ in Figure 5.22, $h = \exp\left(\frac{3\mathrm{j}\pi}{4}\right)$ in Figure 5.23 and $h = -1$ in Figure 5.24. The figures evaluating performance showed approximately same results and are therefore omitted.

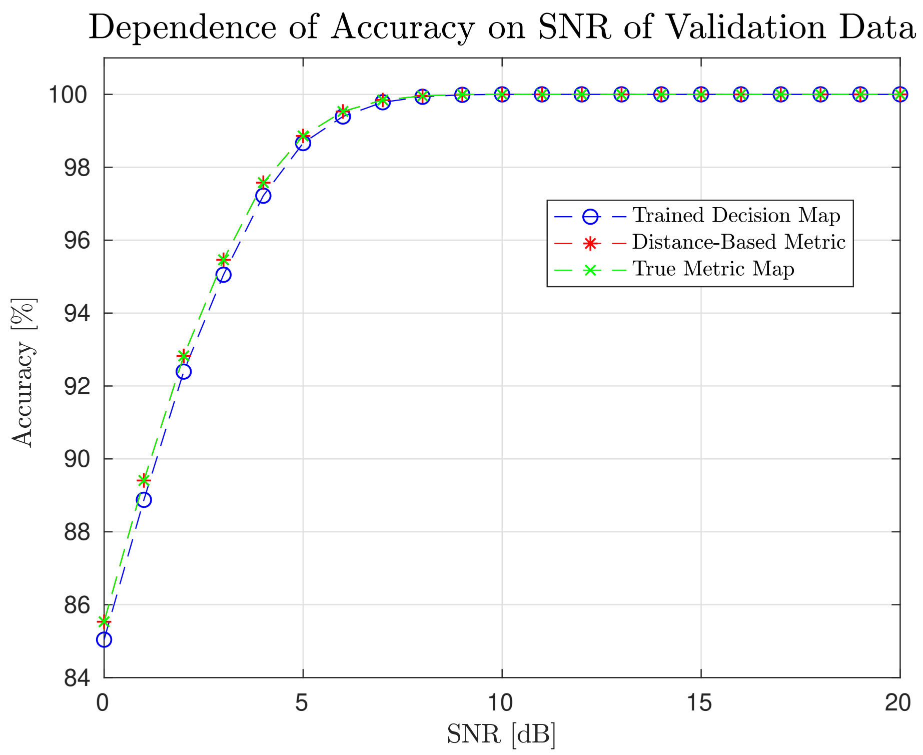


Figure 5.20: Setup 1: Comparison of accuracy of determined maps for $h = \exp\left(\frac{j\pi}{2}\right)$.

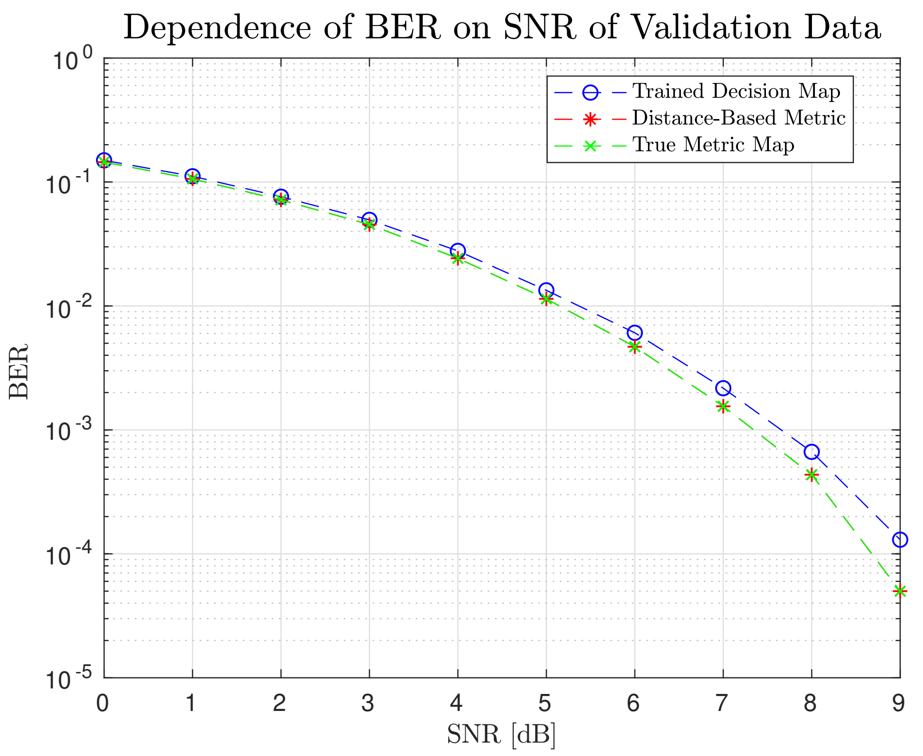


Figure 5.21: Setup 1: Comparison of BER of determined maps for $h = \exp\left(\frac{j\pi}{2}\right)$.

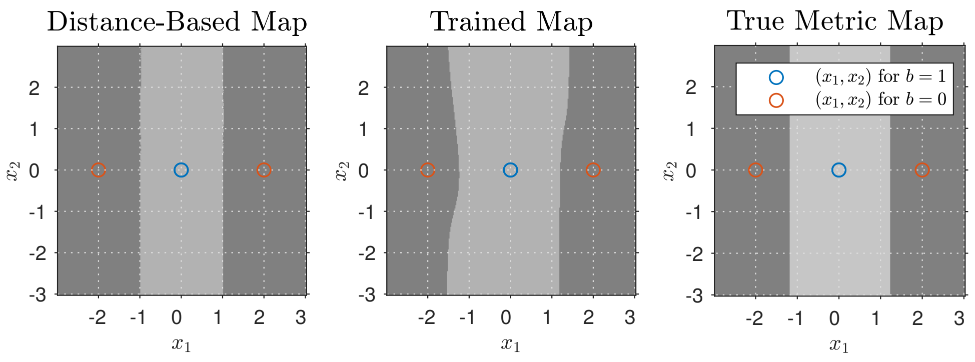


Figure 5.22: Setup 1: Comparison of decision maps for $h = 1$.

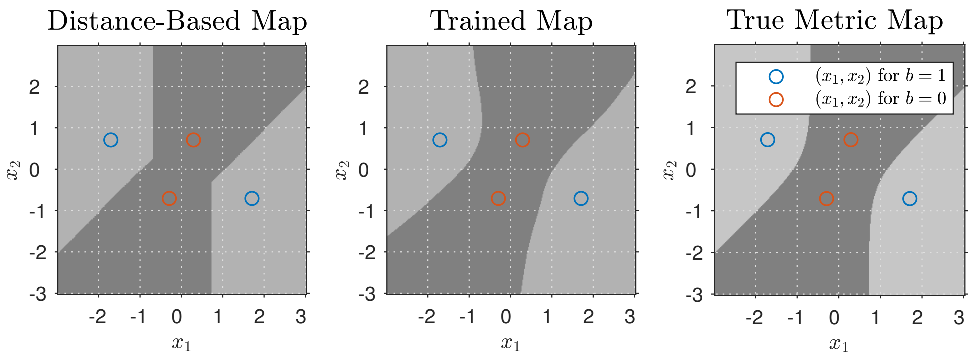


Figure 5.23: Setup 1: Comparison of decision maps for $h = \exp\left(\frac{3j\pi}{4}\right)$.

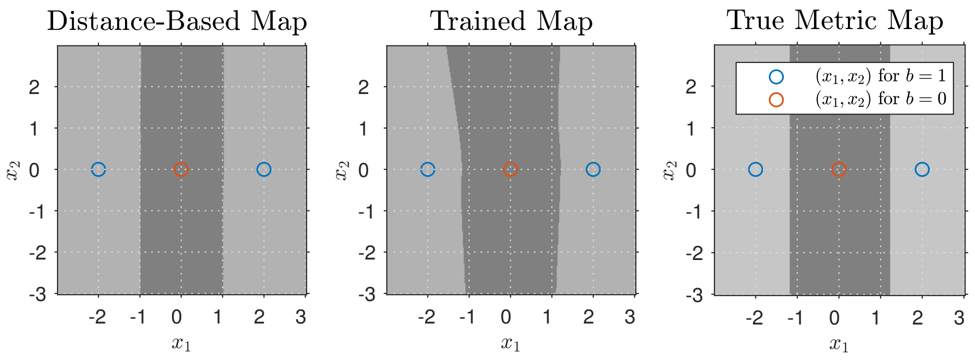


Figure 5.24: Setup 1: Comparison of decision maps for $h = -1$.

Further exemplary results for $|h| = 0,7$ are shown in Figures 5.26–5.29, see the captions for corresponding values of relative fading parameter.

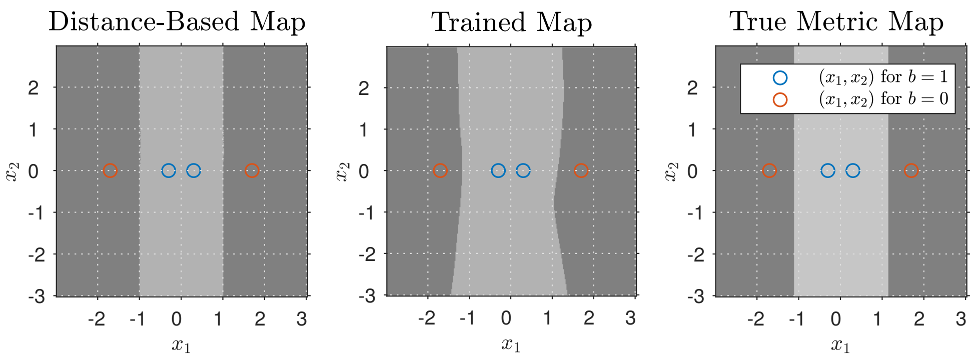


Figure 5.25: Setup 1: Comparison of decision maps for $h = 0,7$.

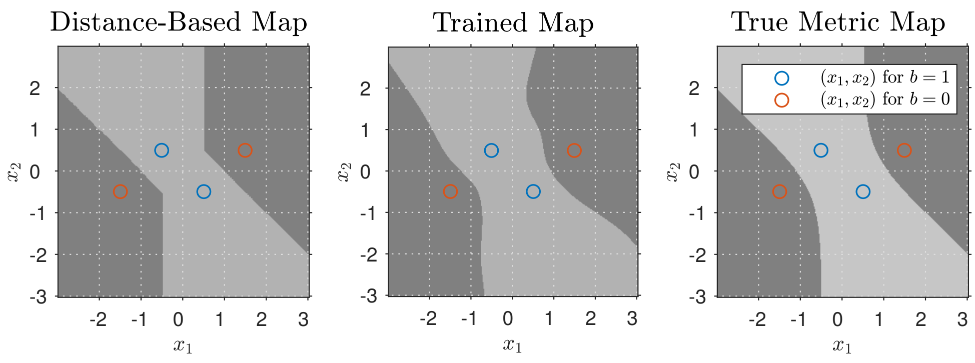


Figure 5.26: Setup 1: Comparison of decision maps for $h = 0,7 \exp\left(\frac{j\pi}{4}\right)$.

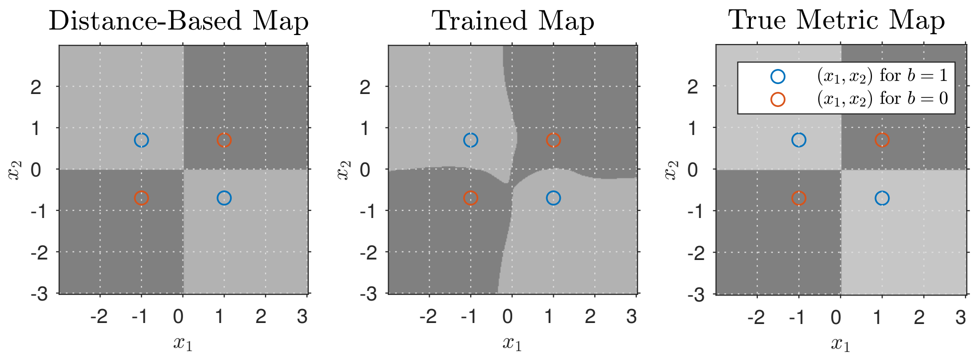


Figure 5.27: Setup 1: Comparison of decision maps for $h = 0,7 \exp\left(\frac{j\pi}{2}\right)$.

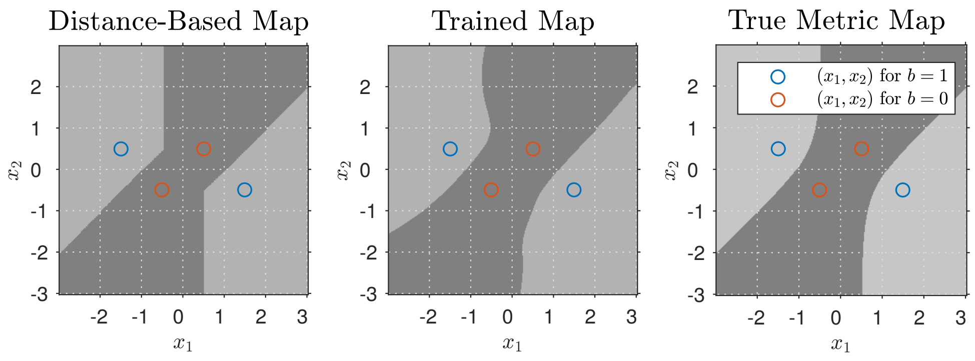


Figure 5.28: Setup 1: Comparison of decision maps for $h = 0,7 \exp\left(\frac{3j\pi}{4}\right)$.

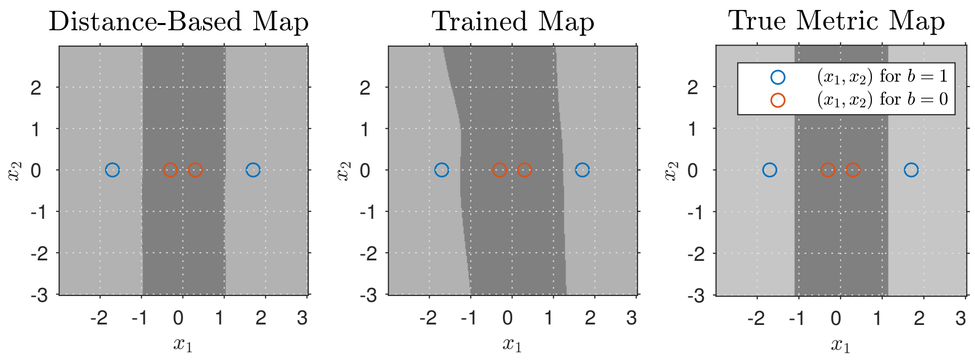


Figure 5.29: Setup 1: Comparison of decision maps for $h = -0,7$.

Setup 2

Parameter	Value
η	0,05
$n^{(1)}_{\text{hidden}}$	70
$n^{(2)}_{\text{hidden}}$	70
SNR of training data set	-1 dB
D	7 000
Number of training epochs	500

Table 5.3: Parameters of simulation in Setup 2.

In Setup 2, all simulations from Setup 1 were repeated with an increased number of neurons in the hidden layers and hereby are provided results for the same relative fading parameters as in Setup 1. I.e., the first half of the simulation results is for $|h| = 1$, the other half for $|h| = 0,7$. We can see that though the number of neurons was significantly increased, the obtained results are of similar performance. (Again, figures showing accuracy of classification and BER are shown only for values of relative fading parameter $h = \exp\left(\frac{\mathrm{j}\pi}{4}\right)$ and $h = \exp\left(\frac{\mathrm{j}\pi}{2}\right)$. Subsequent results were very similar.)

With this repeat of simulations, let us point out that determined parameters are not claimed to be perfect. Hereby, it remains open, what is the optimal number of neurons per layer, how many training epochs are required, and what learning rate is appropriate. Nevertheless, based on the figures comparing BER and accuracy of the trained system, the trained system is shown to perform sufficiently, especially for higher values of SNR.

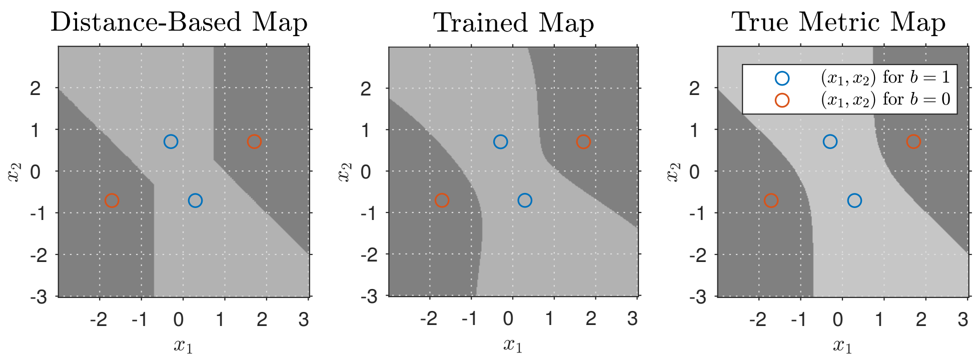


Figure 5.30: Setup 2: Comparison of decision maps for $h = \exp\left(\frac{\mathrm{j}\pi}{4}\right)$.

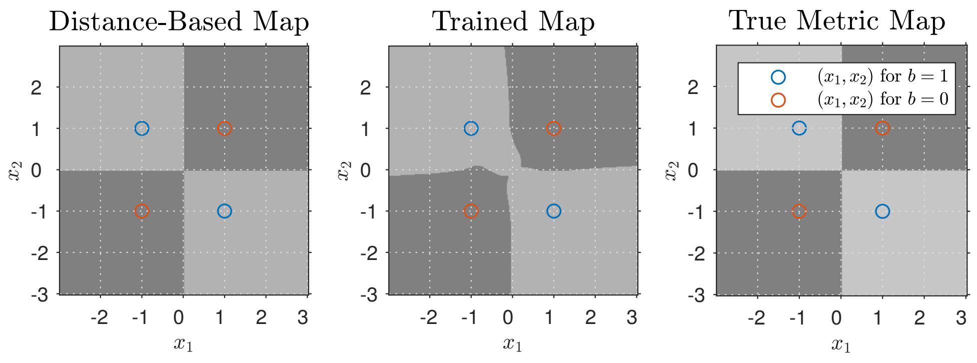


Figure 5.31: Setup 2: Comparison of decision maps for $h = \exp\left(\frac{\mathrm{j}\pi}{2}\right)$.

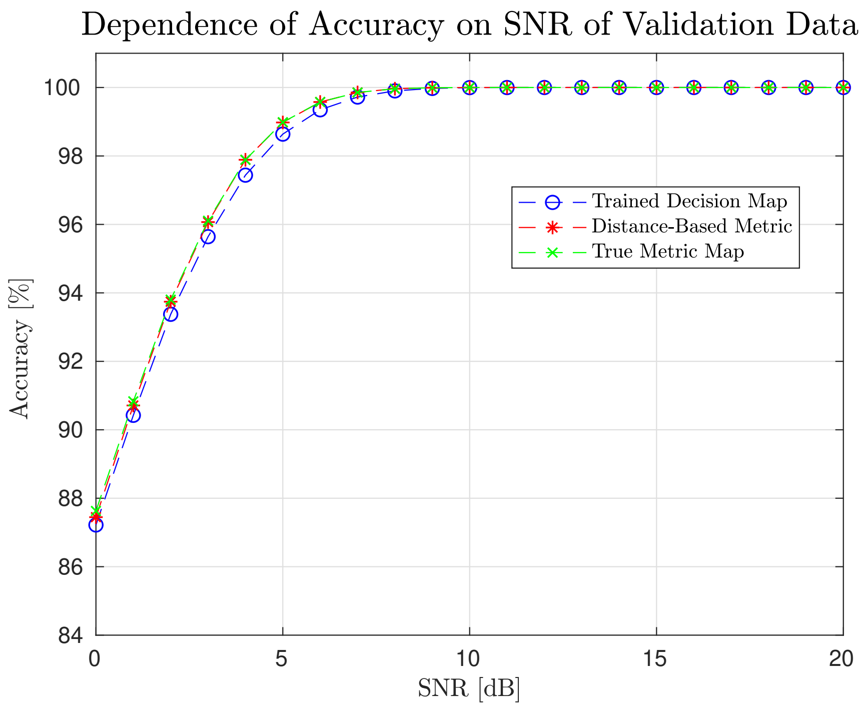


Figure 5.32: Setup 2: Comparison of accuracy of determined maps for $h = \exp\left(\frac{j\pi}{4}\right)$.

