Reliability of Extreme Learning Machines

by **Klaus Neumann**

Ph.D. Thesis
Faculty of Technology, Bielefeld University
October 2013

Printed on permanent paper according to ISO 9706.
Gedruckt auf altersbeständigem Papier nach ISO 9706.

Abstract – Reliability of Extreme Learning Machines

Klaus Neumann

The reliable application of machine learning methods becomes increasingly important in challenging engineering domains. In particular, the application of extreme learning machines (ELM) seems promising because of their apparent simplicity and the capability of very efficient processing of large and high-dimensional data sets. However, the ELM paradigm is based on the concept of single hidden-layer neural networks with randomly initialized and fixed input weights and is thus inherently unreliable. This black-box character usually repels engineers from application in potentially safety critical tasks. The problem becomes even more severe since, in principle, only sparse and noisy data sets can be provided in such domains. The goal of this thesis is therefore to equip the ELM approach with the abilities to perform in a reliable manner.

This goal is approached in three aspects by enhancing the robustness of ELMs to initializations, make ELMs able to handle slow changes in the environment (i.e. input drifts), and allow the incorporation of continuous constraints derived from prior knowledge. It is shown in several diverse scenarios that the novel ELM approach proposed in this thesis ensures a safe and reliable application while simultaneously sustaining the full modeling power of data-driven methods.

Keywords. Machine learning, artificial neural network, random projection, extreme learning machine, reliability, cognitive robotics, continuum robotics, dynamical system, copper wire bonding.

Contents

Chapter 1

Introduction

In recent years, machine learning has matured to a point where the application of learning methods in the engineering of complex systems increasingly comes into focus of applied research. A prominent example is the German leading edge cluster "Intelligent Technical Systems"[1], which implements a dedicated project to apply self-optimization and learning methods in technical production systems and the intelligent mechatronical devices of tomorrow. In this application domain, such systems are supposed to interact with flexible environments, can handle unexpected situations, have access to large amounts of expert knowledge or data-encoded experience, and behave user-friendly. It is undoubted that learning rather than situation-dependent programming is the method of choice to handle the engineering of complex systems, e.g. where analytic models hardly exist or in non-stationary environments which require adaptability by means of re-learning.

But despite the remarkable success of machine learning techniques, which demonstrated their promising potential already in many relevant scenarios, their acceptance is still very weak. Yet, usually, parametric models with a fixed structure derived from first principles about the task are used for engineering challenges. The construction of parametric models requires precise expert knowledge and indeed avoids unexpected behavior. They are thus insufficient to assimilate any uncertainty that arises from the inherent process properties and are infeasible for application where the construction of analytic models is difficult.

Novel learning approaches that are qualified for application in relevant engineering tasks will face the challenge to ensure several different requirements simultaneously, while capitalizing on the full power of data-driven modeling methods in order to cope with the unpredictability of complex environments. A hierarchy of requirements on machine learning algorithms that I perceive as common-ground for machine learning is depicted in Fig. 1.1. I separated the different requirements into three main classes: speed, performance, and reliability.

In fact, many machine learning techniques exclusively focus on the performance and the speed of the learning algorithm [1, 2], but the acceptance of such methods is not only associated to these features. Also reliability plays an essential role [3]. The speed of the model is mainly driven by its computational complexity which can be distinguished in execution time, learning speed and memory consumption,

[1]This research and development project is funded by the German federal ministry of education and research (BMBF) within the leading-edge cluster competition "it's OWL".

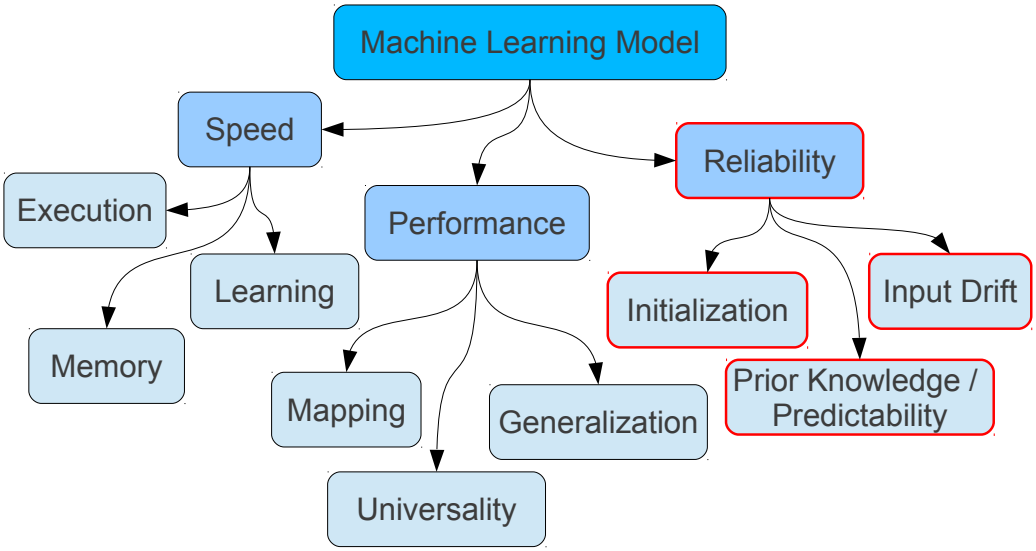

Fig. 1.1: Requirements on machine learning techniques separated into three main categories: speed, performance, and reliability.

but also the complexity due to the specific implementation is important [4]. However, the performance of the model is typically measured by training and test error which quantify the mapping and generalization ability [5]. Also the class of potentially learnable functions plays an essential role and is referred to as universality (universal function approximation capability) of the model [6]. The third important point is the reliability of the machine learning algorithm which can refer to a high probability, or even a mathematical proof, of a safe application (see e.g., [7]).

The incorporation of these demands becomes even more difficult since only sparse and noisy data sets can be provided. Exhaustive sampling of the underlying input-output relation is, in principle, impossible due to physical restrictions of the involved machinery, in particular, if many input dimensions are present. In addition, the provided data sets are typically highly non-linear which makes the recording of uniformly sampled data sets delicate and leads to big gaps in the data. Generalization towards such regions is particularly challenging and requires a dedicated treatment.

In the last decade, the re-advent of random projection methods in many challenging application domains attracted a lot of attention, because random projections, in principle, allow for very efficient processing of large and high-dimensional data sets [8]. These approaches initialize the free parameters randomly and restrict learning to linear methods for obtaining a suitable read-out function. A prominent method based on artificial feed-forward neural networks and high-dimensional random projections is the extreme learning machine (ELM) as proposed in [9]. It comprises a single hidden-layer feed-forward neural network with fixed random

input weights and a trainable linear output layer. ELMs have become popular, because, compared to traditional backpropagation training, they train much faster since output weights are computed in a single batch regression step. Despite this apparent simplicity, ELMs are universal function approximators with high probability under mild conditions if arbitrarily large networks can be considered [10]. Although ELMs are a promising basis for the solution of ambitious learning tasks, it is obvious that the inherently unreliable character of random projection methods repels engineers from application in potentially safety critical tasks.

Therefore, the goal of this thesis is to provide the extreme learning machine approach with the abilities to perform reliably in diverse, challenging engineering tasks by exploiting the simplicity, universality and computational efficiency of the model.

The special focus on reliability of ELMs is also visualized schematically in Fig. 1.1. These crucial demands on the reliability of ELMs are approached in three steps. In the first step, the robustness of the ELM performance against random initializations is tackled. An unsupervised technique that addresses this issue by providing task-specific features is intrinsic plasticity (IP). IP was already successfully used in the context of random projections [11] and for the optimization of static reservoir computing approaches [12]. In order to speed-up the learning, batch intrinsic plasticity (BIP) [13, 14] is introduced, which is a highly efficient batch version of IP. This approach is presented and investigated in Chap. 3.

In the second step, the classical (online) IP learning is applied to compensate small and slow drifts in the input distributions, e.g. due to the application in non-stationary environments. Drift compensation through IP learning offers a different and novel approach, which internalizes the drift in the network model by sustaining the neural encoding in the hidden-layer without using a continuous error feedback. It turns out that only the combination of a proposed modification of the IP rule and the usage of the natural gradient [15, 16] provides learning dynamics that are well suited for compensation of such drifts. These modifications are introduced and analyzed in Chap. 4.

In the third step, the black-box character of ELMs is approached by incorporation of continuous constraints [17], derived from prior knowledge about the specific task, into the learning. This is reasonable, because machine learning solutions have to guarantee certain properties of the learned function for a safe operation in many application domains. A constraint that is present in the ideal function is of great advantage in order to prevent overfitting. Yet, model selection strategies are entirely specific for these constraints and can not easily be found for complex constraints. However, this task is not trivial because the ELM approach gains its very power from the universal approximation capability. The incorporation of continuous constraints into the learning of ELMs is presentd in Chap. 5.

Finally, The novel ELM approach is applied to three salient and relevant tasks emphasizing the power and generality of the presented approach. Chap. 6 shows

how the ELM approach is applied to control the bionic handling assistant (BHA), such that this novel bioinspired robot is enabled to perform in highly reactive tasks [17]. The major problem is that only sparse data can be provided for learning, due to physical restrictions of the robot. Successful learning is only possible because prior knowledge about the BHA's physical structure is used to formulate continuous constraints that lead to robust generalization results.

Chap. 7 introduces a mechanism to estimate dynamical systems that are used to generate movements applied in the field of humanoid robotics and imitation learning [18, 19, 20]. Data sets in these scenarios are sparse and typically derived from only few demonstrations. Therefore, many state-space regions remain where no information on the dynamics are available. Generalization towards such regions is meaningful only if strong biases are introduced, for instance, assumptions on global stability properties of the to-be-learned dynamics. The special role of stability in this context is thus emphasized and Lyapunov's theory is used to obtain continuous constraints for the learning in order to enforce asymptotic stability of the induced dynamics explicitly. The approach is analyzed in a scenario where trajectories are learned and generalized from human handwriting motions. It is also demonstrated that the novel approach leads to robust movement generation, when learning from robotic data obtained by kinesthetic teaching of the humanoid robot iCub.

The last application is shown in Chap. 8, where the modeling of the ultrasonic softening effect for copper bonding is presented [21]. Copper bonding is a future technology and used as interconnection for individual electrodes. Whereas copper has favorable mechanical and electrical properties for building bond connections, the interaction of the process parameters is largely unknown. Modeling by means of learning seems promising but also difficult since the production of big data sets is yet impossible. It is shown that the presented approach is able to deal with sparse data sets and therefore produces models with good generalization capability. The results presented in this chapter were developed in the InCuB[2] project, which aims at the development of self-optimizing techniques to enable the reliable production of copper bonding connections under varying conditions. Chap. 9 draws conclusions on impact and implications of this work.

In summary, this thesis investigates the reliability of extreme learning machines in three different aspects and shows how the ELM technique can be equipped with the abilities to *learn reliable models* in three highly relevant tasks. This emphasizes the *generality of the approach* and shows that learning is feasible despite that provided data sets are sparse and noisy.

[2]InCuB (Intelligent Copper/Cu Bonding) is a project funded by the German BMBF within the "Intelligent Technical Systems" leading-edge cluster.

Chapter 2

Random Projections and Extreme Learning Machines

Random projection methods are very appealing approaches in the field of artificial neural networks that stand for the advances of fast learning. Novel approaches in this field project the input into a high-dimensional space generating features of the data. Characteristic for random projections is that the feature generation is initialized randomly and remains fixed during the supervised learning phase which leads to fast learning in comparison to the learning via backpropagation of errors through the entire network.

A prominent representative of high-dimensional random projection methods is the extreme learning machine (ELM) which is a recently developed learning scheme for single hidden-layer feed-forward neural networks. The ELM approach is the fundamental basis for the methods developed in this thesis. This particular chapter therefore introduces the original ELM approach and various improvements. Also, a rigorous review of the past random projections literature with its relation to ELMs is presented.

The remainder of this chapter is organized as follows. Sect. 2.1 describes the basic ELM model while Sect. 2.2 explains further developments of the ELM technique. Sect. 2.3 presents a review of the developments of random projection methods starting from linear random projections. Sect. 2.4 summarizes the chapter. Parts of this chapter are based on [12, 14].

2.1 Extreme Learning Machine

The extreme learning machine (ELM) [9] as a representative of random projection methods was recently proposed. The method is appealing because of the high learning efficiency, conceptual simplicity, and the good generalization capability. A discussion on the relation between the ELM approach and earlier proposed feed-forward random projection methods is given in [22] and [23]. A survey on ELM techniques can be found in [24].

Consider the ELM network architecture depicted in Fig. 2.1. The figure shows a single hidden-layer feed-forward neural network comprising three layers of neurons: $\mathbf{x} \in \mathbb{R}^I$ denotes the input, $\mathbf{h} \in \mathbb{R}^R$ the hidden, and $\hat{\mathbf{y}} \in \mathbb{R}^O$ the output neurons. The

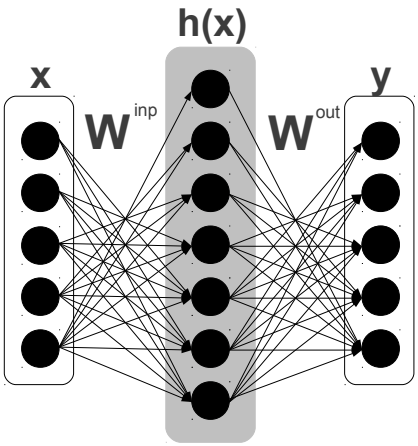

Fig. 2.1: Extreme learning machine with three layer structure. Only the read-out weights are trained supervised.

input is connected to the hidden-layer through the input matrix $W^{\mathbf{inp}} \in \mathbb{R}^{R \times I}$. The read-out matrix is given by $W^{\mathbf{out}} \in \mathbb{R}^{O \times R}$. For input $\mathbf{x}$ the output of the ith neuron is thus given by:

$$\hat{y}_i(\mathbf{x}) = \sum_{j=1}^{R} W_{ij}^{\mathbf{out}} \underbrace{f(\sum_{k=1}^{I} W_{jk}^{\mathbf{inp}} x_k + b_j)}_{\text{hidden-layer: } \mathbf{h}(\mathbf{x})} \ , \tag{2.1}$$

where $i = 1, \ldots, O$, b_j is the bias for neuron j, and $f(x) = \frac{1}{1+e^{-x}}$ denotes the sigmoid activation function called Fermi function applied to each neuron in the hidden-layer component-wise. $\mathbf{h}(\mathbf{x})$ denotes the so-called hidden-layer or hidden state of the ELM. The components of the input matrix and the biases are drawn from a random distribution and remain fixed after initialization.

Let $D = (X, Y) = (\mathbf{x}(k), \mathbf{y}(k))$ with $k = 1 \ldots N_{\text{tr}}$ be the data set for training where N_{tr} is the number of training samples. Supervised learning is restricted to the read-out weights $W^{\mathbf{out}} \in \mathbb{R}^{O \times R}$ and done by minimization of the following training error functional:

$$E_{\text{tr}} = \frac{1}{2N_{\text{tr}}} \| W \cdot H(X) - Y \|^2 \xrightarrow{W^{\mathbf{out}}} \min \ , \tag{2.2}$$

where $H(X) \in \mathbb{R}^{R \times N_{\text{tr}}}$ is the matrix collecting the hidden-layer states obtained for inputs $X \in \mathbb{R}^{I \times N_{\text{tr}}}$ after the application of the non-linearity f, and $Y \in \mathbb{R}^{O \times N_{\text{tr}}}$ is the matrix collecting the corresponding target values. The solution of this functional is given by the least squares solution given by the Moore-Penrose pseudo inverse [25] in computationally cheap fashion:

$$W^{\mathbf{out}} = Y \cdot H(X)^{\dagger} \ , \tag{2.3}$$

where $H(X)^\dagger$ is the pseudo inverse of the hidden-layer matrix. Note that this matrix has the minimal matrix norm among the set of valid solutions.

One of the most attracting properties of the ELM is its ability to approximate universal functions under mild conditions. This feature was rigorously theoretically investigated in [10] and summarized in the following theorem:

Theorem (Universal Function Approximation Capability). *Given any small positive scalar value $\varepsilon > 0$, a non-constant and bounded activation function $f : \mathbb{R} \to \mathbb{R}$ which is infinitely differentiable in any interval, and N_{tr} arbitrary distinct samples $(X, Y) = (\mathbf{x}(k), \mathbf{y}(k))$ where $\mathbf{x}(k) \in \mathbb{R}^I$ and $\mathbf{y}(k) \in \mathbb{R}^O$. Then, there exists a hidden-layer with size $\tilde{N} \leq N_{tr}$ with $\tilde{N}, N_{tr} \in \mathbb{N}$ such that for any $W^{\boldsymbol{inp}}$ and $\mathbf{b}$ randomly chosen from any intervals $\mathbb{R}^{R \times I}$, $\mathbb{R}^R$ according to any continuous probability distribution, the inequality $\|W^{\boldsymbol{out}} \cdot H(X) - Y\| < \varepsilon$ holds with probability one*[1].

The theorem states that the ELM is able to approximate any given data set with targets Y from distinct inputs X with arbitrary small error ε despite the randomness of the high-dimensional projection onto the hidden-layer states. The only restriction is that enough hidden-layer neurons has to be provided before training which is well-known in machine learning as model selection problem.