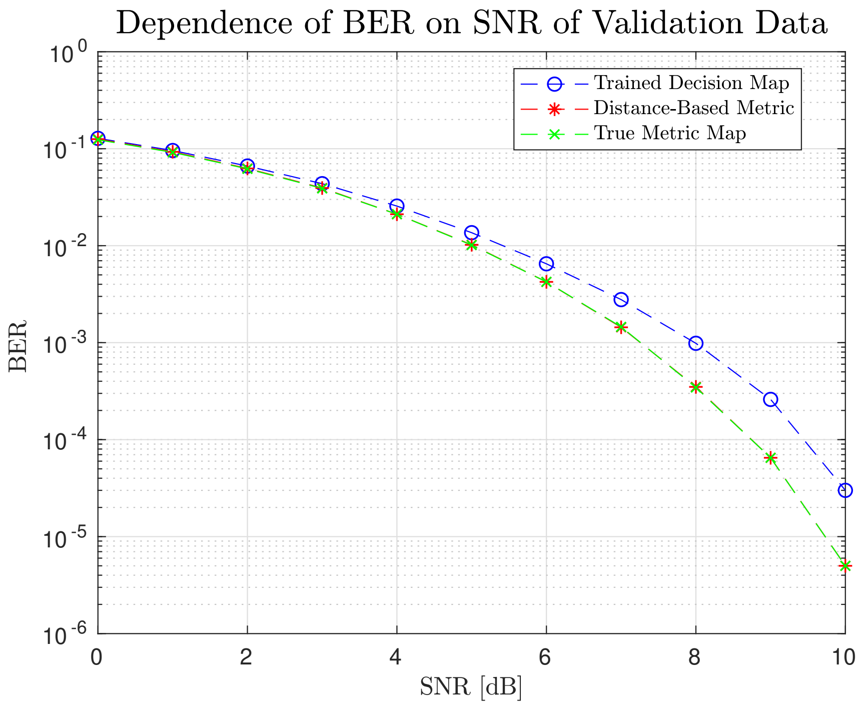


Figure 5.33: Setup 2: Comparison of accuracy of determined maps for $h = \exp\left(\frac{j\pi}{4}\right)$.

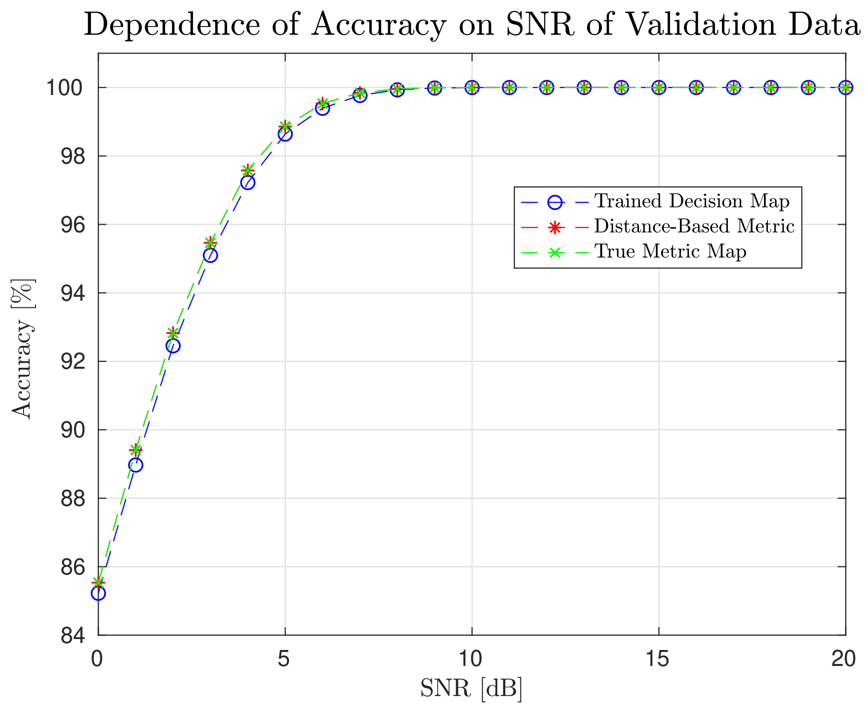


Figure 5.34: Setup 2: Comparison of accuracy of determined maps for $h = \exp\left(\frac{j\pi}{2}\right)$.

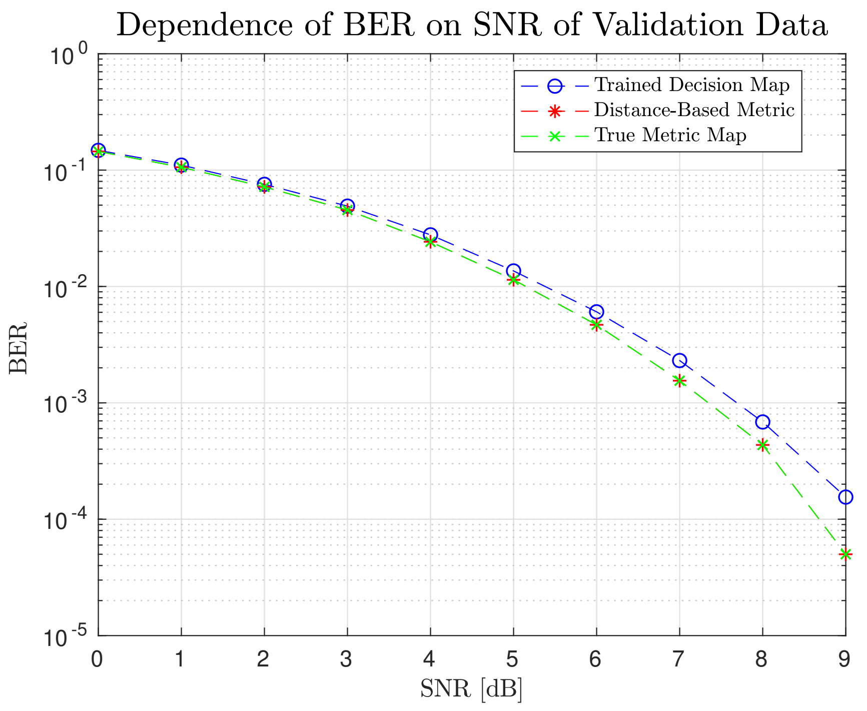


Figure 5.35: Setup 2: Comparison of BER of determined maps for $h = \exp\left(\frac{j\pi}{2}\right)$.

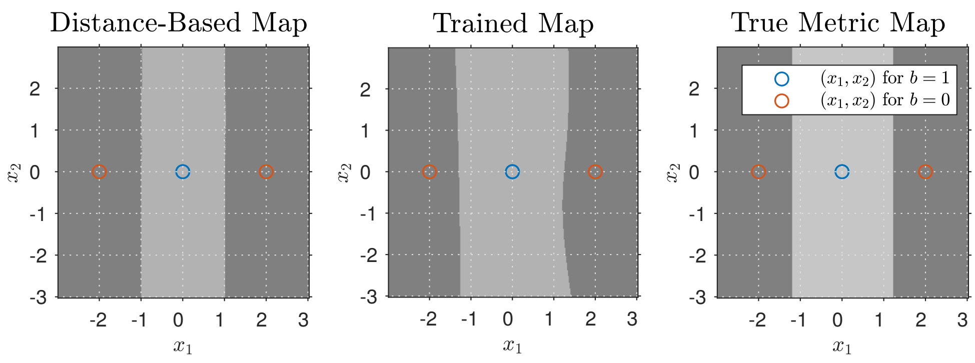


Figure 5.36: Setup 2: Comparison of decision maps for $h = 1$.

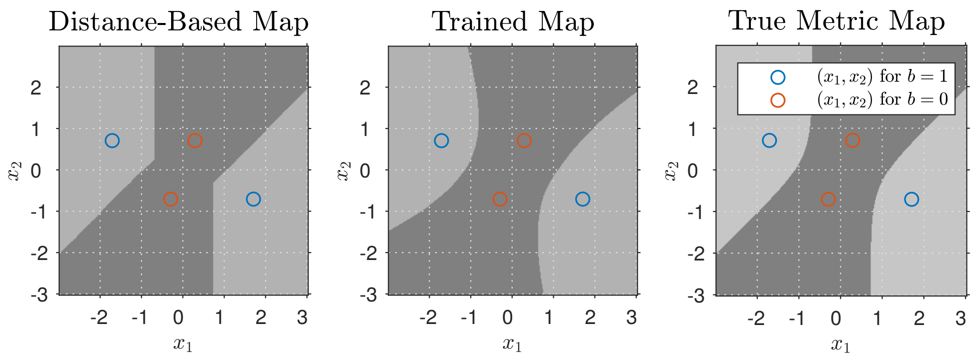


Figure 5.37: Setup 2: Comparison of decision maps for $h = \exp\left(\frac{3j\pi}{4}\right)$.

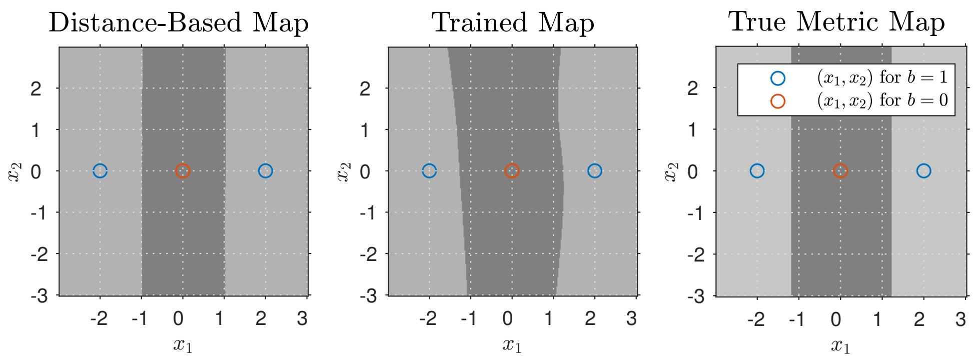


Figure 5.38: Setup 2: Comparison of decision maps for $h = -1$.

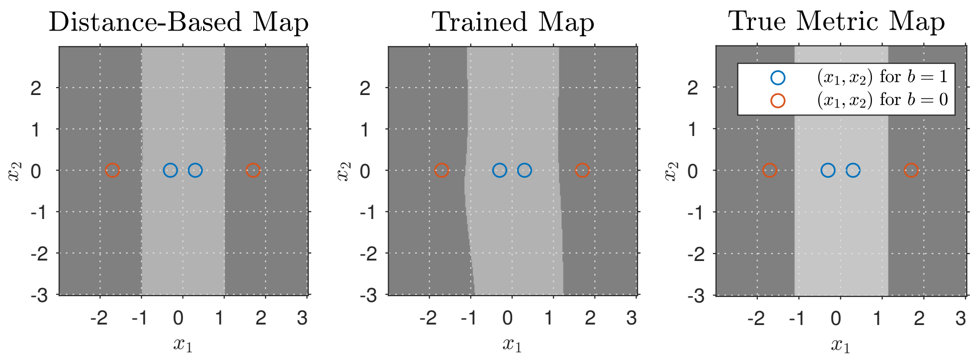


Figure 5.39: Setup 2: Comparison of decision maps for $h = 0,7$.

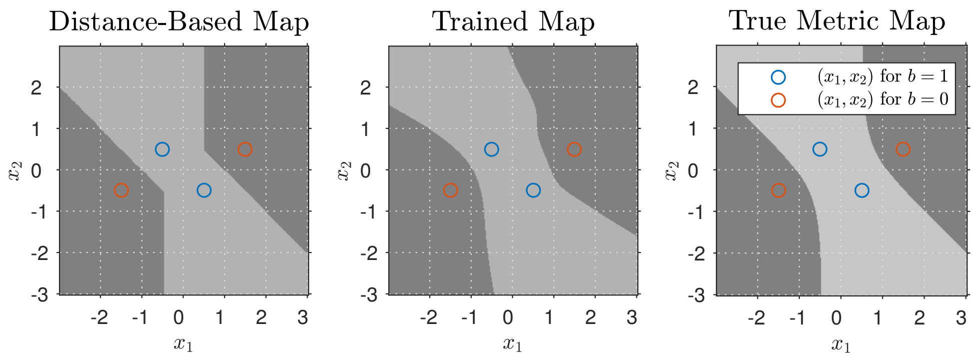


Figure 5.40: Setup 2: Comparison of decision maps for $h = 0,7 \exp\left(\frac{j\pi}{4}\right)$.

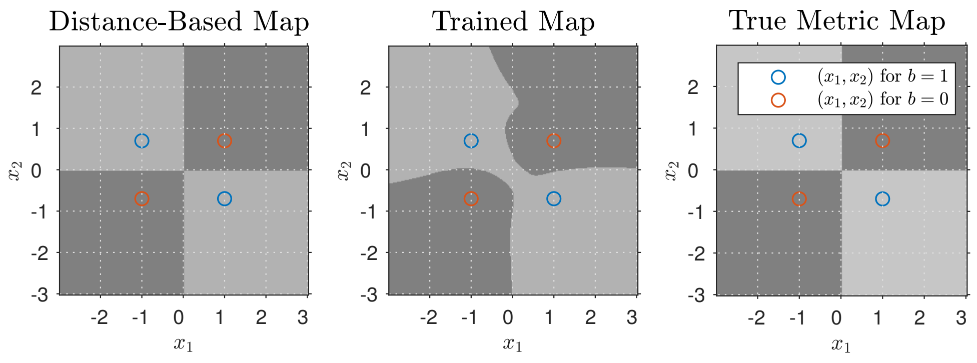


Figure 5.41: Setup 2: Comparison of decision maps for $h = 0,7 \exp\left(\frac{j\pi}{2}\right)$.

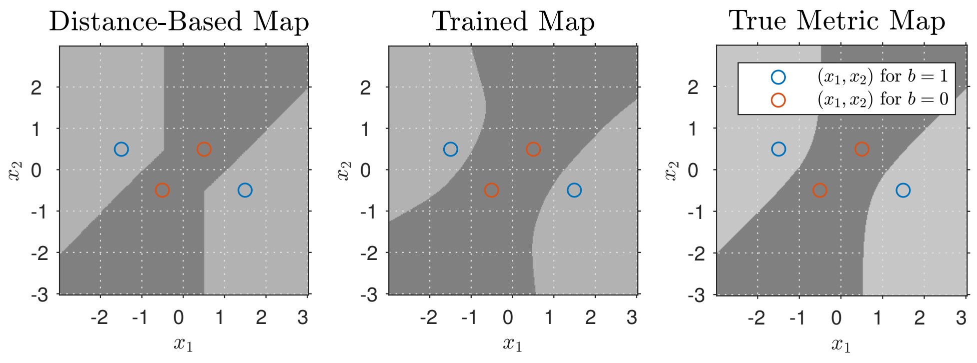


Figure 5.42: Setup 2: Comparison of decision maps for $h = 0,7 \exp\left(\frac{3j\pi}{4}\right)$.

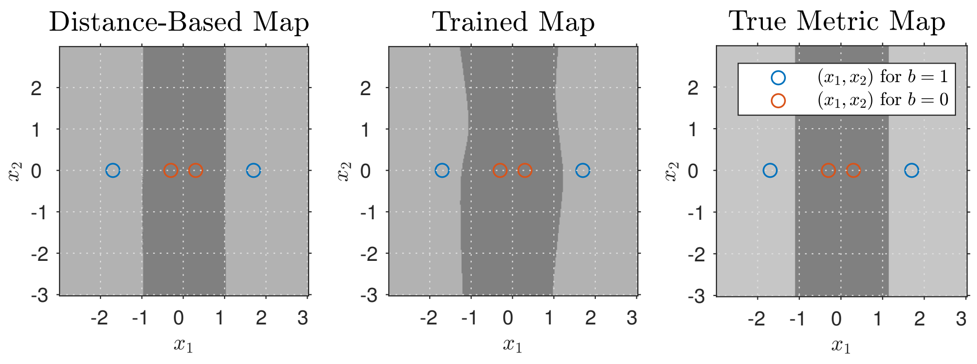


Figure 5.43: Setup 2: Comparison of decision maps for $h = -0,7$.

5.4 Classification of Hierarchical Symbols in simplified 2WRC with QPSK

5.4.1 Extension to QPSK

Further, let us consider an extension of procedure described in Section 5.3 purely for BPSK modulation. We shall consider QPSK modulation with XOR HNC map in 2WRC. Therefore, the observation model is still described by Equations 5.3 and 5.4, but implemented bit-wise. In accordance with [19], the symbols of signle QPSK are assigned according to Table 5.4.

Information Input	Mapping
00	$\mathrm{e}^{\frac{\mathrm{j}\pi}{4}}$
01	$\mathrm{e}^{\frac{3\mathrm{j}\pi}{4}}$
10	$\mathrm{e}^{-\frac{3\mathrm{j}\pi}{4}}$
11	$\mathrm{e}^{-\frac{\mathrm{j}\pi}{4}}$

Table 5.4: Implemented single-user QPSK.

A direct extension of Table 5.1 with corresponding information inputs and training inputs is shown in Table 5.5. Again, four possible hierarchical symbols are indicated by four outputs of the ANN. Hence, $n_{\text{out}} = 4$.

$\boldsymbol{b}_{A,d}$	$\boldsymbol{b}_{B,d}$	$\boldsymbol{b}_d$	$\boldsymbol{t}_d = (t_1, t_2, t_3, t_4)_d$
00	00	00	(1, 0, 0, 0)
00	01	01	(0, 1, 0, 0)
00	10	10	(0, 0, 1, 0)
00	11	11	(0, 0, 0, 1)
01	00	01	(0, 1, 0, 0)
01	01	00	(1, 0, 0, 0)
01	10	11	(0, 0, 0, 1)
01	11	10	(0, 0, 1, 0)
10	00	10	(0, 0, 1, 0)
10	01	11	(0, 0, 0, 1)
10	10	00	(1, 0, 0, 0)
10	11	01	(0, 1, 0, 0)
11	00	11	(0, 0, 0, 1)
11	01	10	(0, 0, 1, 0)
11	10	01	(0, 1, 0, 0)
11	11	00	(1, 0, 0, 0)

Table 5.5: Explanation of training input $\boldsymbol{t_d}$.

Outputs of ANN implemented in Section 5.3 were simply rounded, to indicate the determined hierarchical symbols. With four hierarchical symbols, the determined hierarchical symbol shall be now identified by the maximal value of the corresponding output. This eases graphical representation of decision maps and avoids conflicts in classification.

5.4.2 Exemplary Results

Parameters of the training process were experimentally tuned to reach satisfactory results. Similarly to previous results with BPSK modulation, hereby, we present results for several values of relative fading parameter h. As expected, we shall see that the shape of the decision maps becomes more complicated than previously.

Note that for comparisons of accuracy, we used the validation set of 200 000 samples for each value of SNR. Compared to previous results for BPSK, hereby, we used the term *symbol error rate* (SER) to evaluate the error rate of classification, since the symbol consists of two bits.

Besides an increased number of hierarchical symbols, another issue with QPSK is, that of *singular fading,* described in [19]. It is a situation when the specific value of relative fading parameter causes, that multiple different hierarchical symbols are in constellation space located in the same coordinates. Later, one example is shown to demonstrate this.

✳	(x_1, x_2) for $b = 00$
×	(x_1, x_2) for $b = 01$
□	(x_1, x_2) for $b = 10$
○	(x_1, x_2) for $b = 11$

Figure 5.44: QPSK simulation legend.

Keeping the following results for QPSK modulation more clear, the legend is shown separately in Figure 5.44. Further are shown simulation results generated with parameters shown in Table 5.6. During experimental simulations was observed that good results are shown for SNR of training data set 11 dB.

Parameter	Value
η	0,05
$n_{\text{hidden}}^{(1)}$	50
$n_{\text{hidden}}^{(2)}$	50
SNR of training data set	11 dB
D	10 000
Number of training epochs	1 000

Table 5.6: Parameters of simulation for QPSK.

To provide an orientation, in Figure 5.45 it is shown training data set together with trained map for relative fading parameter $h = 1$. The training samples are distinguished according to legend in Figure 5.44.

This task is more demanding than the example with BPSK. In this setting, only a hierarchical symbol $b = 11$ is located in a single region. Regions corresponding to hierarchical symbols $b = 01$ and $b = 10$ are divided into two parts; finally hierarchical symbol $b = 00$ is divided into four parts. Note that this uneven number of regions, corresponding to specific hierarchical symbols, implies that individual regions are covered with an uneven number of training samples.

Specifically, about four times more training samples are located in the single central region for $b = 11$, than in the four individual regions for $b = 00$.