2.2 Improvements of the Extreme Learning Machine

Despite that ELMs are universal function approximators, neuron model, network size, regularization strength, and, in particular, the features provided by the hidden-layer strongly influence the generalization performance. However, in the classic ELM approach, most of the quantities remain for manual tuning by means of expert knowledge about the specific task. It has been claimed for ELMs that "apart from selecting the number of hidden nodes, no other control parameters have to be chosen manually" ([26], p. 1411). However, it is empirically shown in [12] that this claim is based on the assumption that either very large data sets are used as in [10, 27, 26] or the network size is explicitly chosen to be much smaller than the number of training samples (e.g., in [28], pp. 1355-1356). The theoretical foundation of this phenomenon is based on the empirical risk minimization principle [29]: models such as ELMs that are provided with only few data samples are prone to overfitting of the data. In contrast, training data can be very expensive in practical applications, e.g. in tasks involving robots. It can also be undesirable to limit the hidden-layer size R to a small fraction of the number of training samples N_{tr}, because, then, the network can suffer from a poor approximation capability. Another argument is that the controlling of the network size only affects the num-

[1] This theorem was taken from [10] and modified according to the terminology of this thesis. Some minor changes of this theorem were also made by the author for mathematical consistency.

ber of features rather than their complexity. Effects of regularization, both, in the output learning and the features provided by the hidden-layer encoding are thus ignored. In such cases, appropriate model selection becomes an important issue and remains challenging due to the lack of precise knowledge about the interplay between model complexity, output learning, and performance for ELMs.

Several different techniques to tune the network's parameters towards a given task have been contributed in the recent past. These methods tackle the hidden-layer size, the read-out learning, the encoding in the hidden-layer regarding the features and are discussed in the following sections. Also genetic approaches are also popular due to the fast learning procedure of the underlying ELM approach which is used as an element of a population [30].

2.2.1 Supervised Read-Out Learning and Regularization

Since the reconstruction of a function from a finite set of data is a clearly ill-posed problem, prior knowledge of the function that has to be reconstructed is necessary in order to find a suitable solution. The most commonly used prior knowledge is that the function is smooth, such that similar inputs correspond to similar outputs [31]. Therefore, typical choices for regularization functionals in neural networks punish high curvatures, i.e. strong oscillations, of the attained input-output mapping.

In the original ELM approach, overfitting is prevented by implicit regularization: by either providing a large number of training samples or by using small network sizes. Assuming noise in the data, it is well known that this is equivalent to some level of output regularization [32, 1] punishing big read-out weights $W^{\mathbf{out}}$. It is therefore natural to consider output regularization directly as a more appropriate technique for arbitrary network and training data sizes [33]. As a state of the art method, Tikhonov regularization [34, 35] is used which is also a standard method for reservoir networks[2] [36] and machine learning in general [32]. The assumption of a Gaussian prior for the learning parameters $W^{\mathbf{out}}$ leads to the new error function punishing the growth of the network's read-out weights quadratically. It is flexibly controlled by a regularization parameter $\alpha > 0$ in the error function:

$$E_{\text{tr}} = \frac{1}{2N_{tr}} \|W \cdot H(X) - Y\|^2 + \alpha \|W\|^2 \xrightarrow{W^{\mathbf{out}}} \min \ , \tag{2.4}$$

where $H(X) \in \mathbb{R}^{R \times N_{\text{tr}}}$ is the matrix collecting the hidden-layer states obtained for inputs $X \in \mathbb{R}^{I \times N_{\text{tr}}}$, and $Y \in \mathbb{R}^{O \times N_{\text{tr}}}$ is the matrix collecting the corresponding target values. The solution of this functional is given by ridge or Tikhonov regression [34, 35] in computationally cheap fashion:

$$W^{\mathbf{out}} = Y \cdot H(X)^T \cdot \left(H(X) \cdot H(X)^T + \alpha \mathbb{I} \right)^{-1} \ , \tag{2.5}$$

[2] A Special approach to train recurrent neural networks highly related to learning for ELMs that is more elaborately discussed in Sect. 2.3.4.

where $\mathbb{I} \in \mathbb{R}^{R \times R}$ is the identity matrix. The computation of this solution is, as a side effect, also numerically more stable because of the better conditioned matrix inverse. A suitable regularization parameter $\alpha > 0$ needs to be chosen carefully. Too strong regularization, i.e. too large α, can result in poor performance, because it limits the effective model complexity inappropriately [1]. On the other hand, a too small value of α does not avoid the overfitting of data. This is a typical machine learning model selection problem for the ELM. The parameter α can be determined by line search after definition of a suitable validation set.

2.2.2 Hidden-Layer Size

To match the challenge of model selection, improvements of the ELM have been developed recently that are based on the idea to change the hidden-layer size in order to enhance the generalization ability. This section reviews the most important developments on ELMs searching for a task-specific hidden-layer size providing good performing networks.

The incremental extreme learning machine (I-ELM) is an approach which adds random neurons one-by-one to the initially small hidden-layer until a desired minimum training error is reached. It is shown in [37] that the universal function approximation ability of the ELM is preserved when incrementally updating the size of the hidden-layer. Due to the independence of the hidden-layer neurons, a complete re-computation of the read-out weights is not necessary. The previous solution can be re-used such that learning stays efficient. Also enhanced implementations of the I-ELM exist which choose the neuron-to-add by use of training error criteria from a group of neurons. This approach produces more compact learning architectures with the benefit of faster convergence time [38]. The so-called error minimized extreme learning machine (EM-ELM) [28] also aims at increasing the hidden-layer size while sustaining the universal approximation capability. Instead of adding neurons one-by-one, also groups of neurons can be implemented into the hidden-layer.

A related idea to improve ELMs is named optimally pruned extreme learning machine (OP-ELM) proposed by Miche et al. in [39, 40]. It is based on the original ELM algorithm and extends it by a pruning strategy of the hidden-layer neurons. The OP-ELM methodology can be separated into three main steps: build a big and random hidden-layer, rank the hidden neurons by application of a technique called multi-response sparse regression algorithm, and prune the neurons according to a leave-one-out cross-validation. It was shown that this method leads to favorable performance results. A similar idea which also supplies a pruning strategy for ELMs is proposed in [41]. This method, however, applies statistical criteria (χ^2 and information gain) to select the neurons to prune instead of using a validation set. The recently developed Tikhonov regularized optimally pruned extreme learning machine (TROP-ELM) [42] represents an extension of the already mentioned OP-ELM with an additional L_2-regularization penalty. The approach

applies a cascade of two regularization techniques: a L_1-penalty to rank the hidden-layer neurons is used first, then a L_2-penalty on the read-out weights is applied. This enhances the numerical stability and enables efficient pruning of the neurons. It is shown in [42] that the TROP-ELM approach systematically outperforms the OP-ELM and produces more robust results in terms of standard deviation for different network initializations.

2.2.3 Neural Encoding

In the ELM paradigm, a typical heuristics is to scale the data to $[-1, 1]^I$ and to draw the biases $\mathbf{b}$ and input weights $W^{\mathbf{inp}}$ uniformly from $[-1, 1]^I$. Then, allowing an arbitrary large number R of hidden neurons and a large training set, manual tuning of the input weights or the activation function parameters is not needed, because a random initialization of these parameters is sufficient to create a rich feature set. In practice, the hidden-layer and training set size is limited and the performance does indeed depend on the hyper-parameters controlling the distributions of the initialization at least of the input weights and the biases $\mathbf{b}$. Very small weights result in approximately linear neurons with no contribution to the non-linear approximation capability, whereas large weights drive the neurons into saturation resulting in a binary encoding. This is demonstrated in [12]. Apparently, the choice of scaling matters.

Recently, the relation between ELMs and reservoir networks is investigated in a static reservoir computing setup [43, 12]. Reservoir networks can be interpreted as ELMs with additional recurrent weights in the hidden-layer. Their precise relation to ELMs is discussed in Sect. 2.3.4 One major result is that recurrent connections between the hidden neurons enhances the spatial encoding of static inputs. This result is argued to be caused by the non-linear mixing of features which is favorable for more complex mappings. The experiments are based on considering the cumulative energy content of the hidden state representation. The recurrent connections in the hidden-layer enrich the neural encoding due to the mixing of features and shows that reservoir networks hold a inherently higher dimensional hidden state representation. This is advantageous for the separability of inputs and increases the performance on various regression and classification tasks. Despite this advantage, unfortunately, the analytical properties of the hidden representation are lost due to the convergence process of the networks state. The deep relation between ELM and reservoir computing is also analyzed and improved by other researchers. Reinhart, for instance, introduces an associative version of the ELM in [44], which is very successfully applied in a bi-directional setup to learn forward and inverse mappings at the same time.

Another method for optimization of reservoir networks and ELMs that is very successfully used for neural networks is intrinsic plasticity (IP) [11]. It was developed by Triesch in 2004 [45] as a model for homeostatic plasticity for analog neurons with Fermi-function. The mapping enhancement of static reservoirs via

IP learning is considered as a promising machine learning method and compared to ELMs in [12].