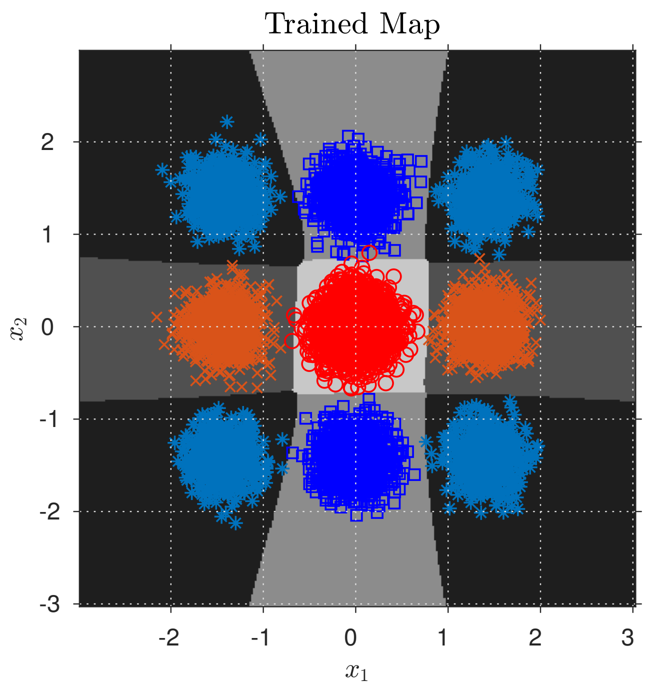


Figure 5.45: Illustration of training data set for $h = 1$.

Further, reference decision maps are together with the trained decision map shown in Figure 5.46. Clearly, the individual regions are separated. The decision boundaries are not as straight as supposed to be. The trained corner regions, corresponding to symbol $b = 00$, were probably trained smaller than desired because the number of training samples in the neighboring regions was bigger. Also, we can observe that for SNR = 11 dB, the difference between Distance-Based Map and True Metric Map is visually negligible.

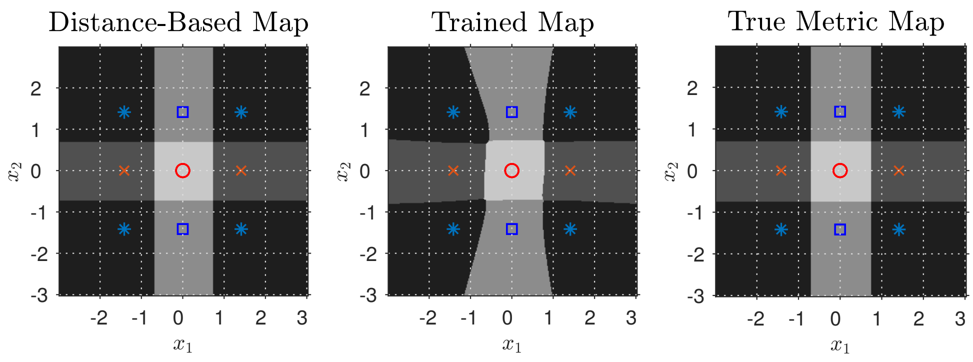


Figure 5.46: Comparison of decision maps for $h = 1$.

To quantitatively evaluate the performance of the trained system, we again use a validation data set for different values of SNR. In Figure 5.47, it is expressed as accuracy of correct classifications. We can see that accuracy rapidly drops off for SNR below 10 dB. Further, we observe, that below 5 dB Trained Decision Map performs better than Distance-Based Map.

In Figure 5.48, the performance is evaluated as symbol error rate in logarithmic scale. At this moment, we can see that around SNR = 11 dB, the trained system performs worse than the reference decision maps.

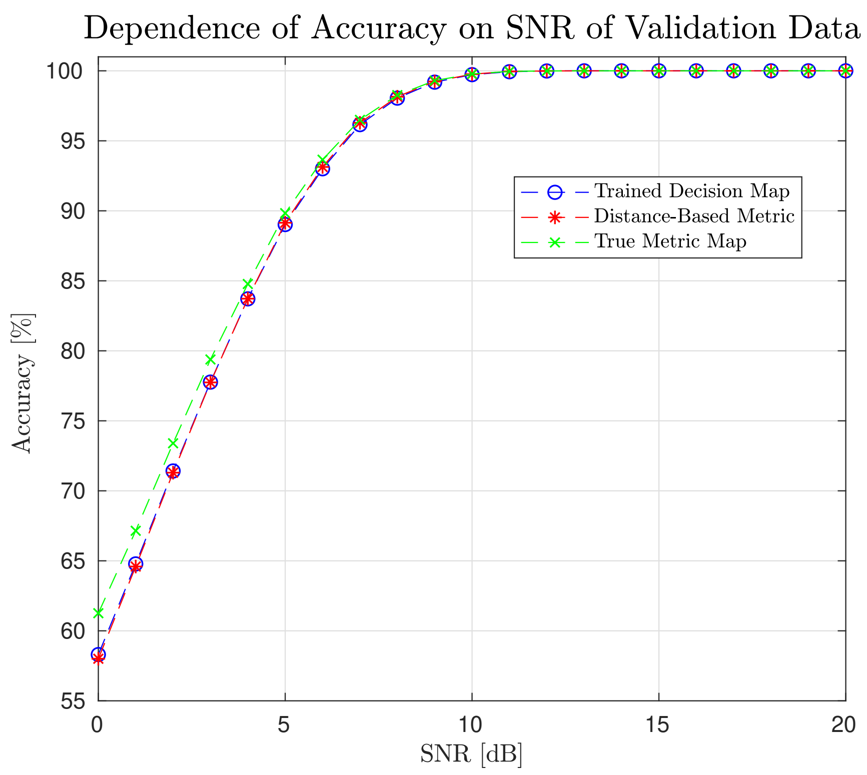


Figure 5.47: Comparison of accuracy of determined maps for $h = 1$.

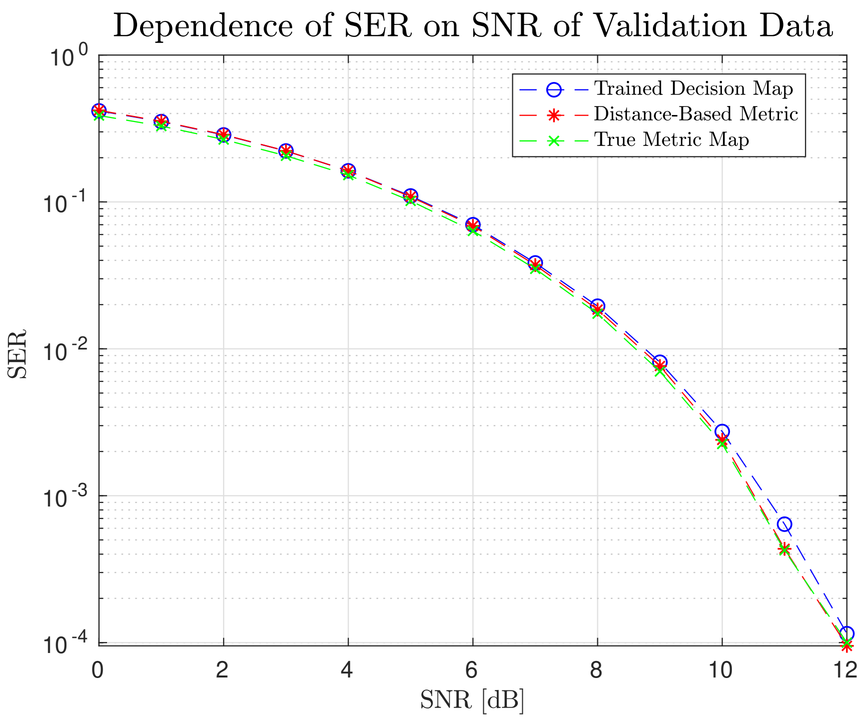


Figure 5.48: Comparison of SER of determined maps for $h = 1$.

Further testing of this ANN configuration was performed for different values of relative fading parameter h. First part of results shows situation when $|h| = 1$. Subsequent result is shown for $h = \exp\left(\frac{j\pi}{6}\right)$. In Figure 5.49, we can again see training data set on top of trained map.

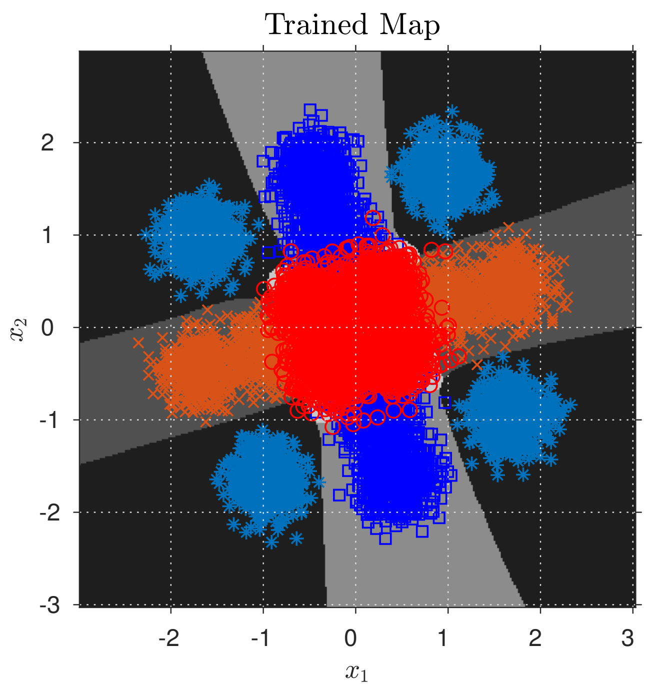


Figure 5.49: Illustration of training data set for $h = \exp\left(\frac{j\pi}{6}\right)$.

In Figure 5.50, we can clearly compare the trained and the reference decision maps. We observe that the central cross-like region, corresponding to hierarchical symbol $b = 11$ is roughly followed by the ANN. The shape of the decision regions is more complex than previously; however, the number of regions is still preserved. A fact is that at this moment, the shape of trained decision boundaries does not follow the reference solutions very well.

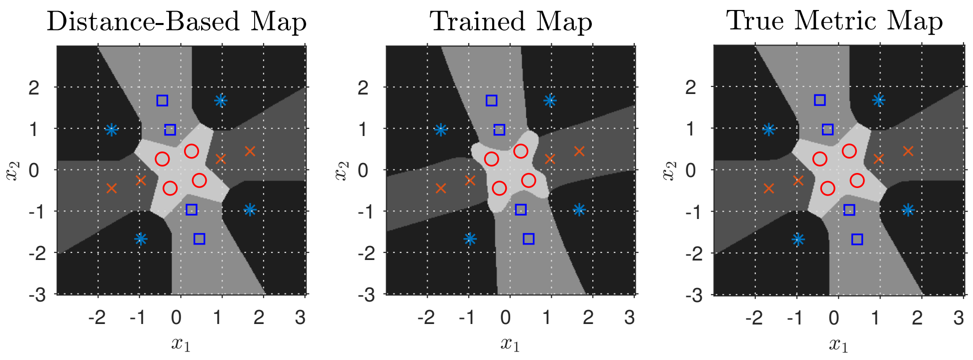


Figure 5.50: Comparison of decision maps for $h = \exp\left(\frac{j\pi}{6}\right)$.

Quantitatively, it is evaluated in Figure 5.51. A rapid drop off of accuracy of classification is observed for the trained system at SNR=15 dB. A poor performance is even emphasized in Figure 5.52, where the symbol error rate is evaluated. Comparing Figures 5.48 and 5.52, we demonstrate a severe dependence of performance of the trained system on actual value of relative fading parameter. This holds for all methods.

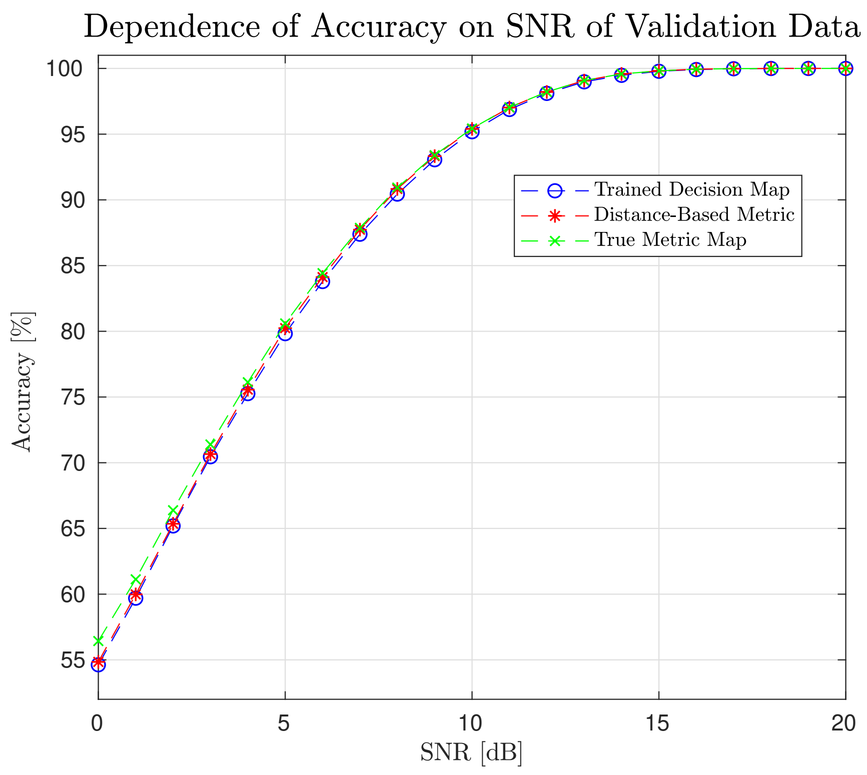


Figure 5.51: Comparison of accuracy of determined maps for $h = \exp\left(\frac{j\pi}{6}\right)$.

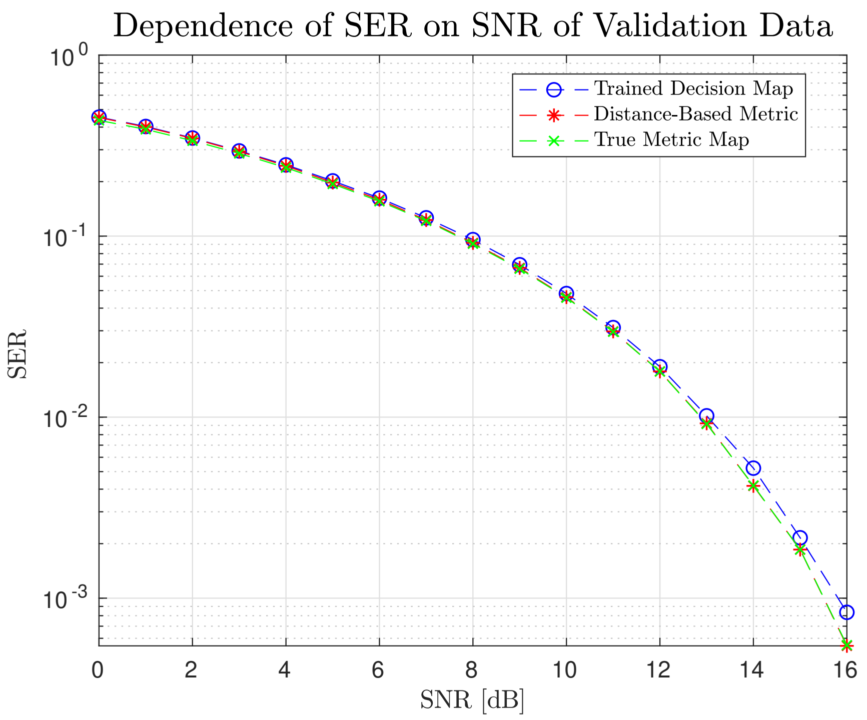


Figure 5.52: Comparison of SER of determined maps for $h = \exp\left(\frac{j\pi}{6}\right)$.

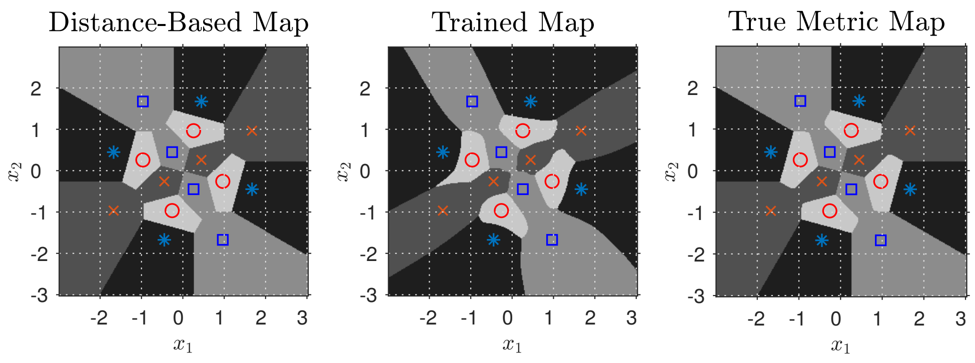


Figure 5.53: Comparison of decision maps for $h = \exp\left(\frac{2j\pi}{6}\right)$.

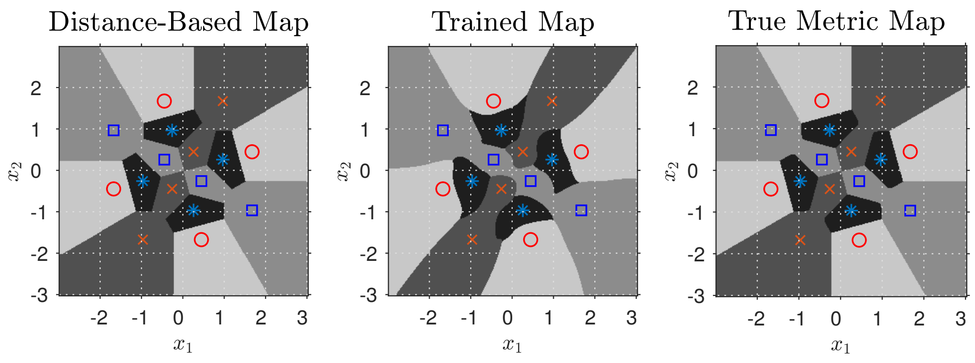


Figure 5.54: Comparison of decision maps for $h = \exp\left(\frac{4j\pi}{6}\right)$.

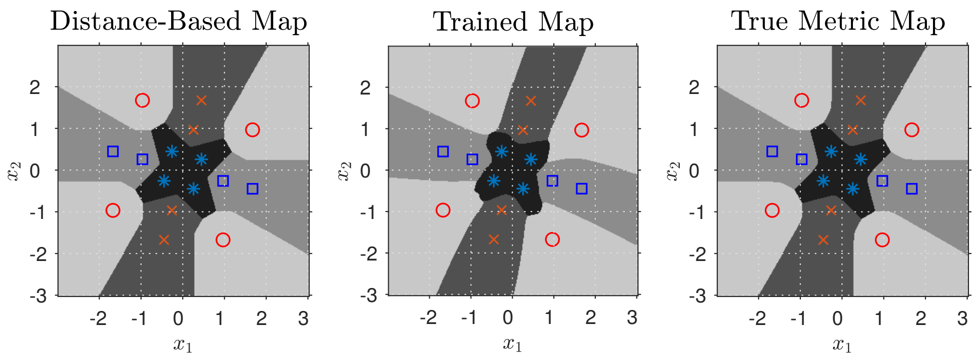


Figure 5.55: Comparison of decision maps for $h = \exp\left(\frac{5j\pi}{6}\right)$.

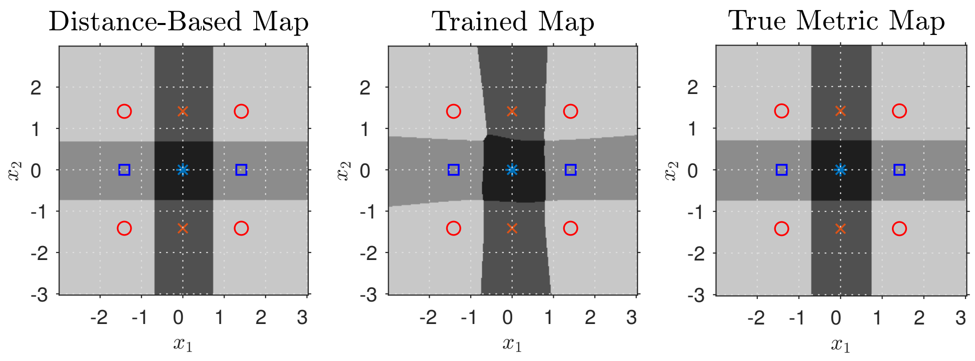


Figure 5.56: Comparison of decision maps for $h = -1$.

In Figures 5.53 to 5.56, further results of training are shown for $|h| = 1$. See captions of the figures for corresponding values of h. In Figures 5.53 and 5.54, we can see, that shape of the decision regions is further complicated, because the central section splits to four parts. Evaluation of accuracy and SER is omitted and was similar to the previous case.