2.3 Review of Random Projection Methods

In the last decades, artificial neural networks (ANN), in particular multi-layer perceptrons (MLP), have gained extensive popularity and the literature is growing day-by-day. Until now, MLPs provide a well-established approach in the field of machine learning and have been used increasingly as a promising modeling tool in almost all areas of research where quantitative approaches are desired such as pattern recognition [1], forecasting [46], economy [47], and medicine [48]. The main advantage of MLPs is that these networks possess the universal function approximation capability, i.e., that a feed-forward network provided with sufficiently many hidden neurons and layers is able to approximate any continuous multivariate function uniformly to a desired degree of accuracy when supported with appropriate activation functions. This was independently proved in 1989 by Cybenko [49], Hornik et al. [50], and Funahashi [6]. In fact, MLPs can approximate functions that are not differentiable in the classical sense, but possess a generalized derivative as in the case of piecewise differentiable functions [51].

The backpropagation (BP) algorithm is one of the most popular supervised learning algorithms for layered networks [52]. Improving the learning speed of BP and increasing the generalization capability of the networks have always played a central role in neural network research. Besides that MLPs trained with BP have demonstrated their potential in many relevant tasks, the training of such networks was nevertheless claimed as the computational "bottleneck" for applications when using large data sets. Moreover, problems such as trapping into local minima, saturation, weight interference, initial weight dependence, and also overfitting, make MLP training difficult. Additionally, it is also very difficult to select parameters like the number of neurons in a layer and the number of hidden-layers in a network, thereby selection decisions for a proper architecture are not easy.

2.3.1 Linear Random Projections.

One proper solution to the issue of high computational complexity is to decrease the number of variables to only the most useful and required ones for the considered problem. Two general ideas can be used: pruning the variables, or combining them, for example, by projection. The problem remains however similar, one has to either select which variables are relevant using a pruning strategy and a criterion, or find a way of combining them appropriately which can be very time taking. Optimizing a projection matrix, for example, with principle component analysis [53] strategies require the calculation and inversion of correlation matrices which might be not feasible for very high-dimensional data. For the matter of projecting to a lower dimensional space, Johnson and Lindenstrauss [54]

have shown that it is possible to embed small sets of points in high-dimensional spaces into spaces of much lower dimension such that distances between these points are preserved with high accuracy. Furthermore, distributions of input patters become more compact after projection into the low-dimensional space [55]. Random projections for dimensionality reduction (see e.g. [56]) are well established and commonly used in the field of machine learning [57, 58]. Early attempts to utilize random projection methods are simply linear, see Fig. 2.2 for illustration. A randomly initialized matrix $W^{\mathbf{inp}}$ is used to project input patterns $\mathbf{x}$ from a high-dimensional space to a lower dimensional representation:

$$\mathbf{y} = W^{\mathbf{inp}}\mathbf{x} + \mathbf{b} \ , \tag{2.6}$$

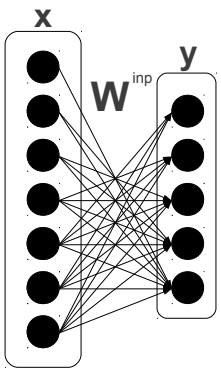

Fig. 2.2: Linear random projection into a lower dimensional space.

where the parameters of the projection $W^{\mathbf{inp}}$ and $\mathbf{b}$ are selected randomly. It was shown that, e.g. clustering techniques that rely on pairwise distances of the data points can then work easily in the projected space [59] and that random projections achieve competitive performances compared to dimension reduction by principle component analysis [53]. Since random projections require no learning, they provide a considerable solution for computation on large and complex data sets.

2.3.2 (Random Vector) Functional-Link Networks.

Novel random projection methods are based on a functional separation between a potentially high-dimensional non-linear feature generating part and the readout function. While the feature generating part of the random projection method is initialized randomly and stays fixed during the supervised learning phase, the learning reduces to a set of linear equations in the adaptable parameters which makes learning very efficient. It is characteristic for such approaches to use very high-dimensional and non-linear projections. These often actually increase the feature dimensionality in contrast to the linear down projections discussed in the previous section.

The first steps towards high-dimensional and efficient random projections were made in the late 80s. Pao introduces the functional-link network (FLN) [60] and provides a rigorous analysis in his popular book about pattern recognition and neural networks, because he realized that "real world tasks require the ability to deal with a very large number of patterns" [61]. A schematic view of the FLN architecture is provided in Fig. 2.3.

From Pao's viewpoint, the successive layers in MLPs carry out a sequence of mappings until an appropriate representation, where it is possible to perform

a desired functionality, is found. He claimed that "it might be possible to enhance the original representation right from the start, in a linearly independent manner, so that hyperplanes for separation can be learned more rapidly" ([61], p. 198). This claim was not solely limited to classification tasks, also regression tasks are solvable by FLNs. The FLN aims at generating a sufficiently enhanced representation of the input pattern such that the associated output can be learned effectively. This network architecture is called flat network because it has no hidden-layers. This may sound a little controversial, since the approximation capability of neural networks is directly associated to the hidden-layer. However, the flat network can be executed without dispensing non-linearity, provided that the input pattern is enhanced with additional higher order information. In other words, higher-order correlations among input components can be used to construct a higher-order network to perform non-linear mappings using only a single layer of units. Pao et al. instantly realized that a promising enhancement of the original representation is to describe it in a space of increased dimensionality. A question arose from that: how to choose additional dimensions?

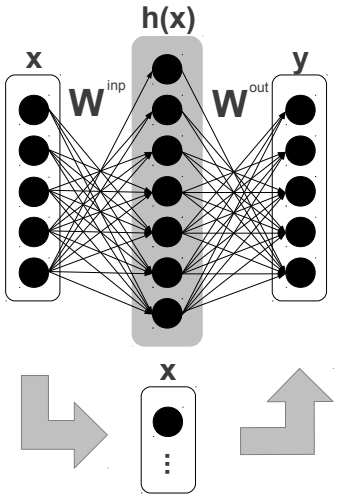

Fig. 2.3: Random vector functional-link (RVFL) net with linear part $\mathbf{x}$ and enhancement nodes $h(\mathbf{x})$.

To answer this question, the FLN follows the idea of enhancing the representation with linearly independent functions equipped with the pattern $\mathbf{x}$ as argument. The functions can, e.g. be a subset of a complete orthonormal basis spanning the n-dimensional representation space, such as the trigonometrical functions: $\sin(\pi\mathbf{x})$, $\cos(\pi\mathbf{x})$, $\sin(2\pi\mathbf{x})$, $\cos(2\pi\mathbf{x})$, and so on; or the polynomial basis: $\mathbf{x}$, $\mathbf{x}^2$, $\mathbf{x}^3$ and so on. Interestingly, those enhancements are inspired by either the Fourier or Taylor analysis. Hence, learning the weights of a flat network is equivalent to solving a system of linear equations. This ability is important in any real world pattern recognition task because learning can be done very efficiently by any linear method. Most importantly, Pao stressed that "pseudo inversion does indeed give a best fit solution [but] that solution is often unacceptable, however, as indicated by the high error value at the end of the process" ([61], p. 206). Also [60] emphasized the use of online learning to adapt the read-out weights. The main reason for this is that the pseudo inverse solely minimizes the quadratic error functional on the training data without any respect to regularization. This fact can lead to overfitting and thus poor generalization which can be avoided by online learning, regularization, appropriate pruning strategies or early stopping.

The random initialization of the input weights and their separation from the read-out learning was proposed for FLNs in [60] and called random vector functional link (RVFL) net. The enhancement of the flat network is substituted by the random hidden-layer of a classical ANN - the read-out function is defined as:

$$\mathbf{y} = W^{\mathbf{out}}(\mathbf{x}|f(W^{\mathbf{inp}}\mathbf{x} + \mathbf{b})) \ , \tag{2.7}$$

where $W^{\mathbf{out}}$ are the read-out weights, $W^{\mathbf{inp}}$ is the collection of input weights for the enhancement neurons, f is the component-wise applied sigmoid activation function, $\mathbf{b}$ are the biases, and $\mathbf{x}$ is the input pattern used as argument mapped onto the output $\mathbf{y}$. The mathematical concatenation $(\mathbf{x}|f(\dots))$ between $\mathbf{x}$ and the random hidden state $f(\dots)$ shows that despite the link between enhancement nodes and output, also a direct connection between input and output is part of the network structure which is, basically, a linear model. The architecture is based on Hornik, Stinchcombe, and White's theorem [50] which proves that the success of single hidden-layer neural networks with sigmoid activation functions is due to their universal approximation capability. The theoretical justification for the RVFL network is given in [62]. It is shown that RVFL nets are indeed universal function approximators for continuous functions on bounded and finite dimensional sets despite the randomness of the enhanced representation and that RVFL are efficient with respect to the error convergence rate. Interestingly, Igelnik et al. also show, by application of a limit integral formulation of the to-be-approximated function and a subsequent Monte-Carlo evaluation method, that similar results are obtained for hidden neurons that are based on products of univariate functions or radial basis functions. Further investigations on learning and generalization characteristics of RVFL nets can be found in [63, 64]. The RVFL and FL nets are still subject to active research - many extensions were developed and applied to various kinds of applications, see [65, 66]. A rigorous survey can be found in [67].

In summary, the main difference between RVFL nets and extreme learning machines is not only the architecture but also the fact that learning without iterative tuning is rather suppressed than explicitly emphasized for RVFL nets.

2.3.3 Radial Basis Function Networks.

Radial basis function (RBF) networks provide an attractive alternative to sigmoid networks for learning real-valued mappings because of two main advantages: first, they provide an excellent approximation capability to smooth functions and, second, the centers of the radial basis functions are interpretable as prototypes of the data samples. The computation of the network's output is determined as:

$$y_i = \sum_{j=1}^{R} W_{ij}^{\mathbf{out}} f_j \left(\|\mathbf{x} - \mathbf{c}_j\| \right) \ , \tag{2.8}$$

where R is the number of center locations c_j, the matrix $W^{\mathbf{out}}$ collects the read-out parameters, and f_j is a non-linear transfer function. The transfer function

is typically Gaussian with covariance matrix Σ_j but also other functions can be applied. The architecture of the RBF network is illustrated in Fig. 2.4 - the input pattern $\mathbf{x}$ is mapped into the projection space through the radial basis functions determined by the centers $\mathbf{c}$ and their covariances Σ.

In parallel to the progress of multi-layer perceptrons represented by functional-link nets, radial basis function (RBF) networks went through a similar exciting development in the late 80s. Broomhead and Lowe deployed the theory of multi-variable interpolation in high-dimensional spaces in [68] to develop a new perspective on RBF networks in which "the nature of the fitting procedure is explicit" ([68], p. 323). This explicit nature of the fitting procedure helped to develop a better understanding of the general properties of layered non-linear networks which perform an equivalent function with a focus on universal approximation [68]. They directly inferred that RBF "generalization is therefore synonymous with interpolation between the data points [...] generated by the fitting procedure as the optimum approximation [...]" ([68], p. 323). In particular, the paper proposes to relax the strict interpolating nature and suggested that "[...] we do not necessarily require that the radial basis function centers correspond to any of the data points" ([68], p. 325), which is the first step towards a random projection for RBF networks. Nevertheless, the baseline performance of the RBF networks in the performed experiments was obtained by assuming fixed non-linearities and choosing the locations of the centers randomly from the training data set. This is based on the assumption that the training data is distributed representatively for the problem. Broomhead and Lowe mentioned that the adjustable weights in the read-out can be determined by linear methods when the centers are selected, which is effectively the same as fixed input-to-hidden-layer weights during supervised learning. Thus, in the least squares context, training the network is equivalent to solving a system of linear equations. By specialization to a minimum norm solution, which is guaranteed by the pseudo inverse technique, the solution exists and becomes unique. In this sense, the network has a "guaranteed learning algorithm" ([68], p. 326) which is at the same time computationally efficient.

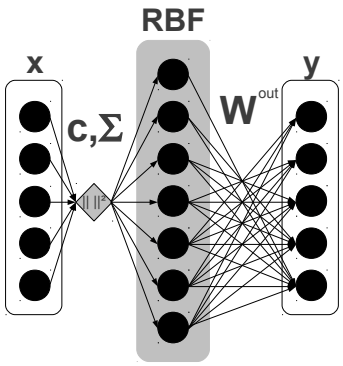

Fig. 2.4: Radial basis function (RBF) network and trainable linear read-out weights.

The main consequence of the generalizations is the need for a mechanism to choose suitable sets of RBF centers and non-linearities. Since the criticism remained that the particular choice of centers and non-linearities influences the performance of the RBF network crucially, Lowe addresses the criticism in [69]. The experimental results in his paper are obtained by adaption of the RBF centers and

the non-linearities by means of a non-linear optimization problem. He claimed that "the initial choice of centers and the parameters of the non-linearity are equivalent to the random start of the weights employed in a multi-layer perceptron prior to training" ([69], p. 171). Learning is thus done subsequently in an iterative manner to minimize the training error until a local minimum is reached. The inclusion of the basis function centers as parameters into the supervised learning process can be carried out by using non-linear optimization techniques in parallel to the linear read-out learning. This can be advantageous, e.g. because, less hidden units are needed, but it certainly adds complexity to the learning due to the performance of non-linear optimization [70]. In this context, Poggio et al. suggests in [70] to use gradient descent based methods to implement supervised learning of the center positions, a method that they call generalized radial basis functions.

Although the application of linear learning methods for the read-out weights and unsupervised methods to learn the center locations yields very efficient training, researchers reported inferior results when compared to sigmoid backpropagation networks. For instance, Moody and Darken's RBF network needs 10 times more training samples than a standard sigmoid network in order to reach a comparable generalization performance on the Mackey-Glass time-series task [71]. Wettschereck et al. "identified several plausible explanations for this performance gap" ([72], p. 1132): first, all parameters in standard sigmoid networks are determined by supervised learning through backpropagation, whereas the supervised learning in typical RBF networks is restricted only to the output weights and second, the application of the Euclidean distance for computation of the basis function argument assumes that all input features are equally important. This assumption is known to be false in many applications. However, one of the main difference between MLPs and RBF networks is that MLPs construct global approximations while RBF networks construct local approximations of input-output mappings due to their exponentially decaying localized non-linearities (e.g. Gaussian functions) [5]. This implies that RBF networks do not suffer from destructive interference during learning but also have no extrapolation ability as MLPs. Therefore, unsupervised adaptation of the randomly initialized feature generating part is indispensable. In summary, the idea of a functional separation between feature generation and linear read-out learning is indeed established for RBFs but it is not common to apply completely random hidden nodes without subsequent unsupervised adaption which is different to commonly used derivatives of the extreme learning machine approach.

2.3.4 Reservoir Computing

Recurrent neural networks (RNNs) are promising tools for application in non-linear time series processing. One reason is that RNNs are universal approximators of dynamical systems [73]. However, the application of RNNs in challenging modeling tasks was limited for a long time, despite the great potential and several

successful applications. The main reason for this is that training of RNNs is usually achieved by gradient descent methods like backpropagation trough time [74] or real time recurrent learning [75] which is problematic for several reasons. RNNs can potentially implement complex spatio-temporal dynamics that are prone to bifurcations [76] where the gradient information degenerates. In general, convergence of the learning cannot be guaranteed. Already single parameter updates are computationally expensive and with the additional need for many cycles in the learning process this results in long training times [77]. Tasks that require long-range memory are challenging, because the gradient information exponentially disappears [78]. Advanced training algorithms for RNNs need substantial experience of the user for successful application.

In recent years, a different approach to use RNNs became popular: the reservoir computing (RC) approach [77, 12, 79], a paradigm to use recurrent neural networks with fixed and randomly initialized recurrent weights. Fig. 2.5 visualizes the typical setup of a reservoir network. The dynamics of the reservoir and the output function can be calculated according to the following equations:

$$\begin{aligned} \mathbf{h}(k+1) &= f\left(W^{\mathbf{inp}}\mathbf{x}(k) + W^{\mathbf{rec}}\mathbf{h}(k)\right) \\ \mathbf{y}(k) &= W^{\mathbf{out}}\mathbf{h}(k) \ , \end{aligned} \tag{2.9}$$

where $\mathbf{h}$, $\mathbf{x}$, $\mathbf{y}$ denotes the hidden state, the input, and the read-out layer respectively. f denotes a non-linear transfer function and W^* ($* = \mathrm{inp}, \mathrm{rec}, \mathrm{out}$) the respective weights for the input, the recurrent reservoir, and the output weights of the network. From a machine learning point of view, the reservoir serves as a fixed spatio-temporal kernel projecting the input data non-linearly into a high dimensional space of the reservoir network states. In the limit of infinitely many neurons, this is equivalent to a recursive kernel transformation [80]. The subsequent use of a trainable non-recurrent linear readout layer $W^{\mathbf{out}}$ combines the advantages of recurrent networks with the ease, efficiency and optimality of linear regression methods. New applications for processing temporal data have been reported, for instance in speech recognition [81, 82], sensori-motor robot control [83, 84, 85, 86], detection of diseases [87, 88], or flexible central pattern generators in biological modeling [89].

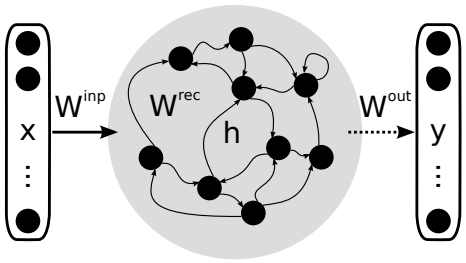

Fig. 2.5: Echo state network (ESN) without output feedback. The hidden-layer neurons are connected via recurrent weights.