In Figure 5.57 is demonstrated a situation of singular fading, addressed in [19]. As an example, for a value of relative fading parameter $h = \exp\left(\frac{j\pi}{2}\right)$ it happens, that coordinates of multiple different hierarchical symbols in constellation space are identical. Note, that in Figure 5.57 some of the markers, distinguishing individual hierarchical symbols overlap and effectively prevent correct classification.

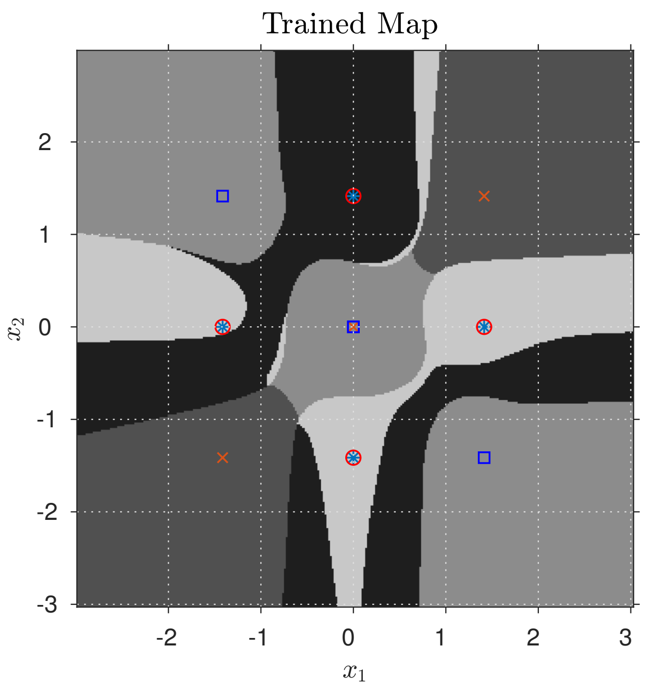


Figure 5.57: Illustration of training data set for $h = \exp\left(\frac{j\pi}{2}\right)$.

The following examples are shown for $|h| = 0,7$ for the same angles of h as previously. Only in Figure 5.58 are shown training data to give an illustrative example. With the decreased magnitude of the fading parameter, each hierarchical symbol is represented in a constellation space with four possible coordinates, and this makes the training further complicated.

In Figure 5.59 are decision maps for $h = 0,7$ and in Figure 5.60 for $h = 0,7 \cdot \exp\left(\frac{j\pi}{6}\right)$. The graphs, evaluating performance of the maps are then shown in Figures 5.61 and 5.62 for $h = 0,7$ and in Figures 5.63 and 5.64 for $h = 0,7 \cdot \exp\left(\frac{j\pi}{6}\right)$.

It is worth notice, that for $h = 0,7 \cdot \exp\left(\frac{j\pi}{6}\right)$, the performance is very poor for all used decision maps. Finally, in Figures 5.65 to 5.68 are shown additional example for $|h| = 0,7$. We observe, that the trained maps are not symmetric.

To conclude this section, an extension of the simulation from BPSK to QPSK was quite straightforward. However, it is not clear what parameters of ANN result in optimal performance. Some results were experimentally determined and hereby are provided. At this moment, we do not claim that the reached results of the ANN trained system are optimal. In most examples, the performance of the trained system falls behind the reference methods for higher values of SNR. On the other side, it sometimes slightly overcomes the Distance-Based Maps for lower values of SNR. Performance of True Metric Map, derived in [19], was shown to be overall superior in the simulations.

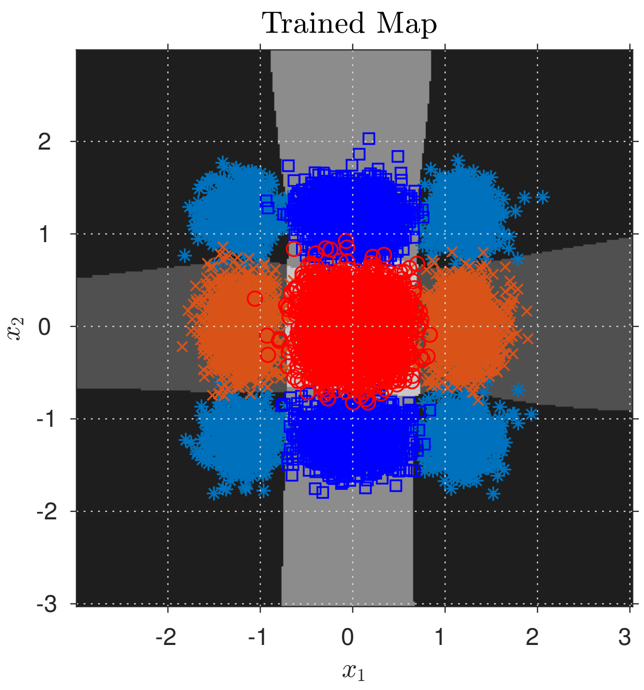


Figure 5.58: Illustration of training data set for $h = 0,7$.

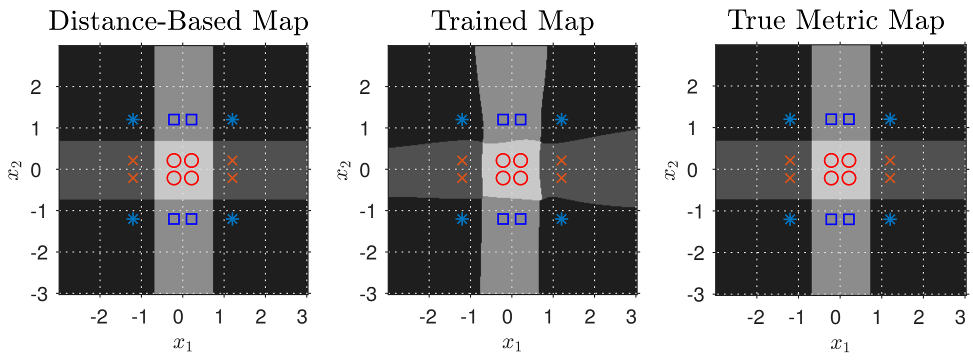


Figure 5.59: Comparison of decision maps for $h = 0,7$.

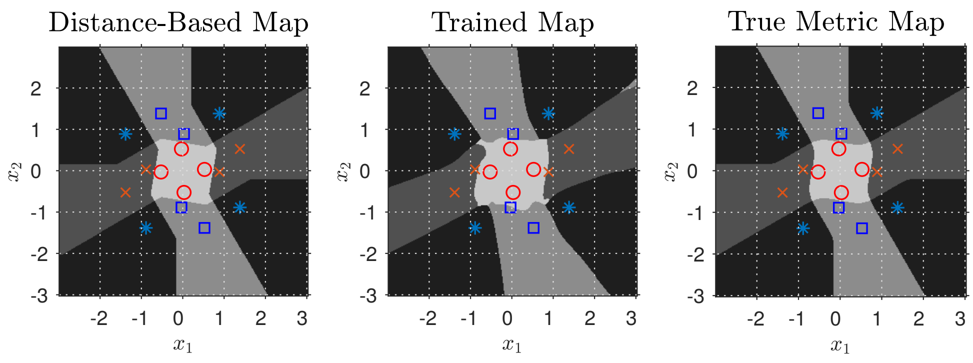


Figure 5.60: Comparison of decision maps for $h = 0,7 \cdot \exp\left(\frac{\mathrm{j}\pi}{6}\right)$.

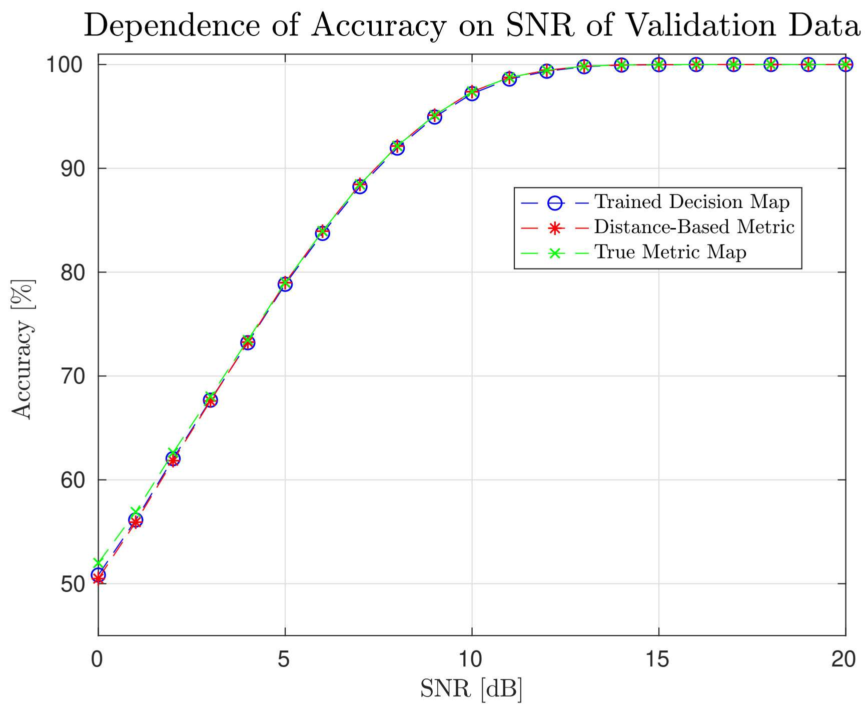


Figure 5.61: Comparison of accuracy of determined maps for $h = 0, 7$.

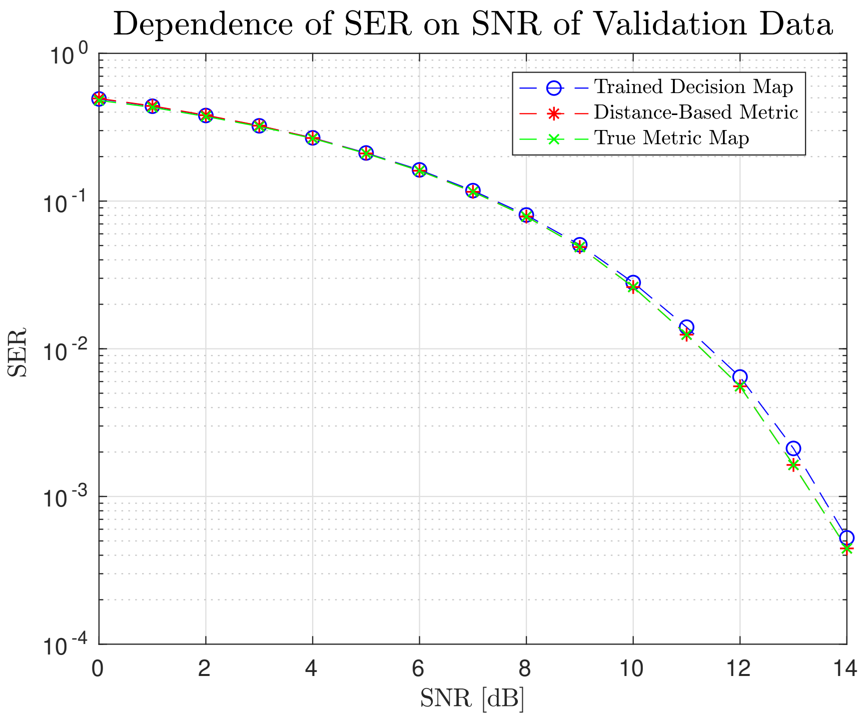


Figure 5.62: Comparison of SER of determined maps for $h = 0, 7$.

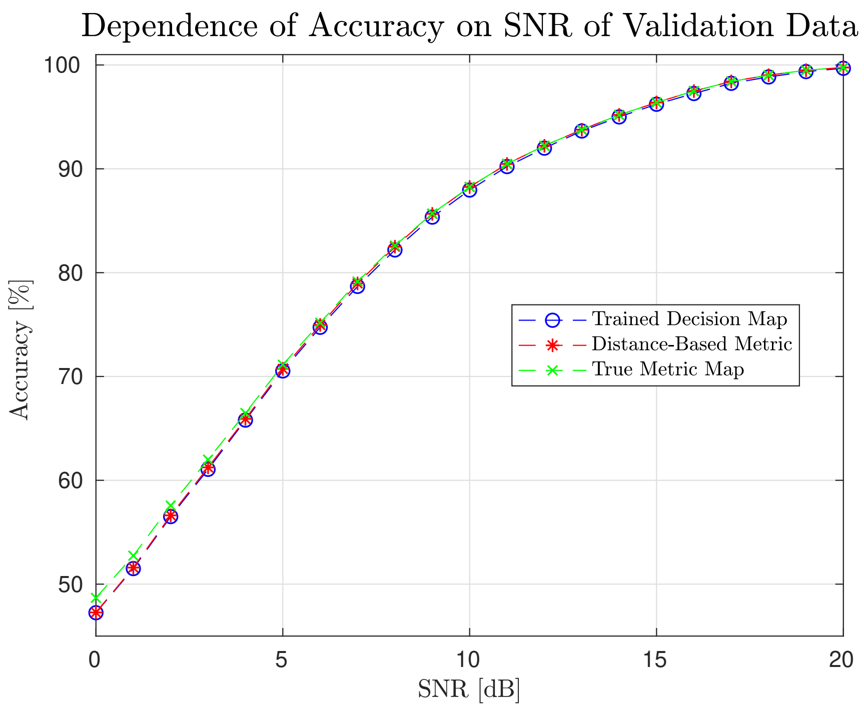


Figure 5.63: Comparison of accuracy of determined maps for $h = 0,7 \cdot \exp\left(\frac{j\pi}{6}\right)$.

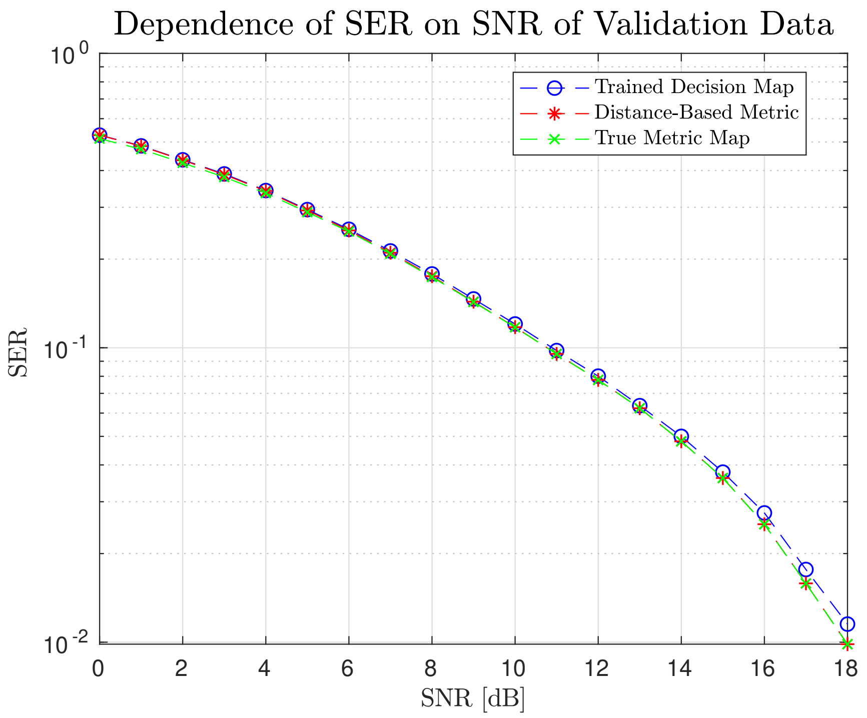


Figure 5.64: Comparison of SER of determined maps for $h = 0,7 \cdot \exp\left(\frac{j\pi}{6}\right)$.

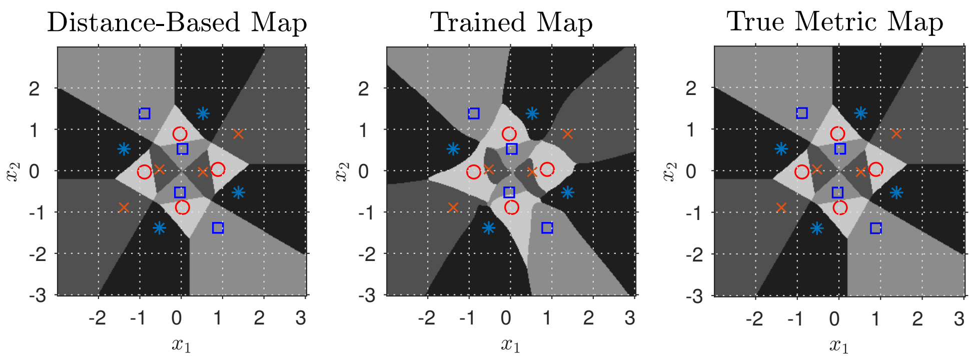


Figure 5.65: Comparison of decision maps for $h = 0,7 \cdot \exp\left(\frac{2j\pi}{6}\right)$.

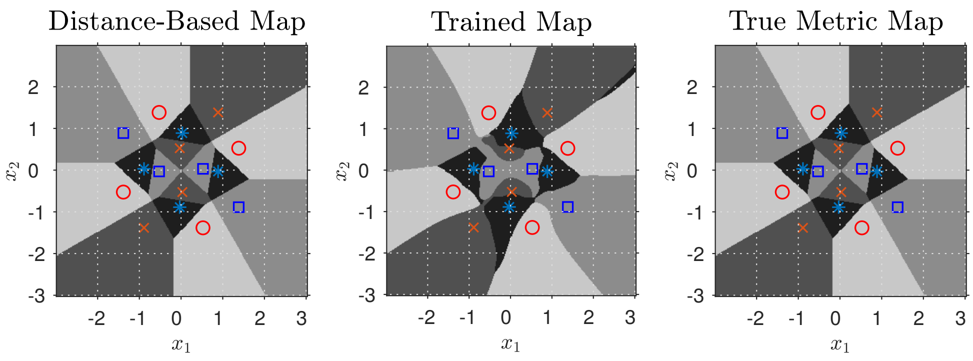


Figure 5.66: Comparison of decision maps for $h = 0,7 \cdot \exp\left(\frac{4j\pi}{6}\right)$.

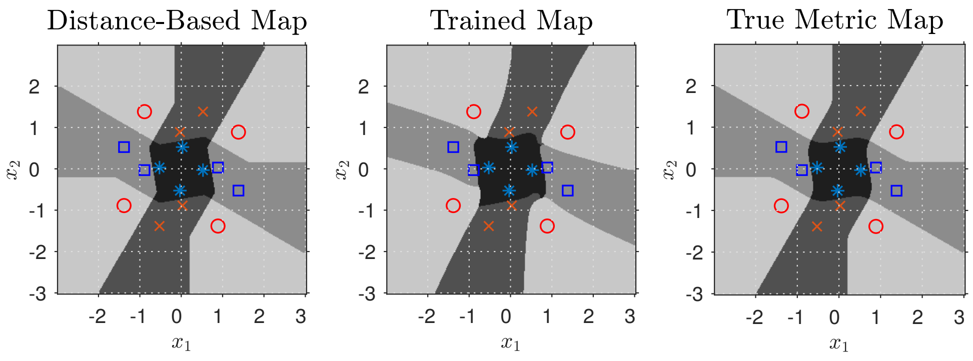


Figure 5.67: Comparison of decision maps for $h = 0,7 \cdot \exp\left(\frac{5j\pi}{6}\right)$.

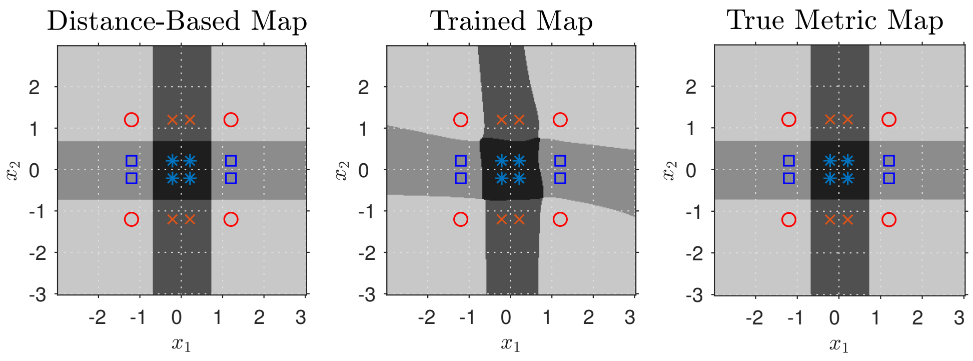


Figure 5.68: Comparison of decision maps for $h = -0,7$.

5.5 Effect of Size of Training Data Set on Classification in 2WRC with BPSK

In this section, we experimentally addressed the influence of the size of the training data set D on the performance of the trained system. Analytical tools for this task were not found in the literature. Motivation is to reduce the number of resources required for training.

Besides the variable value of D, the tested ANN was designed with parameters of Setup 1 in Section 5.3, and for convenience repeated in Table 5.7. We focused on a single value of relative fading parameter, $h = \exp\left(\frac{\mathrm{j}\pi}{4}\right)$.

Parameter	Value
η	0,05
$n_{\text{hidden}}^{(1)}$	40
$n_{\text{hidden}}^{(2)}$	40
SNR of training data set	-1 dB
Number of training epochs	500

Table 5.7: Parameters of simulation.

The simulations were organized, such that a value of D ranged in 20 values between 100 and 3 900 samples with a uniform step of 200 samples. For each of these 20 values, 20 ANNs were trained and tested. These tests were performed for 3 different levels of SNR. The size of the validation data set was 100 000 samples. Resulting values of Accuracy and BER were averaged and are shown in Figures 5.69 and 5.70 for SNR = 5 dB; in Figures 5.71 and 5.72 for SNR = 7,5 dB; and in Figures 5.73 and 5.74 for SNR = 10 dB. Moreover, in the figures of accuracy are shown polynomials of the fifth order, fitting the trained results.

Looking at these figures, we observe that for all values of SNR, a rapid drop-off of performance occurs for the particular tested ANN below $D = 2\,000$ samples. We observe that the SNR of validation data does not affect this value.

The presented simulation is significantly time-consuming. Though the results were averaged over 20 realizations, none of the curves that evaluate the performance of the trained decision maps seem to be smooth. This probably means that more realizations of the experiment would have been desirable. Another issue is that these results are valid only for this specific parameter of ANN. Moreover, to avoid disturbance of the experiment by other parameters, the results are shown only for a single value of h. A similar experiment but on a larger scale might be performed in case of tuning of the parameters for some more practical utilization.

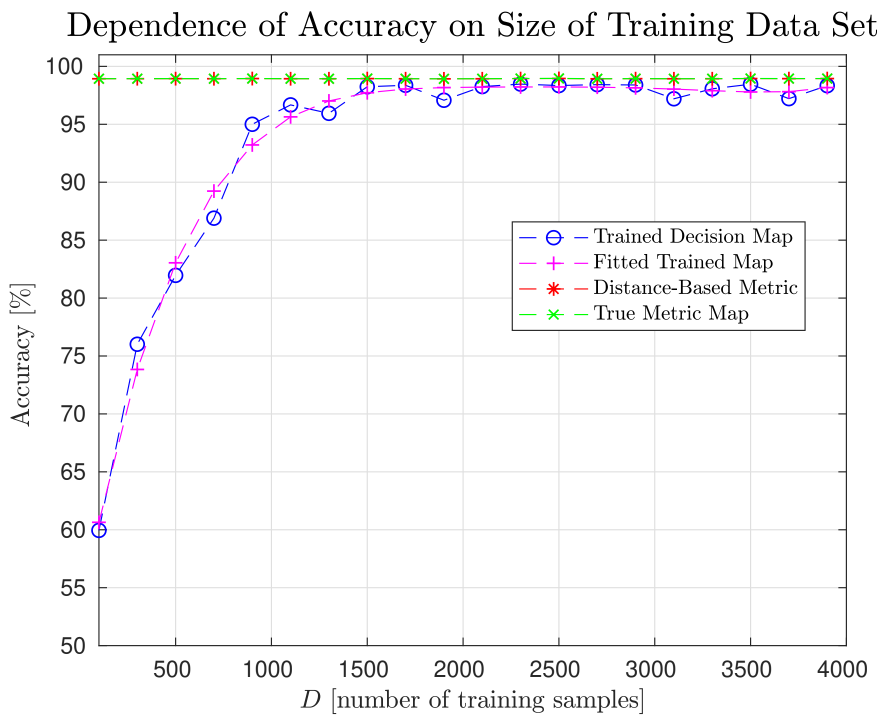


Figure 5.69: Comparison of accuracy for different values of D for SNR = 5 dB.

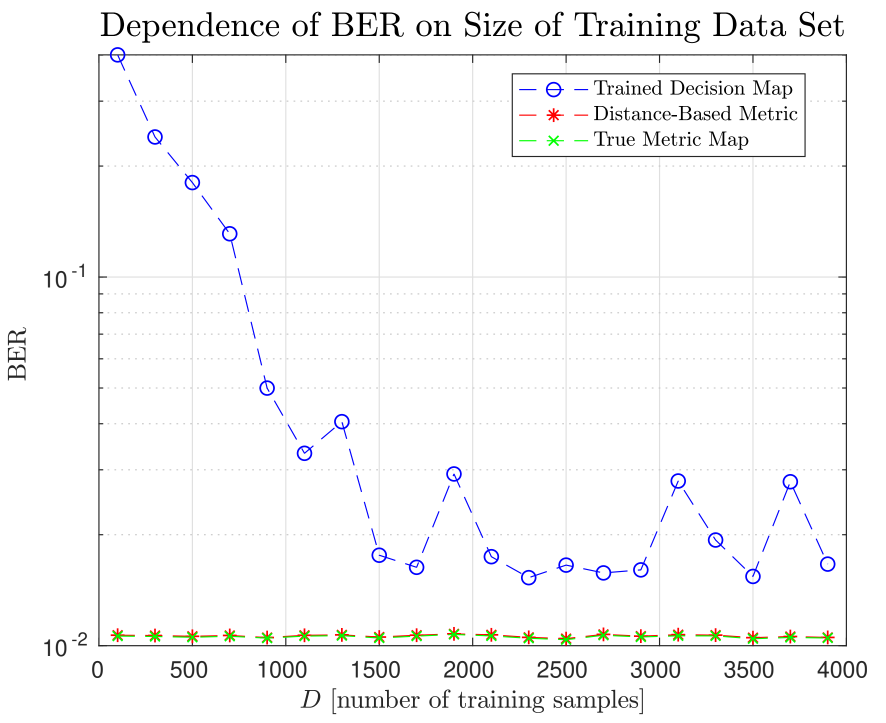


Figure 5.70: Comparison of BER for different values of D for SNR = 5 dB.

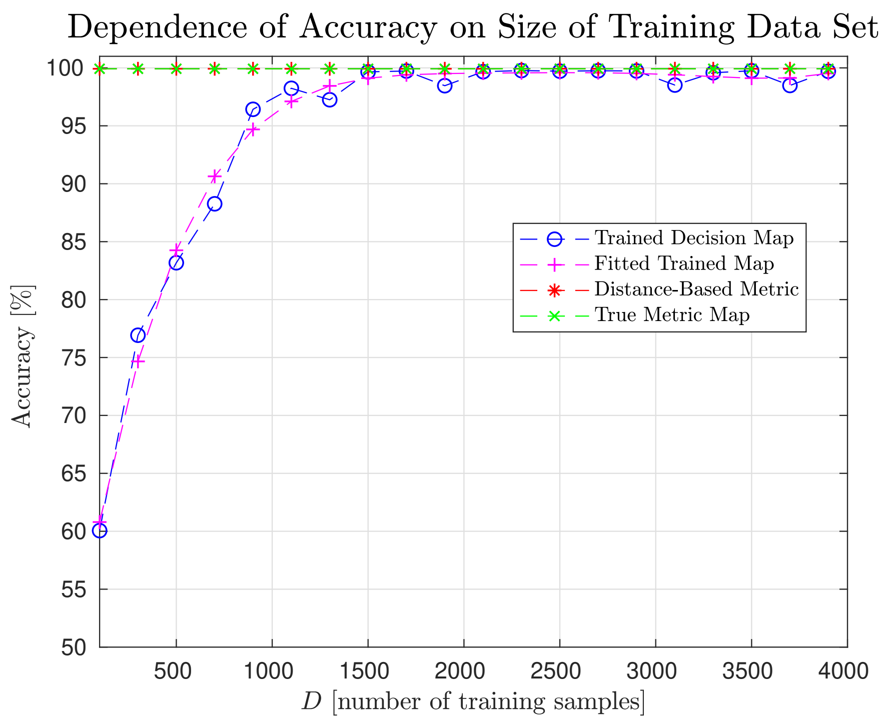


Figure 5.71: Comparison of accuracy for different values of D for SNR = 7,5 dB.

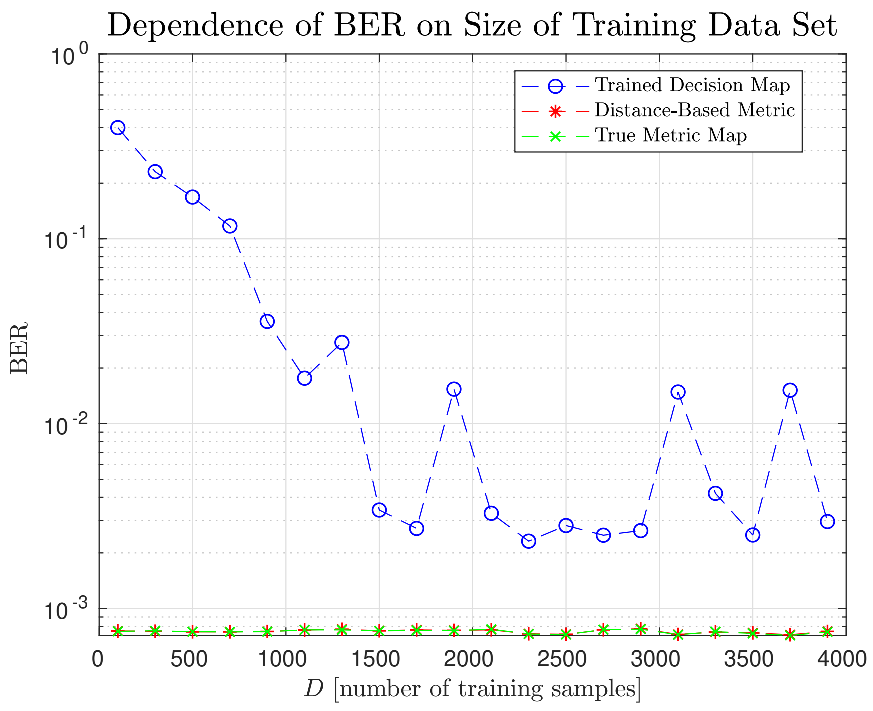


Figure 5.72: Comparison of BER for different values of D for SNR = 7,5 dB.

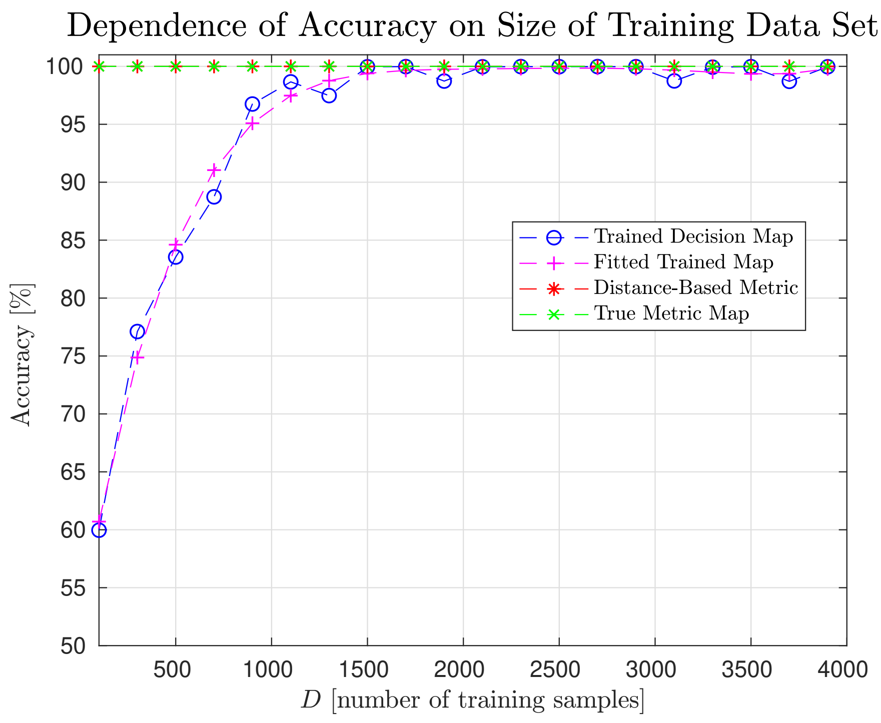


Figure 5.73: Comparison of accuracy for different values of D for SNR = 10 dB.

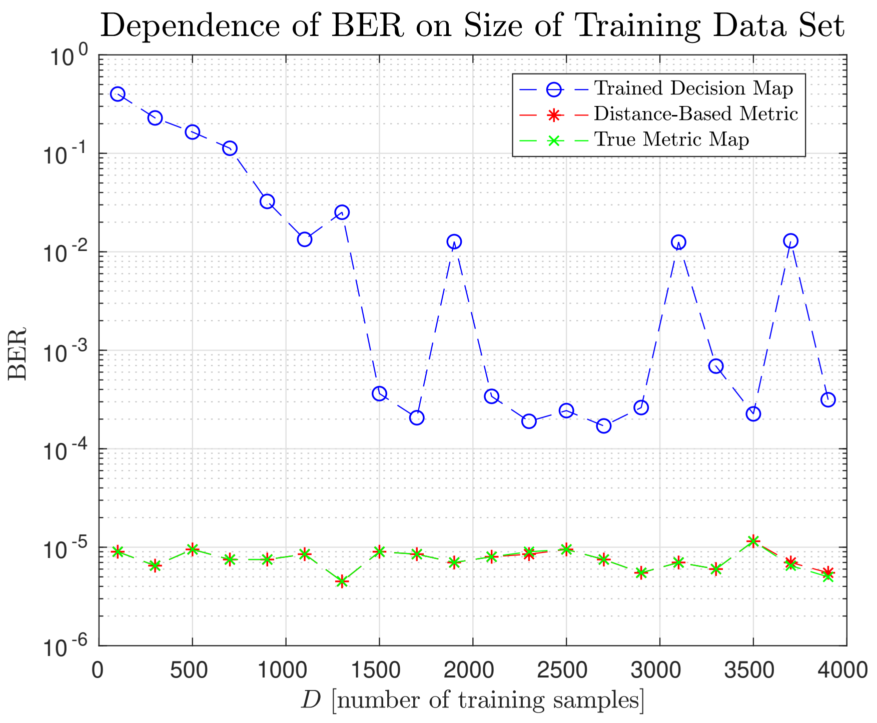


Figure 5.74: Comparison of BER for different values of D for SNR = 10 dB.

5.6 Variable Relative Fading in 2WRC with BPSK

In this section is shown an online mode modification of the algorithm that determines the decision regions for BPSK modulation in 2WRC for a variable value of relative fading parameter h. Parameters of the ANN used for this experiment are shown in Table 5.8. Note that the size of the training data set and the number of training epochs was significantly decreased, compared to Setup 1 in Section 5.3, the other parameters are preserved. These parameters were heuristically determined based on several experiments.

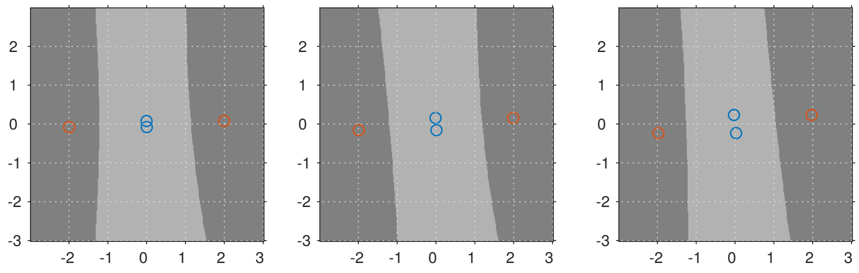
Parameter	Value
η	0,05
$n_{\text{hidden}}^{(1)}$	40
$n_{\text{hidden}}^{(2)}$	40
SNR of training data set	-1 dB
D	2 000
Number of training epochs	50

Table 5.8: Parameters of simulation for online mode of BPSK in 2WRC.

The explanation of the modification follows. All previous examples of trained decision regions were determined independently, during separated training experiments. At the beginning of all these experiments, all weights of the ANNs were initialized with small random numbers. In this section is assumed a variable relative fading parameter. Initially, magnitude is fixed for $|h| = 1$, and the variable is only the angle, denoted $\measuredangle h$. Further, the angle is assumed to be changing very slowly.

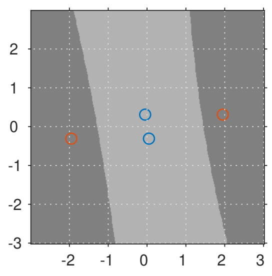
Let us present exemplary results. In this setup of the simulation, 2 000 training samples are provided and $\measuredangle h$ changed in 20 steps from $\measuredangle h = 4,5°$ to $\measuredangle h = 90°$. With all previous assumptions, the weights of the ANN are initialized to small random values only in the first step for $\measuredangle h = 4,5°$. In all subsequent steps, i.e., for $\measuredangle h = 4,5°$ to $\measuredangle h = 90°$, the previously trained weights of ANN are further modified in only 50 epochs of training.

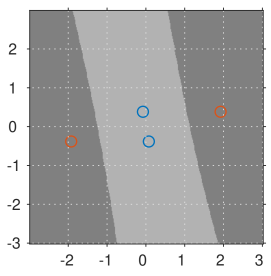
Resulting trained decision maps are shown in Figures 5.75 and 5.76. We can see that though the size of the training data set is significantly reduced, as well as the number of training epochs, the reached results are very similar to those previous in Section 5.3. For example, the decision map trained for $\measuredangle h = 90°$ in Figure 5.76 seems to be visually quite accurate, compared e.g. to the previous results in Figure 5.19.

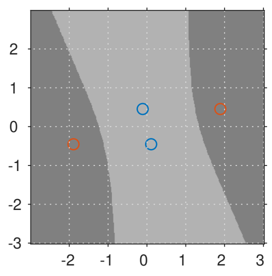


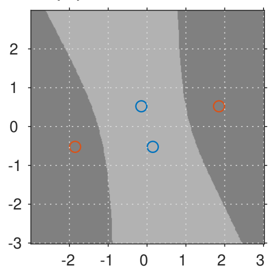
(a) : $\measuredangle h = 4,5°$ **(b)** : $\measuredangle h = 9°$ **(c)** : $\measuredangle h = 13,5°$

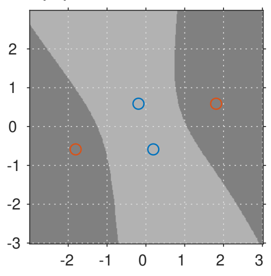
Figure 5.75: Examples of Online Mode.

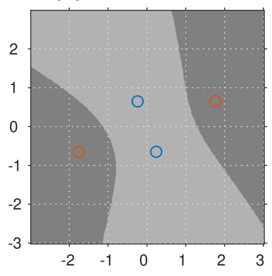

(a) : $\measuredangle h = 18^\circ$

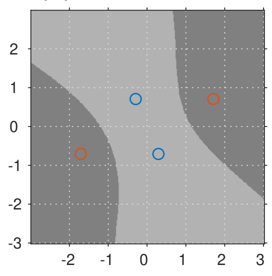

(b) : $\measuredangle h = 22,5^\circ$

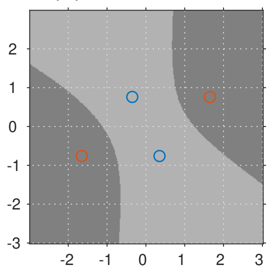

(c) : $\measuredangle h = 27^\circ$

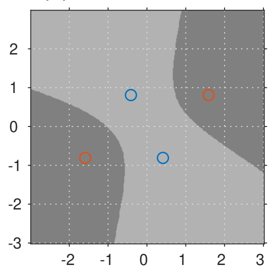

(d) : $\measuredangle h = 31,5^\circ$

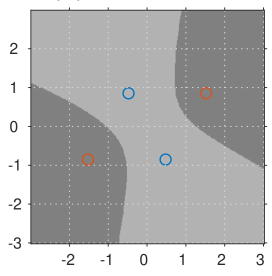

(e) : $\measuredangle h = 36^\circ$

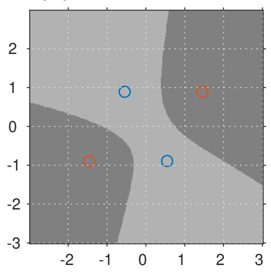

(f) : $\measuredangle h = 40,5^\circ$

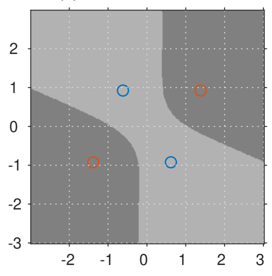

(g) : $\measuredangle h = 45^\circ$

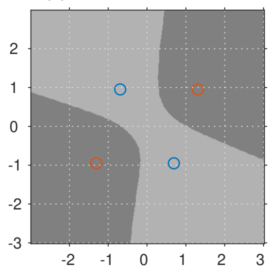

(h) : $\measuredangle h = 49,5^\circ$

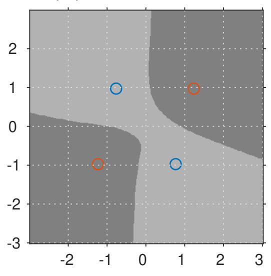

(i) : $\measuredangle h = 54^\circ$

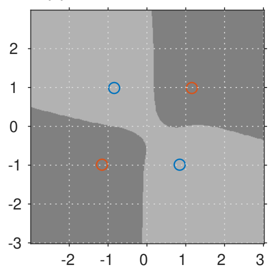

(j) : $\measuredangle h = 58,5^\circ$

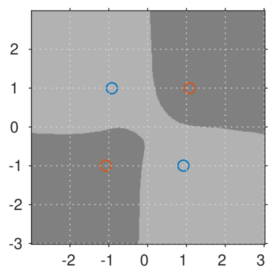

(k) : $\measuredangle h = 63^\circ$


(l) : $\measuredangle h = 67,5^\circ$


(m) : $\measuredangle h = 72^\circ$


(n) : $\measuredangle h = 76,5^\circ$


(o) : $\measuredangle h = 81^\circ$


(p) : $\measuredangle h = 85,5^\circ$

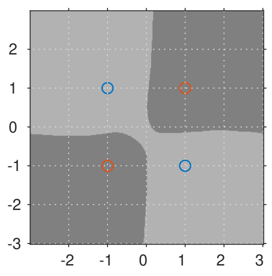

(q) : $\measuredangle h = 90^\circ$

Figure 5.76: Examples of Online Mode (continued).