The first ideas towards RC were independently introduced by Maass et al. under the notion of liquid state machines [90] and by Jaeger as echo state networks [91]. The learning of these networks is done by harvesting the reservoir

states during application of the training data followed by linear regression to learn the read-out weights. In [92, 93], Schiller and Steil realized that the functional decomposition between dynamic reservoir and read-out layer is an inherent property of networks that are trained with the constrained optimization method by Atiya and Parlos [94]. The weight dynamics show that the internal recurrent connections adapt slowly in comparison to the weights connecting the reservoir with the read-out layer [93]. This natural separation was used to develop an interesting learning scheme also restricted to the read-out weights called backpropagation decorrelation [95, 96, 97]. In this sense, random projections into a high-dimensional spatio-temporal space through a dynamic reservoir is a natural means to look at neural network approaches. A more detailed discussion on reservoir computing approaches and their relation to random projections with additional consideration of output feedback can be found in [44].

2.3.5 Miscellaneous Random Projection Methods.

Various other researchers anticipated the importance of fast and robust learning of artificial neural networks based on linear least-squares methods as well. However, the functional separation between input and output mapping in these models is not explicit: either the input weights are learned supervised or the error-signal is propagated trough multiple layers. It is still worth it to mention these methods.

In [98], an algorithm for the training of MLPs solely based on linear algebraic methods is presented. Besides that the convergence speed of the algorithm is up to orders of magnitude higher than that of back-propagation, all weights connecting the hidden-layers are adapted without following the idea of a functional separation. Yam et al. [99] proposed a method based on linear least squares to initialize feed-forward neural networks. Despite the partition of input and output layer, the initialization method is supervised and compared to random initializations. Specifically, in [100], a method for learning a single layer neural network by solving a linear system of equations is proposed. This method also learns the inverse non-linear transfer functions and avoids the problem of local minima. This method is also used in [101] to learn the last layer of a neural network, while the rest of the layers are updated employing any other non-linear algorithm (for example, conjugate gradient). Castillo extended his method [100] in [102], although in this case multiple layers are learned by using a system of linear equations. Recently, "re-inventions" of elder random projection models are proposed in [103, 104].

2.4 Summary

Random projection methods have already demonstrated their great potential in several different applications for two main reasons. On the one hand, the initialization of random projections is simple because many parameters are initialized randomly and stay fixed during supervised learning. On the other hand, learning

is typically restricted to the read-out weights and is thus very efficient due to the use of linear optimization techniques. This is caused by the functional separation of the mapping parameters before and after the non-linearity which essentially simplifies the learning procedure.

Recently, improvements in the acquisition of data through general purpose computers made it possible to obtain large scale data sets [8]. More abundant information in terms of data set size and dimensionality gradually increased the required computational complexity. In 2004, Extreme learning machines became popular, because of the demand for simple models with efficient learning procedure.

In fact, ELMs are successfully used in several applications, but many aspects of the learning parameters are not considered in the original contribution. The neuron model, network size, regularization strength and in particular the features provided by the hidden-layer strongly influence the generalization performance but often remain fixed during learning. One reason is that it has been claimed for ELMs that the random projection is already sufficient for a successful approximation of data set [26]. This is not true, because further studies showed that this claim is based on the (strong) assumption that either very large data sets are used [12] or the network size to number of training samples ratio is chosen to be very small [28]. In cases where this assumptions are violated (which is the rather common case), appropriate model selection and optimization becomes an important issue. Several interesting improvements on the ELM approach are discussed in this chapter, but model selection still remains challenging due to the lack of precise knowledge about the interplay between model complexity, output learning, and performance. This thesis therefore provides an approach to feature reliable learning with ELMs.

Despite the great success of ELMs in various applications by many researchers, the original contribution of the ELM approach with respect to earlier proposed random projection methods is discussed controversially (see [22] and [23]). In this chapter, the most important developments in the field of random projections is pointed out and also the chronological order of the original contributions is emphasized. Also the relation to ELMs and random projections in general is discussed in detail in this chapter. Tab. 2.1 states a review of random projection methods and their relations listed in tabular form. In my opinion, the ELM approach is the only random projection method which explicitly emphasizes the importance of the high-dimensional data representation, the simplicity of the model (solely through random projection), and the fast learning procedure through batch learning. However, Huang missed - in his first contributions about ELMs ([9, 10]) - to discuss the relation of the ELM approach to earlier proposed FLN and RBF networks. It is obvious that this is problematic since the original contribution of ELM was not entirely clear at this point. I think that one major problem of the actual ELM literature is the lack of a comprehensive survey and also a systematic taxonomy summarizing the improvements of ELMs. [24] and [105] are attempts to solve this issue but they only cover specific parts of the literature.

Tab. 2.1: Overview: random projection methods.

Method	**Mapping**	**Projection**	**Learning/Tuning**	**Application**	**Citations**
Linear random projection	Random, analytical, linear, spatial, global	Low-dimen.	No learning	Dimensionality reduction	[56, 57, 58]
Functional link net	Deterministic, analytical, linear and non-linear, spatial, global	High-dimen.	features (selection), read-out (linear, online)	regression, classification	[60, 61, 67]
Random vector functional link net	Random, analytical, linear and non-linear, spatial, global	High-dimen.	Features (random, unsupervised), read-out (linear)	regression, classification	[60, 62, 65]
Radial basis function net	Random, analytical, non-linear, spatial, local	Low or high-dimen.	Features (unsupervised), read-out (linear)	Regression, classification, probability density estimation	[68, 69, 70]
Reservoir computing	Random, non-analytical, non-linear, global, spatio-temporal or static	Rich dynamics, low or high-dimen.	Features (random, unsupervised), read-out (linear)	Regression, classification, sequence transduction, prediction	[77, 91, 95]
Extreme learning machine	Random, analytical, non-linear, spatial, global	Low or very high-dimen.	Features (random, unsupervised), read-out (linear, batch)	Regression, classification	[9, 10, 12]

Chapter 3

Robustness to Initializations by Batch Intrinsic Plasticity

The last decade showed that the extreme learning machine (ELM) successfully allows very efficient learning with good generalization. However, it is obvious that the task performance and the generalization ability is depending on the random projection. To enhance the robustness to initializations and different input data distributions is therefore an important ingredient for reliable learning. Batch intrinsic plasticity (BIP), a highly efficient unsupervised learning scheme, addresses this issue by adaptation of the activation functions of the hidden-layer neurons such that certain desired output distributions are achieved.

The remainder of this chapter is organized as follows. The first section of this chapter, Sect. 3.1, integrates unsupervised learning by BIP into the context of reliability. The conversion of intrinsic plasticity (described in Sect. 3.2) towards a batch method suited for the prerequisites of the ELM approach is described in Sect. 3.3. Sect. 3.4 analyzes the impact of BIP on a single neuron model statistically. Different studies on the performance of networks trained with BIP are provided by Sect. 3.5.

BIP was first introduced in [13] and further analyzed in [14]. A deeper analysis of intrinsic plasticity and its synergies with recurrence in the context of reservoir computing and random projections is provided in [12]. It is explained that intrinsic plasticity is comparable to a feature-regularization mechanism and therefore enhances the generalization capability.

3.1 Reliability and Robustness to Initializations

"The task performance of the ELM approach strongly depends on the random initialization". This is indeed a fact that one has to face when applying ELMs. One way to deal with it, is to tune the ELM parameters towards a task-suitable regime which can lead to good generalization results (e.g. see [10, 106]). However, this approach is clearly heuristic. In fact, automatic methods to tune the random initialization of the ELM approach in order to avoid overfitting are desired, i.e. an appropriate optimization of the input weights and the biases of the non-linear transfer functions.

One reason for the high dependence on the random initialization is that, without expert-tuning by means of additional task knowledge, a random initialization can lead to the problem of saturated or constant neurons. This can be avoided by finding activation functions which are in a favorable regime. Methods only controlling the network size and the output matrix are insufficient in tuning the neurons to a good regime, where the encoding is optimal. This motivates techniques to tune the output distributions of hidden-layer neurons in a regime where the encoding is suited for good generalization.

To achieve this goal, a biologically plausible mechanism first introduced by Triesch [45] under the notion of intrinsic plasticity (IP) can be used. It is shown in [11], that IP enhances the encoding of recurrent neural networks in the field of reservoir computing. The output of a respective hidden-layer neuron is forced by IP to approximate an exponential distribution. This maximizes the network's information transmission, caused by the high entropy of the distribution. IP can, in principle, be applied for ELMs as well, but counteracts the idea of computationally cheap one-shot learning.

Therefore, batch intrinsic plasticity (BIP) [13] is introduced, which is a highly efficient batch version of intrinsic plasticity. This approach is used to optimize the hidden-layer of ELMs. BIP is applied in a pretraining phase which adapts the activation function of the hidden-layer neurons analytically by a pseudo inverse technique such that desired output distributions are achieved. It is shown in [13, 14] that BIP mitigates the dependence of the performance on the random initialization.