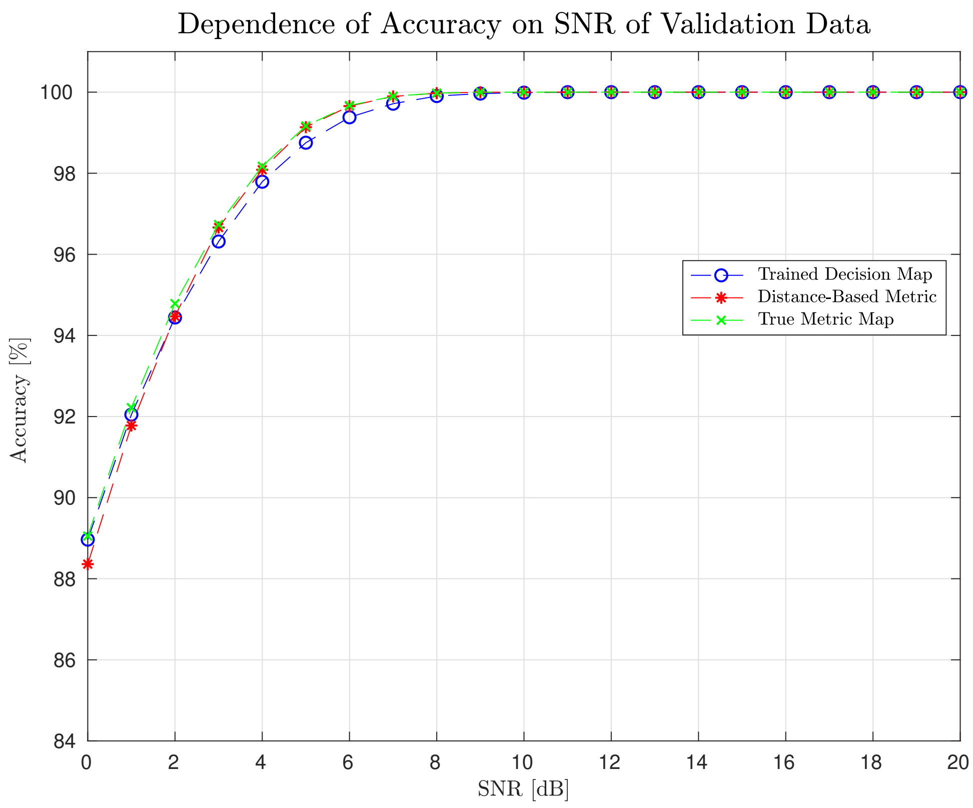


Figure 5.77: Comparison of accuracy in online mode for $\measuredangle h = 4,5^\circ$.

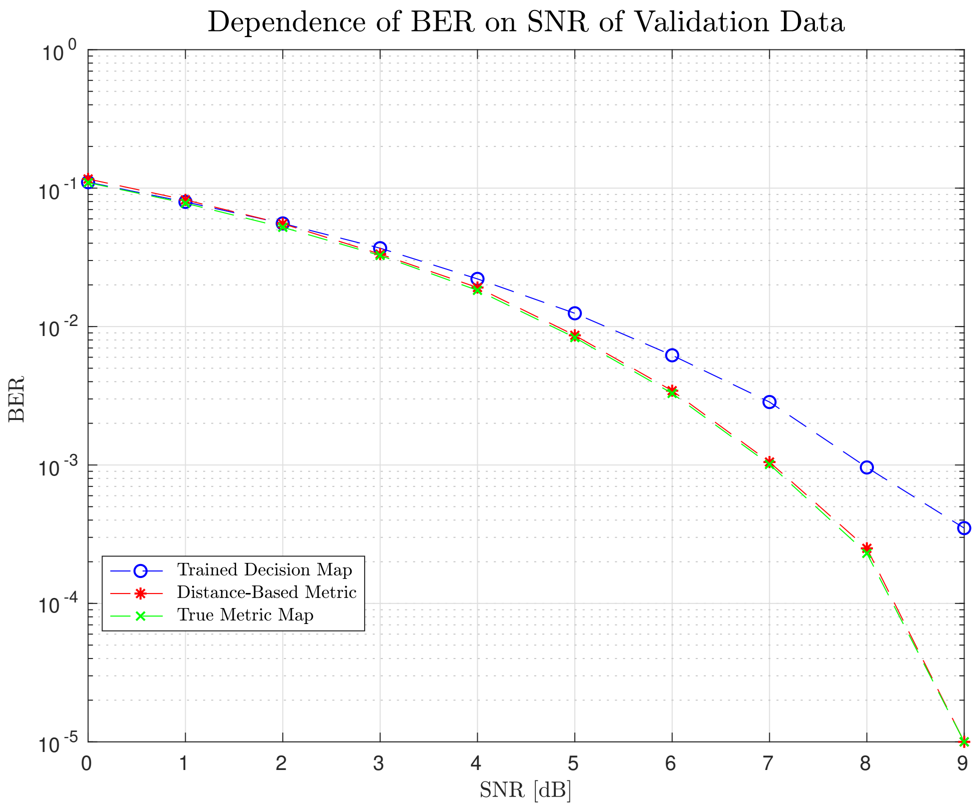


Figure 5.78: Comparison of BER in online mode for $\measuredangle h = 4,5^\circ$.

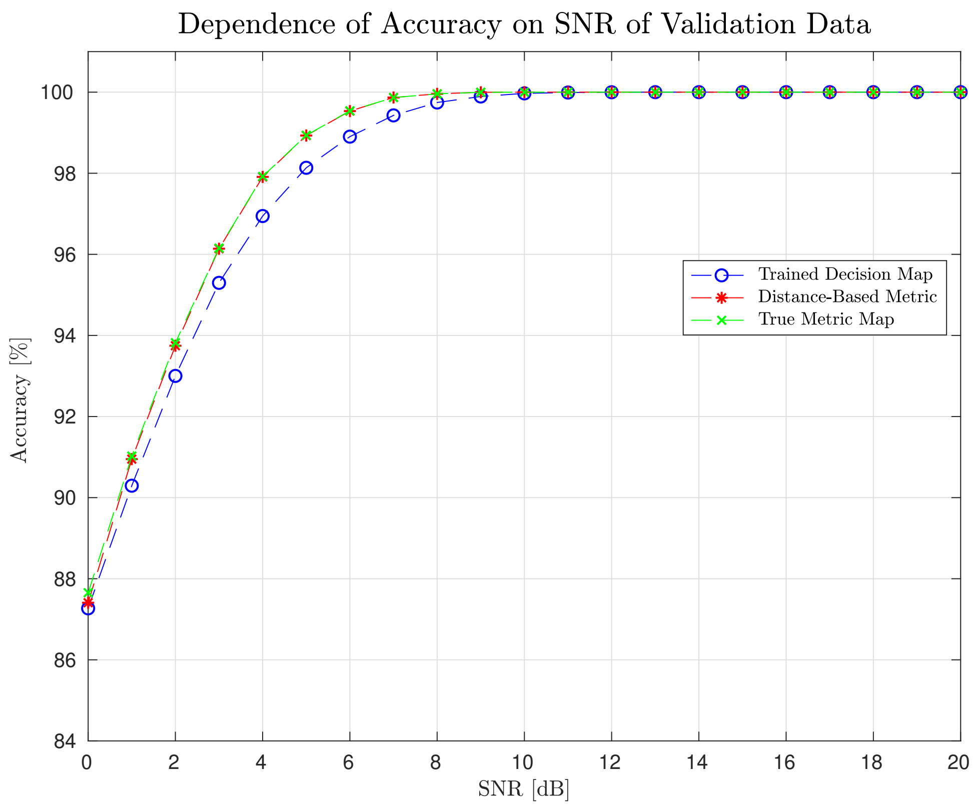


Figure 5.79: Comparison of accuracy in online mode for $\measuredangle h = 40,5^\circ$.

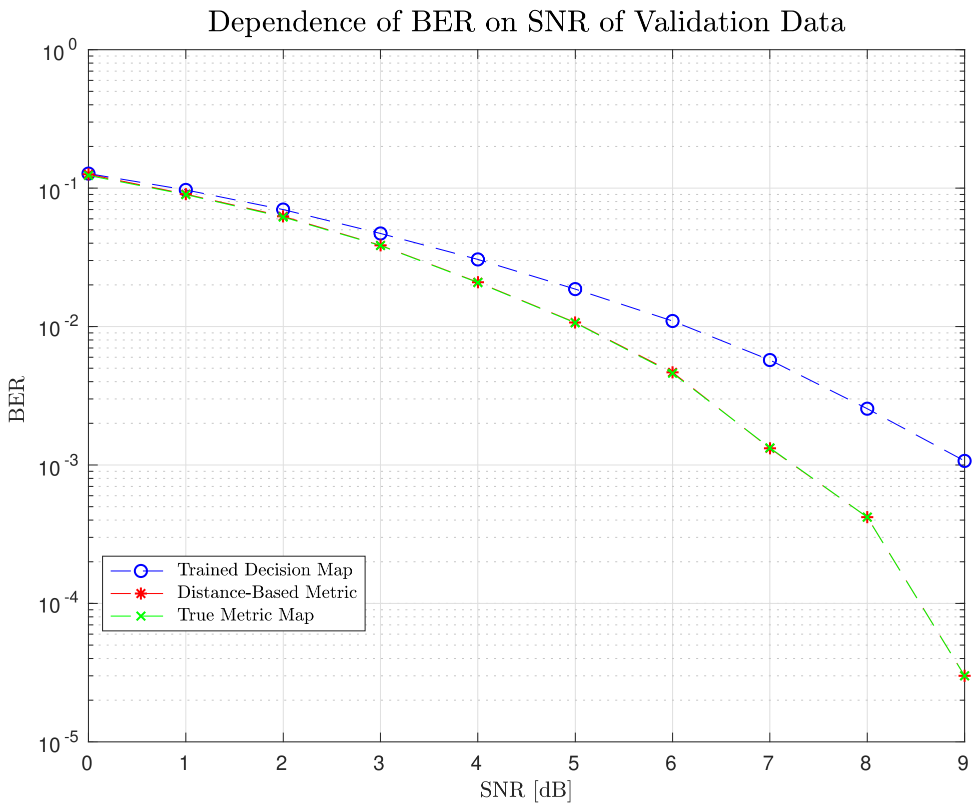


Figure 5.80: Comparison of BER in online mode for $\measuredangle h = 40,5^\circ$.

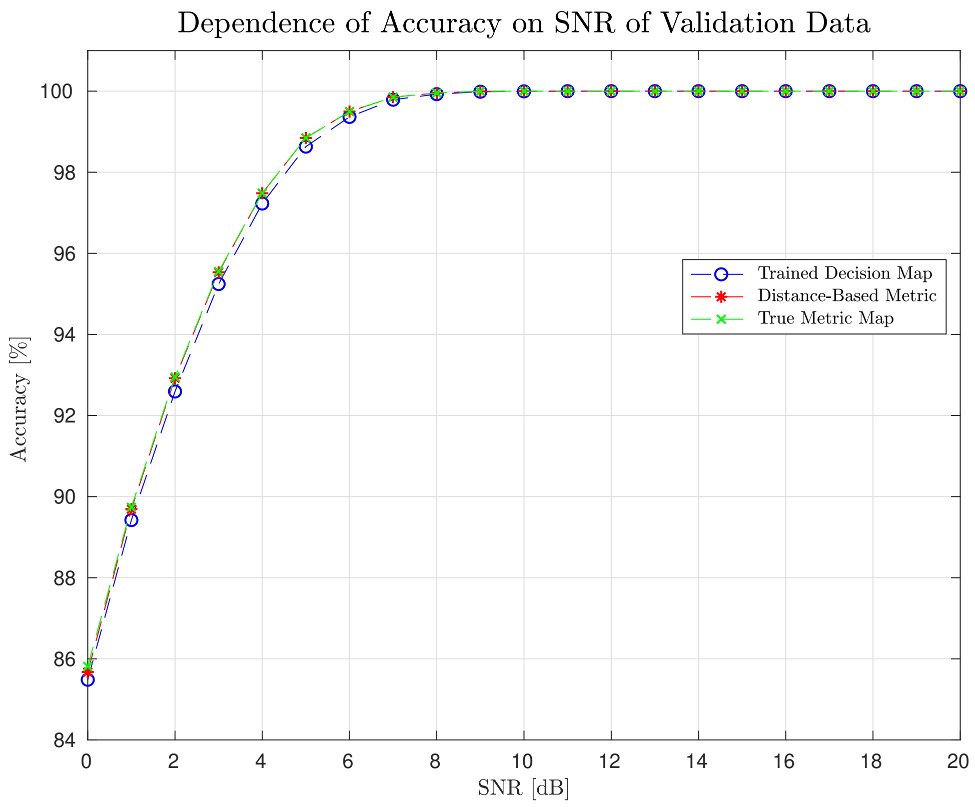


Figure 5.81: Comparison of accuracy in online mode for $\measuredangle h = 76,5°$.

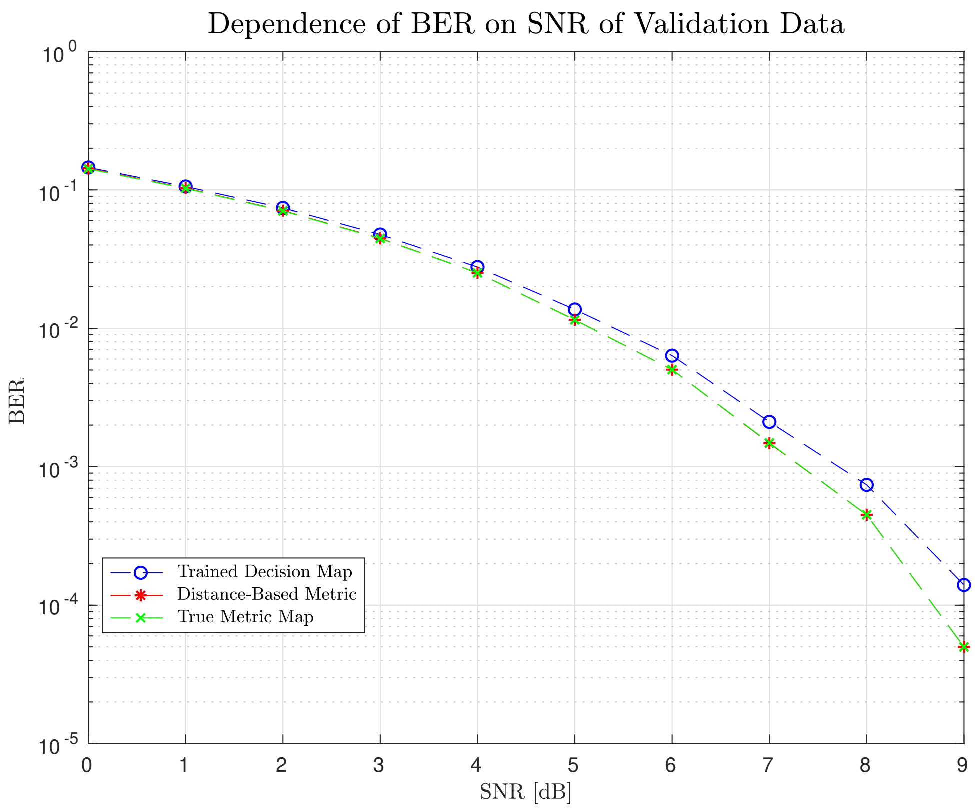


Figure 5.82: Comparison of BER in online mode for $\measuredangle h = 76,5°$.

Reference decision maps are hereby omitted; see results of Setup 1 in Section 5.3 for visual comparison. Instead, let us evaluate the performance of the results quantitatively using plots of BER and Accuracy, as previously. Results are shown in Figures 5.77-5.82. These were calculated using validation sets of 100 000 samples. We can see that the performance of trained decision maps is not identical for different angles. However, the results are competitive to performance reached in Setup 1 in Section 5.3.

5.6.1 Analysis of the Parameters and Tuning

The previous part shows that parameters of the considered ANN might be further tuned to reach satisfactory results with smaller computational complexity. To further discover this, we can observe how the weights of the ANN evolve during the training process. The motivation for this is to reduce the resources required for training as much as possible. Initially, we can focus on the number of required training samples and the number of training epochs. A product of these values dictates the time required for training. In Figure 5.83 is shown a plot with three curves, corresponding to three layers of the ANN. Each curve represents a sum of all squared differences of the weights between two subsequent epochs. Specifically, the shown variables in Figure 5.83 were obtained as

$$\omega^{(j)}(e) = \frac{1}{W^{(j)}} \sum_i \left[w_i^{(j)}(e) - w_i^{(j)}(e-1) \right]^2, \tag{5.5}$$

where j is the index of the layer of the ANN; $W^{(j)}$ denotes the number of weights in the jth layer, and index e distinguishes the subsequent epochs.

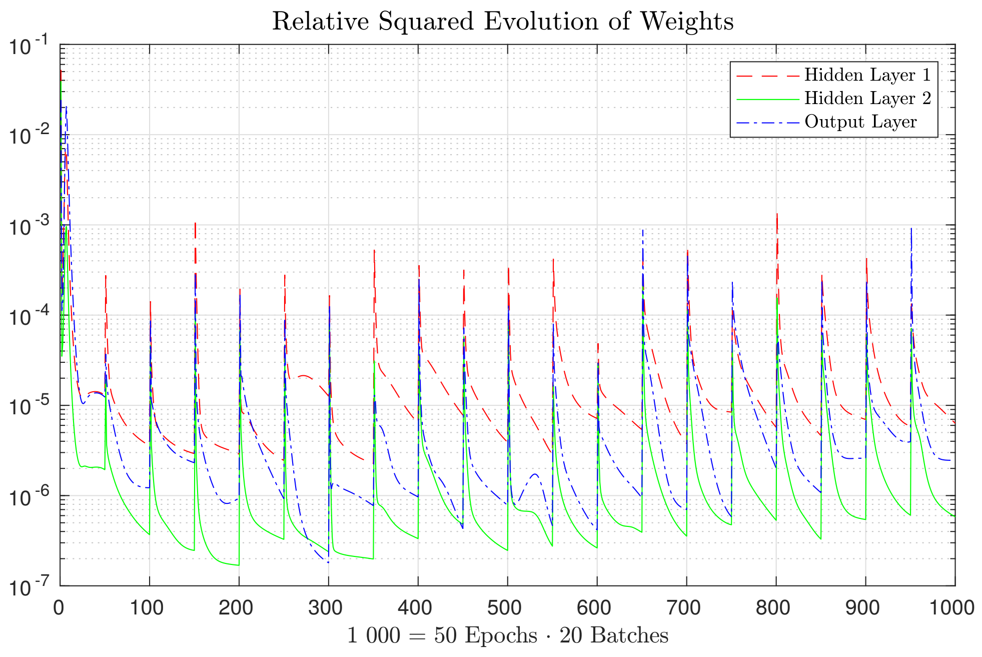


Figure 5.83: Visualization of changes of weights of ANN for 50 training epochs for 20 trained systems corresponding to 20 values of $\measuredangle h$. To avoid misunderstanding, on the horizontal axes are epochs of the training. Hereby, each batch lasts 50 epochs and 20 batches are shown (again, 20 values of $\measuredangle h$).

Let us comment this Figure 5.83. Note, that all epochs corresponding to a single value of $\measuredangle h$ are hereby referred to as a single *batch*. Firstly, we can clearly see that the largest rate of cumulative change of the weights was observed during the first batch, specifically within the initial 10 epochs. All subsequent updates were of a significantly smaller impact. Secondly, all batches are visually clearly separated by peak values, and the largest relative rate of changes was mostly observed for the weights in the first hidden layer (i.e. $w_i^{(1)}$). It is also worth note that all the subsequent batches showed very similar behavior.

In the previous section, we experimentally found that the considered ANN starts to be successfully trained with 2 000 training samples. We can fix this value and try to determine what is the required amount of training epochs. Asking this question, we can see in Figure 5.83, that within the first batch, the rate of changes significantly dropped after 30 epochs of training. Thus, the example was repeated, with the number of training epochs decreased from 50 to 30. Visualization of evolution of ANN weights is shown in Figure 5.84.

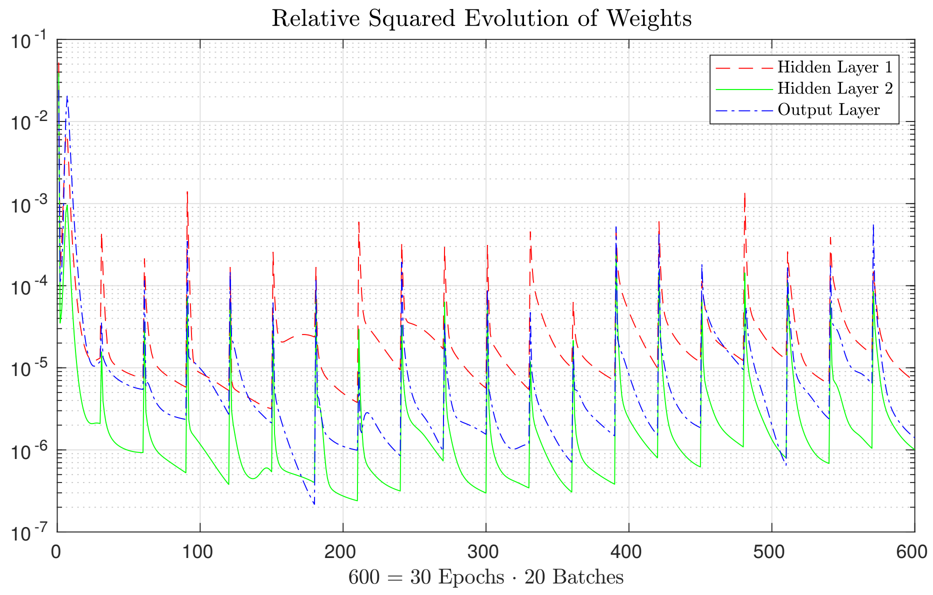


Figure 5.84: Visualization of changes of weights of ANN for 30 training epochs for 20 trained systems corresponding to 20 values of $\measuredangle h$. Thus hereby, the horizontal axes again carries the epochs, but 20 batches (for twenty different $\measuredangle h$) last only 30 epochs, compared to 50 in Figure 5.83.

In doing so, the amount of time required for training is significantly decreased, specifically by 40%. An appropriate question is: How does the decrease in the number of training the epochs affect performance? For this purpose, let us evaluate BER, averaged over all 20 tested values of $\measuredangle h$. The result is shown in Figure 5.85 and reveals that the impact seems to be negligible. As expected, a greater number of training epochs caused better performance. Note, 100 000 validation samples were used to test the performance over individual SNRs.

Due to a large number of parameters in the system, it is not clear how to optimize the parameters to save as many resources as possible. As noted, it is desired to minimize the product of the number of training samples, D, and the number of epochs denoted E. Two methods are proposed. Firstly, it is not necessary to keep the number of epochs E constant. Ideally, a long initial training sequence with a number of epochs E_1 might be used to train the

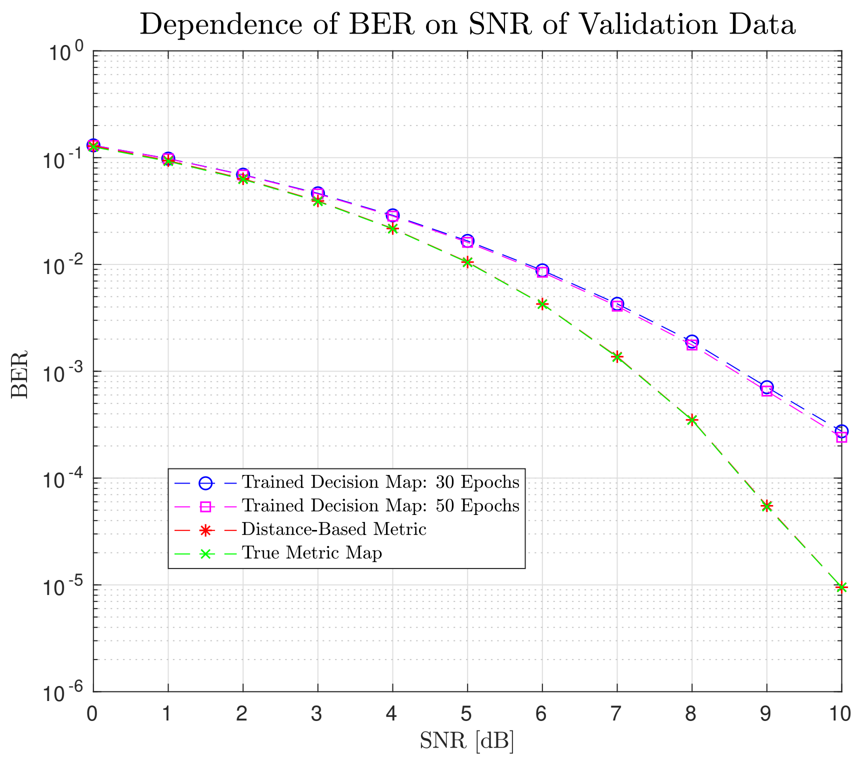


Figure 5.85: Comparison of averaged BER for reference methods and for average BER of trained systems with 30 and 50 training epochs.

system as previously. Several subsequent training epochs can be performed with a number of training epochs E_2, where $E_2 < E_1$. Secondly, the value of learning rate η might also be distinguished: Initially, let E_1 epochs be performed with learning rate η_1 with an aim to properly turn on the system. With an assumption of slowly changing $\measuredangle h$, subsequent training of E_2 epochs can be performed with slightly larger learning rate η_2 to partially compensate the smaller value of E_2.

Further, we might be interested, how these different values of E_1, E_2 and η_1, η_2 affect robustness of the classification. All the effect of past training is erased when the ANN is reinitialized with small random weights. It is desirable to explore how beneficial or malicious it is to keep the weights updated instead of reinitialized. Three routines are proposed to study these effects and used for comparison.

The first routine is shown in Figure 5.86. It is the simplest scenario, where the weights are reinitialized before each training. It differs from the model in previous Sections, such that initial training is performed with parameters E_1, η_1, and then $I-1$ training for subsequent values of $\measuredangle h$ are performed with parameters E_2, η_2. This is meant to reduce the number of training epochs, as described above. Besides this modification, it is the same scenario as implemented before.

The second implemented training route is shown in Figure 5.87. In this case, the reinitialization is performed only if $i = I$, and thus the number of reinitializations is decreased, and the effect of memory is introduced. The random reinitialization is, in this case, meant to improve the robustness of the routine. It tries to find a balance between the scenarios, where the weights of

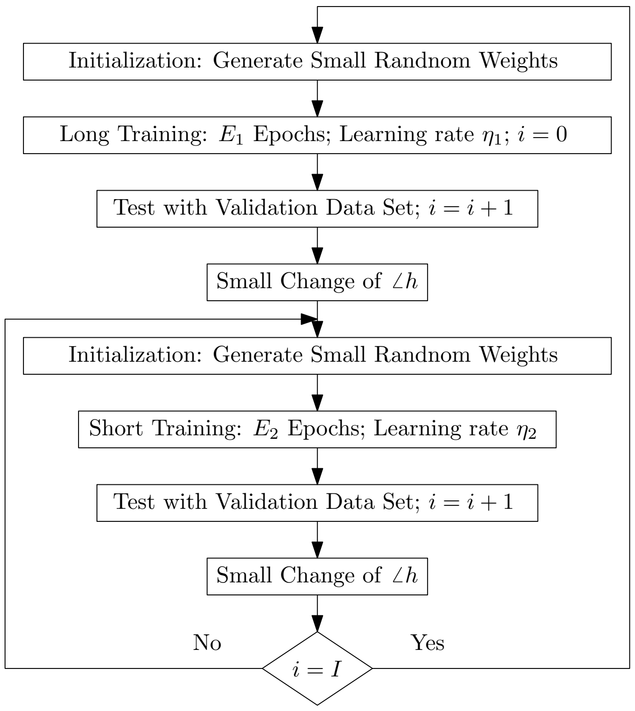

Initialization: Generate Small Randnom Weights
Long Training: E_1 Epochs; Learning rate η_1; $i = 0$
Test with Validation Data Set; $i = i + 1$
Small Change of $\angle h$
Initialization: Generate Small Randnom Weights
Short Training: E_2 Epochs; Learning rate η_2
Test with Validation Data Set; $i = i + 1$
Small Change of $\angle h$
No
Yes
$i = I$

Figure 5.86: Block diagram illustrating the implemented training routine, when all the weights are reinitialized before each training.

ANN are randomly reinitialized before each training, and where the weights are only updated. The longer training epochs are used for reinitialization because therein seems the training to be the most efficient.

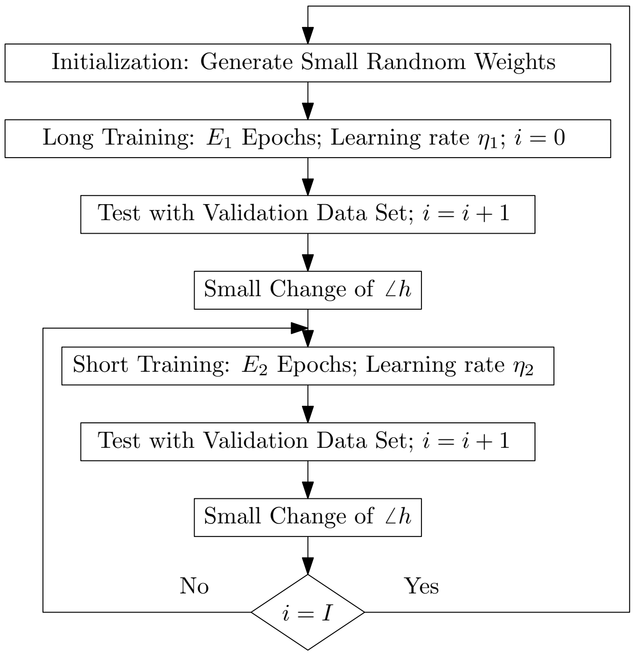

Initialization: Generate Small Randnom Weights
Long Training: E_1 Epochs; Learning rate η_1; $i = 0$
Test with Validation Data Set; $i = i + 1$
Small Change of $\angle h$
Short Training: E_2 Epochs; Learning rate η_2
Test with Validation Data Set; $i = i + 1$
Small Change of $\angle h$
No
Yes
$i = I$

Figure 5.87: Block diagram illustrating the implemented training routine, when all the weights are reinitialized only if $i = I$.

The last scenario to be compared is described in Figure 5.88, and in this case, the weights are randomly initialized only before the first training. All the subsequent training is thus a modification of the previously trained systems. This scenario assumes that the short training is sufficient to follow the dynamics of the system, i.e., the changes of $\measuredangle h$ are only very small.

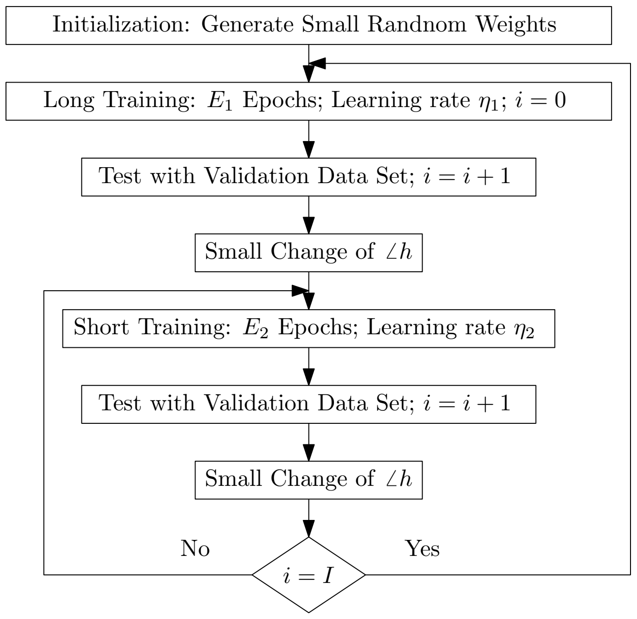

Initialization: Generate Small Randnom Weights
Long Training: E_1 Epochs; Learning rate η_1; $i = 0$
Test with Validation Data Set; $i = i + 1$
Small Change of $\angle h$
Short Training: E_2 Epochs; Learning rate η_2
Test with Validation Data Set; $i = i + 1$
Small Change of $\angle h$
No
$i = I$
Yes

Figure 5.88: Block diagram illustrating the implemented training routine, when all the weights are randomly initialized only before the first training.

These three described routines were implemented and used for comparison. The point of using these three systems is to evaluate quantitatively, what is the effect of a random reinitialization of the weights of the ANN for variable values of $\measuredangle h$. Two values of I were used for simulations, $I = 3$, and $I = 5$.

Parameter	Value
η_1	0,05
η_2	0,07
$n^{(1)}_{\text{hidden}}$	40
$n^{(2)}_{\text{hidden}}$	40
D	2 000
SNR of training data set	-1 dB
E_1	30
E_2	20
$\measuredangle h$	80 steps $(4,5^\circ : 4,5^\circ : 360^\circ)$
Size of validation data set	100 000

Table 5.9: Parameters of simulation for variable $\measuredangle h$.

Further parameters of the performed simulations may be found in Table 5.9. Let us emphasize that $D = 2\ 000$ training samples were used for each training and that 80 different values of $\measuredangle h$ were used for verification with a step of $4,5^\circ$.

The resulting comparison may be found in Figure 5.89. Note that for different procedures, only the same value of I is to be compared, since these were obtained using the same amount of computations. Let us comment on these results. We observe that the least successful approach resulted from the reinitialization of the weights before each training. This fact validates that it is meaningful to update the weights, instead of randomly reinitialize them before each training. The most successful results were obtained from the scenario, where the weights were randomly initialized only before the first training and subsequently were only modified by the backpropagation algorithm, instead of randomly reinitialized. Observing these results, we can also conclude that the smaller value of parameter I, which controls the rate of longer training processes, results in better performance. This is intuitive since it implies more training epochs. A middle performance is shown for a random reinitialization before the long training processes, though, the difference is not very significant.

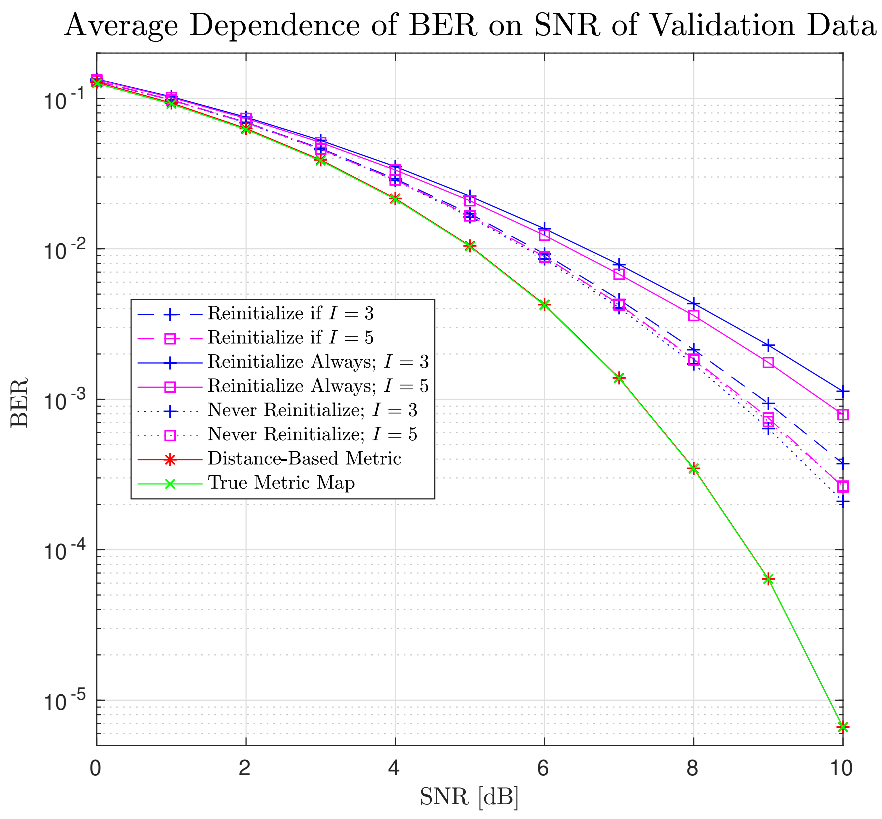


Figure 5.89: Comparison of reference methods and resulting average BER of different training procedures.

In Figures 5.90 – 5.92 are provided visualizations of updated of the weights of the ANN for the value of $I = 5$ for all three considered scenarios. These figures are used to provide detailed information on how the training process differed for individual scenarios. Let us emphasize that this experiment was designed to fairly compare the individual methods, that were allowed to perform a fixed number of training steps. (Operations for reinitialization of the ANN are neglected.) We also remind that these plots are showing *relative* changes of weights. Compared to the previous Figure 5.84, the bottom levels of changes in the weights are for all the scenarios similar and around the level of 10^{-6}. The weights in the first hidden layer seem to change the most, during the training process. Note, weights evolution for only the first 35 out of 85 values of $\measuredangle h$ are provided to make the figures more illustrative.

Finally, in Figure 5.93 is demonstrated, how much variance is in BER of individually trained system. Only a single figure is provided for scenario according to Figure 5.88, where the weights are never reinitialized, for $I = 5$. Variability of the individual BERs was observed to be very similar to this result for all the trained scenarios. Most of the trained systems seem to be well trained, however performance demonstrated for several values of $\measuredangle h$ is

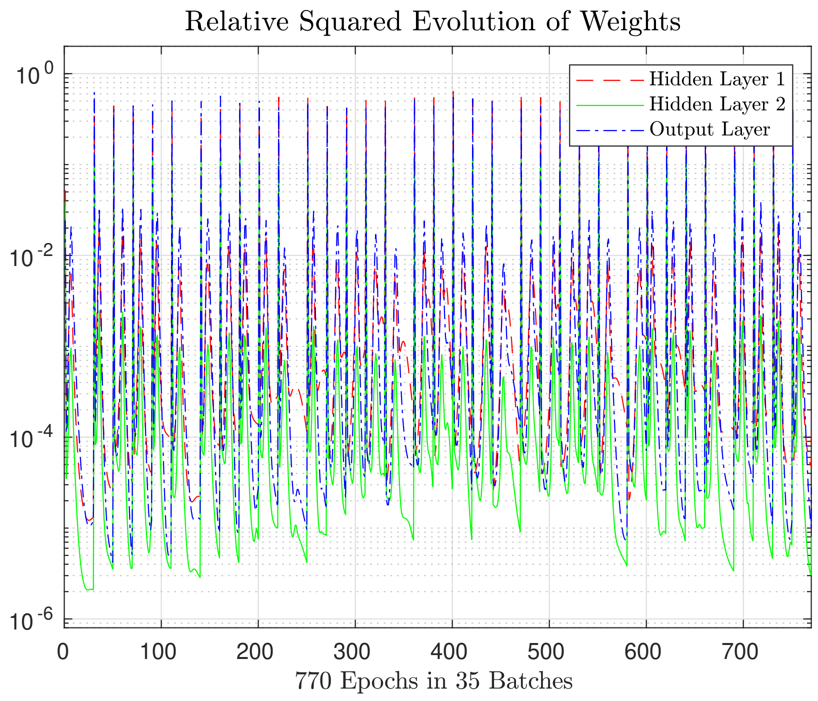


Figure 5.90: Visualization of changes of weights of ANN according to training procedure in Figure 5.87, where the weights are reinitialized before each training. It is not very illustrative, but we can observe rapid changes during all epochs of the training, caused by reinitialization. Within individual batches, corresponding to the individual values of $\measuredangle h$, a significant peak is caused by re-initialization of the weights to small random numbers. In this and two subsequent figures, 35 batches correspond to 35 values of $\measuredangle h$. Since $I = 5$, first batch lasts 30 epochs; 4 subsequent batches last 20 epochs. Therefore 7 complete cycles of 110 $(= 30 + 4 \cdot 20)$ epochs are shown.

very bad, and these probably do not meet the minimal conditions to allow successful communication. It is questionable, how meaningful it is to observe an average value of BER; possibly, the worst cases might be compared.