This chapter shows that ELMs pretrained by BIP are significantly more robust to random initializations and achieve better generalization results than purely random ELMs on different real world tasks. The approach is also compared to other state of the art optimization techniques for ELMs.

3.2 Intrinsic Plasticity

Intrinsic Plasticity (IP) was developed by Triesch in 2004 [107, 45, 108] as a model for homeostatic plasticity for analog neurons with Fermi-function. Since its introduction, the IP-rule has been used to learn sensory representations [109] and to enhance the encoding of reservoir networks [11, 36]. Also static reservoirs are considered as promising machine learning methods [12]. Triesch also shows in [108] that synergies of synaptic plasticity and IP can detect heavy tail input distributions.

The goal of IP is to optimize the information transmission of a single neuron strictly locally by adaptation of slope a and bias b of the non-linear activation function $f(x) = \frac{1}{1+e^{-ax-b}}$ (Fermi-function) such that the neurons' outputs become exponentially distributed. IP-learning can be derived by minimizing the Kullback-Leibler-divergence $D(f_x, f_{\exp})$ between the output f_x and an exponential

distribution f_{exp}:

$$D(f_x, f_{\text{exp}}) = \int_\Omega f_x(x) \log\left(\frac{f_x(x)}{f_{\text{exp}}(x)}\right) = -\mathbb{H}(x) + \frac{1}{\mu}\mathbb{E}(x) + \log(\mu) \ , \tag{3.1}$$

where $\mathbb{H}(x)$ denotes the entropy and $\mathbb{E}(x)$ the expectation value of the output distribution. In fact, minimization of $D(f_x, f_{\text{exp}})$ in Eq. (3.1) for a fixed $\mathbb{E}(x)$ is equivalent to entropy maximization of the output distribution. The following online update equations for slope and bias - scaled by the step-width η_{IP} - are obtained:

$$\Delta a = \frac{\eta_{\text{IP}}}{a} + y\Delta b \qquad \Delta b = \eta_{\text{IP}}\left(1 - \left(2 + \frac{1}{\mu}\right)x + \frac{1}{\mu}x^2\right) \ . \tag{3.2}$$

The only quantities used to update the neuron's non-linear transfer function are x, the synaptic sum arriving at the neuron, its squared value x^2 , and the firing rate μ. Since IP is an online learning algorithm, training is organized in epochs: for a predefined number of training epochs the network is fed with the entire training data and each hidden neuron is adapted to the network's current input separately. In fact, for small mean values, i.e. $\mu \approx 0.2$, the neuron is forced to respond strongly only for a few input stimuli.

3.3 Batch Intrinsic Plasticity: Methodology

Since batch intrinsic plasticity (BIP) is an unsupervised training algorithm, only the set of inputs $\mathbf{u} = (\mathbf{u}(1), \dots \mathbf{u}(N_{\text{tr}})) \in \mathbb{R}^{I \times N_{\text{tr}}}$ are used for optimization. The goal is to adapt slope a_i and bias b_i of the activation function such that a desired distribution f_{des} for the neuron's outputs $h_i(k) = f(a_i s_i(k) + b_i)$ is realized. The main idea is to formulate a linear regression problem instead of online learning. Linear regression problems can be solved by computing a pseudo inverse of a matrix which features efficient unsupervised learning. Random virtual targets $\mathbf{t} = (t_1, t_2 \dots t_{N_{\text{tr}}})^T$ are drawn in ascending order $t_1 < \dots < t_{N_{\text{tr}}}$ from the desired output distribution f_{des}. Fig. 3.1 illustrates BIP schematically: the targets

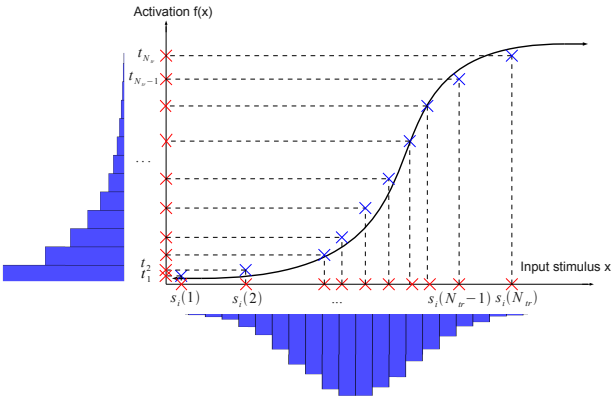

Fig. 3.1: Batch intrinsic plasticity formulated as a regression problem.

$\mathbf{t}$ and the input stimuli $\mathbf{s}_i$ are combined to build data pairs inducing a regression problem. The synaptic sum arriving at neuron i when stimulated by training sample k is given by $s_i(k) = \mathbf{W}_i^{\text{in}}\mathbf{u}(k)$ and collected in $\mathbf{s}_i = \mathbf{W}_i^{\text{in}}\mathbf{u}$. Since the stimuli need to be mapped onto the right targets, a rearrangement of the stimuli in ascending order $s_i(1) < \cdots < s_i(N_{\text{tr}})$ is done by sorting $\mathbf{s}_i \leftarrow \text{sort}(\mathbf{s}_i)$. This is necessary because a monotonically increasing activation function f is used to map all incoming training stimuli on the right targets and infer the desired distribution f_{des} for the neuron's output. Defining the data matrix $\Phi(\mathbf{s}_i) = \left(\mathbf{s}_i^T, (1 \dots 1)^T\right)$ and the parameter vector $\mathbf{v}_i = (a_i, b_i)^T$, learning for the i-th neuron is formulated as a linear and over-determined regression problem, where the outputs are mapped onto the targets $h_i(k) \approx t_k$:

$$\|\Phi(\mathbf{s}_i) \cdot \mathbf{v}_i - \mathbf{f}^{-1}(\mathbf{t})\| \rightarrow \min \quad . \tag{3.3}$$

The solution for the optimal slope a_i and bias b_i is obtained by computation of the Moore-Penrose pseudo inverse [25]:

$$\mathbf{v}_i = (a_i, b_i)^T = \Phi^{\dagger}(\mathbf{s}_i) \cdot \mathbf{f}^{-1}(\mathbf{t}) \quad . \tag{3.4}$$

Typically Fermi and tanh functions are used as activation functions. The algorithm is always terminating, since the pseudo inverse always exists. It is also important that the virtual targets are in $[0, 1]$ (if the Fermi function is used). For this reason, only truncated probability distributions are applied. The learning is done in an one shot fashion and summarized in Alg. 1.

Algorithm 3.1 Batch intrinsic plasticity (BIP) for ELMs

Require: get inputs $u = (u(1), u(2) \dots u(N_{\text{tr}}))^T$
  for all hidden neurons i **do**
    get stimuli $s_i = W_i^{\text{in}} \cdot u$
    draw targets $t = (t_1, t_2 \dots t_{N_{\text{tr}}})^T$ from desired distribution f_{des}
    sort targets $t \leftarrow \text{sort}(t)$ and stimuli $s_i \leftarrow \text{sort}(s_i)$
    build $\Phi(s_i) = \left(s_i^T, (1 \dots 1)^T\right)$
    calculate (pseudo-)inverse $(a_i, b_i)^T = v_i = \Phi(s_i)^{\dagger} \cdot f^{-1}(t)$
  end for
  return $v = (v_1, v_2 \dots v_R)^T$

The pretraining is of the same order of complexity as the supervised read-out learning for ELMs, since only the least squares solutions of the linear model Φ have to be calculated. Only the virtual targets t_i used as meta-parameters are necessary to fully define the regression model. Since the method is unsupervised, Alg. 3.1 has to be recomputed if new input samples are used, e.g. when changing the task. The following section Sect. 3.4 investigates the behavior of BIP when stimulated with different inputs.

3.4 BIP and Single Neuron Behavior

A single neuron model with different fixed input distributions f_s is considered in order to illustrate the behavior of BIP learning. $N_{\text{tr}} = 50$ samples are used for training and $N_{\text{te}} = 1000$ samples for testing - both drawn from f_{s}.