In this section was presented a modification of the previously trained system for an operation in a channel with a slowly varying angle of relative phase coefficient. Aspects of tuning the systems for minimal computational complexity were considered and addressed. Three scenarios were implemented and compared. The results were demonstrated, and the parameters were tuned to obtain satisfactory results for a scenario, where the weights of the ANN were updated during slow changes of the $\measuredangle h$. For the fixed amount of training samples $D = 2\ 000$, the number of training epochs was (on average) decreased from 50 per batch to 22 for $I = 5$. Quantitatively, the number of operations required for training was thus decreased by 56 %.

This approach might be virtually considered as an alternative to Hierarchical Channel State Estimator, presented in [25] and [26].

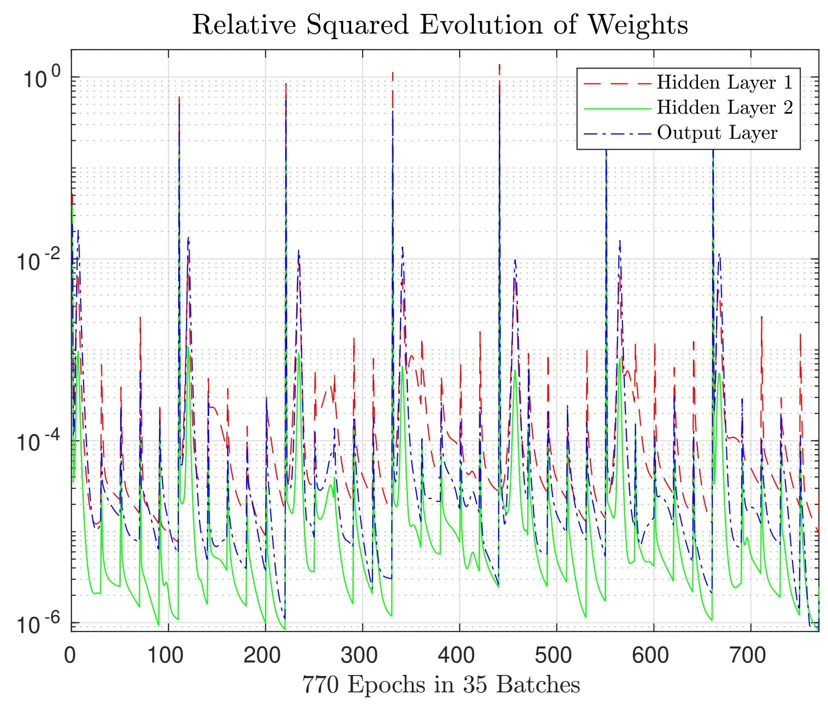


Figure 5.91: Visualization of changes of weights of ANN according to training procedure in Figure 5.87, where the weights are reinitialized only when $i = I$. All 35 individual batches corresponding to individual values of $\measuredangle h$ are visually clearly separated.

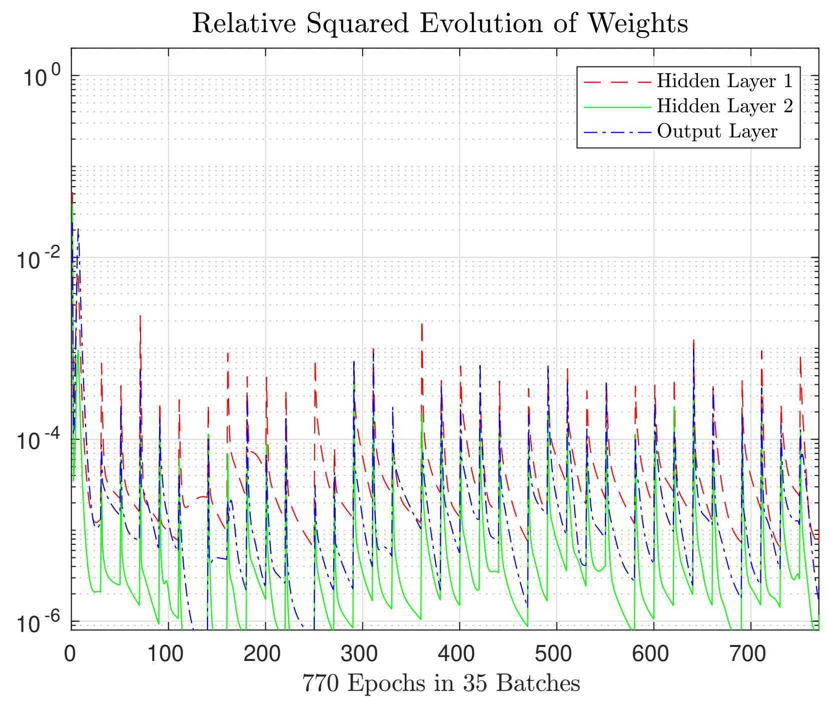


Figure 5.92: Visualization of changes of weights of ANN according to training procedure in Figure 5.88. Compared to previous Figures 5.90 and 5.91, we can clearly distinguish, that the updates during the training process are significantly smaller and the only significant peak value corresponds to the only random initialization at the beginning of the simulation.

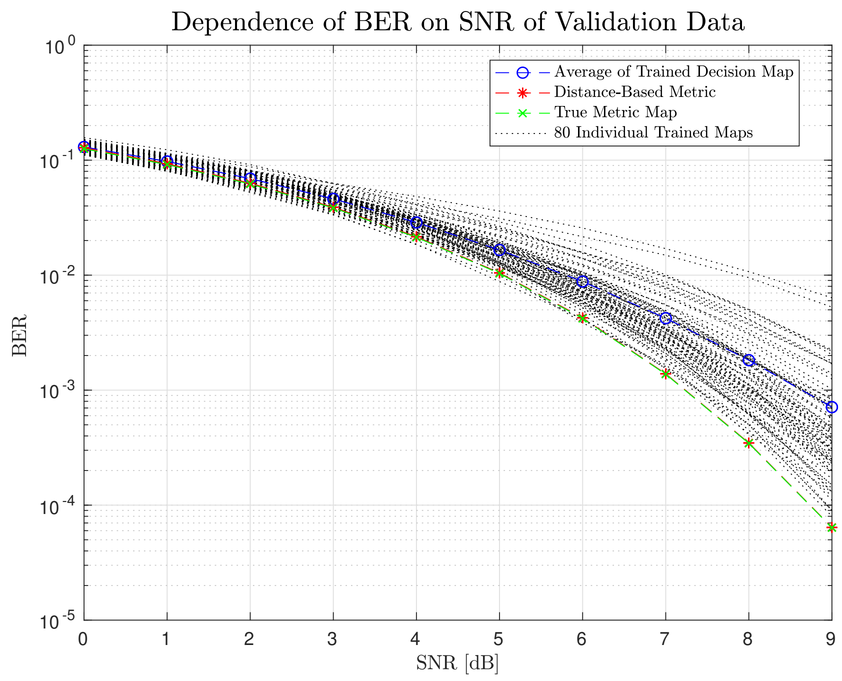


Figure 5.93: Comparison of reference methods and resulting average BER together with BER of all 80 individually trained systems for different $\measuredangle h$, according to training procedure in Figure 5.88 for $I = 5$.

Chapter 6

Conclusions

The main objective of this thesis was to investigate how the approaches of ML might be useful in the field of WPLNC. In the introductory chapters, the fundamental principles of ML and WPLNC were briefly explained. Subsequently, the fundamental theory about ANN was addressed, and examples were implemented.

The knowledge obtained in the initial part was then utilized to define selected problems from the broad topic of WPLNC. Let us summarize the achieved results. In Section 5.2 was considered a problem of an unknown amount of transceivers in the WPLNC scenario, simplified by an assumption of zero phase-shift. In Section 5.3 was considered a problem of classification of hierarchical symbols in 2WRC with BPSK and developed a system to perform this classification and to determine the corresponding decision regions. This task was subsequently extended and tested in 2WRC with QPSK in Section 5.4. Results provided in these sections demonstrate that the trained systems based on ANN have representational ability to perform these tasks. Reference solutions were implemented and used to evaluate the performance.

Since the parameters of the trained systems were determined experimentally, the number of training samples used for training was quite excessive. Therefore, Section 5.5 provided results that show a dependence of the performance of the training process on the size of the training data set. For a selected system, tested previously in Section 5.3, a sufficient number of training samples was experimentally determined.

To make the considered task of classification of hierarchical symbols more practical, in Section 5.6 was considered a situation where the relative fading parameter is slowly variable. An effort was made to minimize the resources required for the training of the system. A simple routine to solve this task was proposed an tested. The number of resources required for training was significantly reduced, compared to initial roughly guessed parameters, while average performance was not seriously decreased.

My contribution is that based on the studied theory (i.e., fundamentals of ML and WPLNC; ANN; stochastic gradient descent, backpropagation algorithm), I implemented and tested the performance of the solutions of the stated exemplary problems. The obtained results were evaluated and compared to the reference solutions, and graphically represented.

Bibliography

[1] E. Alpaydin, *Introduction to machine learning*, 2nd ed. Cambridge, Mass: MIT Press, 2010.

[2] C. M. Bishop, *Pattern recognition and machine learning.* New York: Springer, 2006.

[3] O. Simeone, “A brief introduction to machine learning for engineers,” *CoRR*, vol. abs/1709.02840, 2017. [Online]. Available: http://arxiv.org/abs/1709.02840

[4] O. Simeone, “A very brief introduction to machine learning with applications to communication systems,” *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 4, pp. 648–664, Dec 2018.

[5] S. Peng, H. Jiang, H. Wang, H. Alwageed, Y. Zhou, M. M. Sebdani, and Y. Yao, “Modulation classification based on signal constellation diagrams and deep learning,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 3, pp. 718–727, March 2019.

[6] S. M. Hiremath, S. Behura, S. Kedia, S. Deshmukh, and S. K. Patra, “Deep learning-based modulation classification using time and stockwell domain channeling,” in *2019 National Conference on Communications (NCC)*, Feb 2019, pp. 1–6.

[7] S. Fang and T. Lin, “Indoor location system based on discriminant-adaptive neural network in ieee 802.11 environments,” *IEEE Transactions on Neural Networks*, vol. 19, no. 11, pp. 1973–1978, Nov 2008.

[8] C. Hsu, Y. Chen, T. Juang, and Y. Wu, “An adaptive wi-fi indoor localization scheme using deep learning,” in *2018 IEEE Asia-Pacific Conference on Antennas and Propagation (APCAP)*, Aug 2018, pp. 132–133.

[9] A. Balatsoukas-Stimming, “Non-linear digital self-interference cancellation for in-band full-duplex radios using neural networks,” in *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, June 2018, pp. 1–5.

[10] A. Zappone, M. D. Renzo, and M. Debbah, “Wireless networks design in the era of deep learning: Model-based, ai-based, or both?” 2019.

[11] J. Jagannath, N. Polosky, A. Jagannath, F. Restuccia, and T. Melodia, "Machine learning for wireless communications in the internet of things: A comprehensive survey," *CoRR*, vol. abs/1901.07947, 2019. [Online]. Available: http://arxiv.org/abs/1901.07947

[12] "Machine learning for communications emerging technologies initiative," New York, 2019. [Online]. Available: https://mlc.committees.comsoc.org/research-library/

[13] T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention," in *2016 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, Dec 2016, pp. 223–228.

[14] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, Dec 2017.

[15] C. de Vrieze, S. Barratt, D. Tsai, and A. Sahai, "Cooperative multi-agent reinforcement learning for low-level wireless communication," 2018.

[16] T. Hynek, "Distributed algorithms for wireless physical layer network coding self-organisation in cloud communication networks," 2017. [Online]. Available: https://dspace.cvut.cz/handle/10467/67682

[17] T. Matsumine, T. Koike-Akino, and Y. Wang, "Deep learning-based constellation optimization for physical network coding in two-way relay networks," 2019.

[18] A. Pastore, P. de Kerret, M. Navarro, D. Gregoratti, and D. Gesbert, "Neural network aided decoding for physical-layer network coding random access," 2018.

[19] J. Sykora and A. Burr, *Wireless physical layer network coding.* Cambridge: Cambridge University Press, 2018.

[20] T. Uřičář, "Wireless (physical layer) network coding design for parametric channels and systems with partial hierarchical side-information," 2014. [Online]. Available: https://dspace.cvut.cz/handle/10467/22862

[21] X. Wang, L. Kong, F. Kong, F. Qiu, M. Xia, S. Arnon, and G. Chen, "Millimeter wave communication: A comprehensive survey," *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 1616–1653, thirdquarter 2018.

[22] T. M. Mitchell, *Machine learning.* Boston: McGraw-Hill, 1997.

[23] P. Chandra and Y. Singh, "Regularization and feedforward artificial neural network training with noise," vol. 3, pp. 2366–2371 vol.3, July 2003.

[24] A. Seghouane, Y. Moudden, and G. Fleury, "On learning feedforward neural networks with noise injection into inputs," pp. 149–158, Sep. 2002.

[25] P. Hron, "Channel estimation and network coded modulation for parametric H-MAC Channel in WPNC Radio Networks," 2019. [Online]. Available: https://dspace.cvut.cz/handle/10467/82566

[26] P. Hron and J. Sykora, "Performance analysis of hierarchical decision aided 2-source BPSK H-MAC CSE with feed-back gradient solver for WPNC networks," 2019.

Appendix A

Appendix

A.1 MATLAB Code to Train ANN with Backpropagation Algorithm to Classify XOR

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Script to illustrate backpropagation algorithm %%%%%%
%%%%%%       Jakub Kolar, last edit 19. 12. 2019     %%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%this specific example learns XOR function classification
%% initialization of the ANN
clear all;
close all;
D=4; %number of training samples
x=[0 0; 0 1; 1 0; 1 1]; % training inputs - samples
t=[0;1;1;0]; % training inputs - desired outputs
n_in = 2; %parameters of the network
n_out = 1;
n_hidden = 2;
K=0.2;
eta = .5; % learning rate
w_hidden=K*randn(n_in, n_hidden); %initialize weights
bias_hidden=K*randn(n_hidden,1);
w_output=K*randn(n_hidden, n_out);
bias_out=K*randn(n_out,1);
c=0; %to count outputs

%% training process
for epoch = 1: 5000 % iterate epochs

for i = 1:4 %iterate training inputs
    c=c+1;

%forward feed the network
for h = 1 : n_hidden
activation_hidden(h) = x(i,:)*w_hidden(:,h);
activation_hidden(h)= activation_hidden(h) + bias_hidden(h);
sigmoid_hidden(h) = 1/(1+exp(-activation_hidden(h)));
end
```

```
activation_output=sigmoid_hidden*w_output;
activation_output=activation_output+bias_out;
sigmoid_output= 1/(1+exp(-activation_output));

%%backpropagate error

%compute delta parameters
delta_output = sigmoid_output*(1-sigmoid_output)* ...
(t(i)-sigmoid_output);
err(c)=t(i)-sigmoid_output; %capture error term of the network

for h = 1 : n_hidden
delta_h(h)=sigmoid_hidden(h)*(1-sigmoid_hidden(h))* ...
delta_output*w_output(h);
end

%update the weights
bias_out = bias_out+ eta*delta_output;

for h = 1 : n_hidden
   bias_hidden(h) = bias_hidden(h)+eta*delta_h(h);
   w_hidden(:,h)=w_hidden(:,h)+eta*x(i,:)'*delta_h(h);
   w_output=w_output+eta*delta_output*sigmoid_hidden';
end

end %end of cycle for training samples
end %end of cycle for the epochs

%% plot decision boundary

% how to feed trained network with arbitrary data x
x=[0 1];
for h = 1 : n_hidden
activation_hidden(h) = x*w_hidden(:,h);
activation_hidden(h)= activation_hidden(h) + bias_hidden(h);
sigmoid_hidden(h) = 1/(1+exp(-activation_hidden(h)));
end

activation_output=sigmoid_hidden*w_output;
activation_output=activation_output+bias_out;
sigmoid_output_test= 1/(1+exp(-activation_output));

% determine the decision boundary
x_val=-.10:1/100:1.1;
x_21=-(w_hidden(1,1)/w_hidden(2,1))*x_val- ...
bias_hidden(1)/w_hidden(2,1);
x_22=-(w_hidden(1,2)/w_hidden(2,2))*x_val- ...
bias_hidden(2)/w_hidden(2,2);
x_3=-bias_out/w_output;

%% figure to visualize decision boundary and data
figure;
```

```
plot(x_val,x_21);
hold on;
plot(x_val,x_22);
axis([-0.1 1.1 -0.1 1.1]);
hold on;
scatter([0;1],[1;0]);
hold on;
scatter([1;0],[1;0]);
hold off;
ttl1=title( 'Learned␣XOR␣function'  );
ttl1=set(ttl1,'Interpreter','latex','FontSize', 15);
xlbl1=xlabel('$x_1$');
xlbl1=set(xlbl1,'Interpreter','latex');
ylbl1=ylabel('$x_2$');
ylbl1=set(ylbl1,'Interpreter','latex');
lgd = legend('Decision␣Boundary␣of␣Hidden␣Neuron␣1', ...
    'Decision␣Boundary␣of␣Hidden␣Neuron␣2',...
    'XOR($x_1,x_2$)=␣1','XOR($x_1,x_2$)=␣0'  );
lgd=set(lgd,'Interpreter','latex');
grid on;

%% figure to visualize convergence (error term reduces)
figure;
plot(1:epoch,(err(1:4:end)));
hold on;
plot(1:epoch,(err(2:4:end)));
hold on;
plot(1:epoch,(err(3:4:end)));
hold on;
plot(1:epoch,(err(4:4:end)));
hold off;
ttl1=title( 'Error␣Term␣␣for␣Different␣Training␣Inputs'  );
ttl1=set(ttl1,'Interpreter','latex','FontSize', 15);
xlbl1=xlabel('Training␣Epochs');
xlbl1=set(xlbl1,'Interpreter','latex'); % latex friendly :o)
ylbl1=ylabel('Error␣Term␣$(t-o)$␣');
ylbl1=set(ylbl1,'Interpreter','latex');
lgd = legend('$(t-o)$␣for␣$x_1␣=␣0,␣x_2=␣0,␣t␣=␣0$', ...
             '$(t-o)$␣for␣$x_1␣=␣0,␣x_2=␣1,␣t␣=␣1$', ...
             '$(t-o)$␣for␣$x_1␣=␣1,␣x_2=␣0,␣t␣=␣1$', ...
             '$(t-o)$␣for␣$x_1␣=␣1,␣x_2=␣1,␣t␣=␣0$');
lgd=set(lgd,'Interpreter','latex');
grid on;
```

A.2 Folder List of the Attached CD

```
thesis.pdf %text of the thesis
codes      %contains all the utilized MATLAB scripts
    chapter_2
       polynomilal_curve_fitting
           figures_pol_fit
    chapter_4
       1_gradient_descent
       2_sigmoid
       3_backpropagation
       4_auto_encoder
    chapter_5
        Section_5_2_count_users
        Section_5_3_detect_symbols_2wrc_bpsk_xor
           intro_example
           Setup_1
              Setup_1_figures
              SNR_minus_1_500
                  h_0_7
                  h_1
           Setup_2
              Setup_2_figures
              SNR_minus_1_500_70
                  h_0_7
                  h_1
           test_SNR_bpsk
        Section_5_4_detect_symbols_2wrc_qpsk_xor
           simulation_results_SNR_11
              figs_SNR_11
              h_0_7
              h_1
           SNR_qpsk
               0_7
               0_7_j_pi_6
               1
               pi_6
        Section_5_5_effect_of_D_to_training
           results
               SNR_10
               SNR_5
               SNR_7_5
        Section_5_6_online_mode_2wrc_bpsk_xor
            Section_5_6_1_tuned_parameters
               1_reinit_I_3
               2_reinit_I_5
               3_reinit_always_I_3
               4_reinint_always_I_5
               5_reinit_never_I_3
               6_reinit_never_I_5
            Section_5_6_compare_30_50_epochs
            Section_5_6_initial_example
```