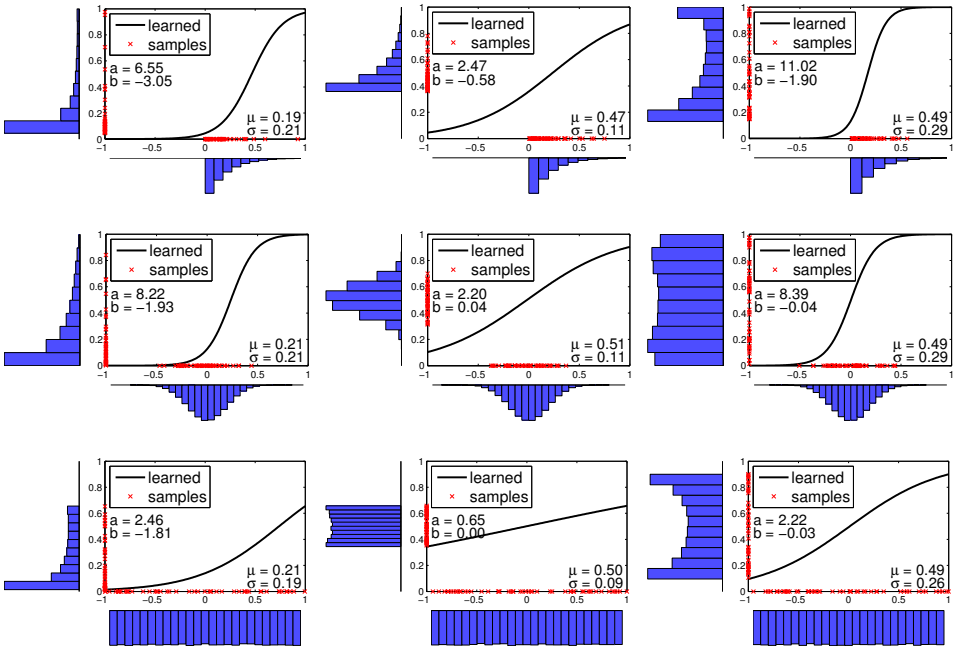

Fig. 3.2: A neuron's activation function adapted by BIP to approximate the output distributions f_{des} (y-axis) while starting from the input distributions f_{s} (x-axis). The input distribution is varied over the rows, while the output distributions varies column-wise.

Three different input and output distributions are taken into account: $f_{\text{des}} = f_{\text{s}} =$ exp(onential), norm(al), and uni(form). The moments of the distributions are: $\mu(\text{exp}) = 0.2$, $\sigma(\text{exp}) = 0.2$, $\mu(\text{norm}) = 0.5$, $\sigma(\text{norm}) = 0.1$, $\mu(\text{uni}) = 0.5$, and $\sigma(\text{uni}) = 0.3$. These values are used for the following experiments.

Fig. 3.2 illustrates the result of adapting the neuron's non-linear transfer function. The input distribution is assigned to the rows of the figure, while the desired output distribution is given column-wise. The incoming training stimuli are visualized by the crosses on the x-axis, while the corresponding targets are displayed on the y-axis. The x-axis shows a histogram of the synthetically created test stimuli while the y-axis shows a histogram of the outputs produced by the learned activation function transforming the inputs. Especially when stimulated with Gaussian input, the neuron is able to achieve the three desired output distributions very accurately - illustrated by the second row in Fig. 3.2. It is demonstrated in the

χ^2 μ σ	exp	norm	uni
	.49±.36	.98±1.04	1.83±.37
exp	.18±.02	.49±0.01	0.46±.04
	.21±.03	.08±0.01	0.25±.02
	.08±.06	.05±0.04	0.27±.11
norm	.20±.02	.50±0.01	0.49±.04
	.19±.02	.09±0.01	0.29±.01
	.27±.11	.25±0.09	1.14±.13
uni	.19±.02	.49±0.01	0.49±.03
	.18±.02	.09±0.01	0.31±.01

Tab. 3.1: Fits of output distributions. A cell contains mean and standard deviation of the χ^2-value, μ and σ.

rnd	lg α=-15	−12	−9
BIP			
R=	.062±.003	.062±.003	.060±.002
50	.062±.004	.063±.004	.059±.002
100	.094±.034	.093±.032	.077±.017
	.073±.014	.072±.013	.061±.002
150	.149±.076	.148±.076	.107±.042
	.073±.013	.073±.013	.062±.003
200	.229±.160	.227±.158	.153±.085
	.075±.015	.075±.015	.062±.003

Tab. 3.2: Test errors on the figure-eight robotics task. Comparison of randomly initialized and BIP pretrained ELMs.

first column of Fig. 3.2 that the exponential distribution is approximated for all inputs. However, since the sigmoid activation function has only two degrees of freedom, the match is typically not perfect. The figure shows that large deviations from the optimal output distribution can sometimes be observed.

Further statistics are summarized in Tab. 3.1. The table shows the results obtained for one neuron which is trained by BIP for 100 trials. After each trial, the mean and the standard deviation of the output distribution are collected which determines the deviation of samples from the desired distribution. The table shows, that μ and σ of the output distribution are always approximated very well with low variance.

3.5 Experimental Results

This section contains the results obtained for the experiments involving optimization by batch intrinsic plasticity. Sect. 3.5.1 shows a robotic scenario where the network learns the inverse kinematics of a six degrees of freedom (DOF) robot arm. Corke's tool box [110] is applied for simulation of the robot. Sect. 3.5.2 investigates the interpolation and extrapolation behavior of ELMs pretrained with BIP and Sect. 3.5.3 demonstrates the network abilities for real world regression tasks.

3.5.1 Figure-Eight with a 6-DOF Robot

The following two sections describe experiments, where the networks' input matrix components W_{ij}^{in} and the biases b_i are drawn from an uniform distribution in the interval $[-10, 10]$, while the slopes a_i are set to unity. In the experiments,

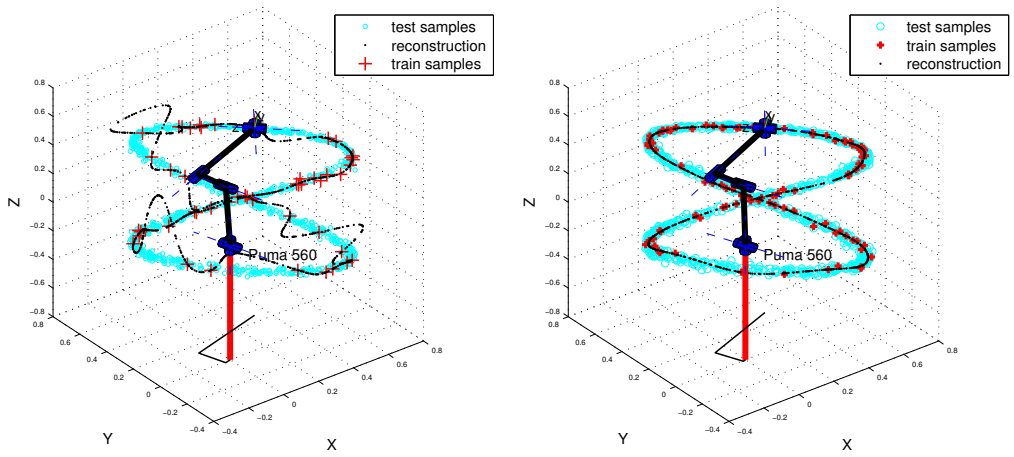

Fig. 3.3: Robotics task for ELMs with: $R = 150$ and $\varepsilon = 10^{-12}$. Left: performance of the randomly initialized ELM. Right: performance of the ELM which was first trained with the BIP method.

the Fermi-function is used as activation function and the desired output is the truncated exponential distribution $f_{\text{des}} = f_{\text{exp}}$ with a fixed mean $\mu = 0.2$.

The network models are applied to learn the observed inverse kinematics mapping between joint and task space of the redundant six degrees-of-freedom (DOF) Puma560 robot arm shown in Fig. 3.3. $N_{\text{tr}} = 100$ training samples are generated by projecting a task trajectory specified in Cartesian end-effector coordinates into the joint space of the robot arm by means of the analytically calculated inverse kinematics function $F : \mathbb{U} \to \mathbb{Y}$, where $\mathbb{U}$ is the task and $\mathbb{Y}$ the joint space. For each task space input $(u_1(k) \dots u_6(k))^T$ containing the end-effector position and orientation the six-dim target vector $(y_1(k) \dots y_6(k))^T$ is computed and additionally corrupted with Gaussian noise ($\sigma_N = 0.1$). The generated trajectory forms an eight - see Fig. 3.3. The left plot images the learned inverse kinematics for a randomly initialized ELM, which apparently overfits the data. The right plot shows the result of the supervised learning for an ELM which was first trained with BIP. The learned part of the inverse kinematics is approximated very well.

Additionally, $N_{\text{te}} = 1000$ test samples are created to verify the generalization capability for different hidden-layer sizes R and output regularization strengths ε. The results of the experiments are summarized in Tab. 3.2 and done for 10 different ELMs and 10 different data set configurations for each cell. The results show that the ELMs trained with BIP perform significantly better than the randomly initialized networks over the whole range of the parameters. Even ELMs with a big hidden-layer and low output regularization (e.g. with $R = 200$, $\varepsilon = 10^{-15}$) do not tend to overfit the data after BIP-pretraining. Also the variance in the performance is much less after pretraining, a robust solution from the learning can

be guaranteed.

3.5.2 Two-Circles learned with a 6-DOF Robot

The following section describes why the shaping of output distributions of the hidden-layer neurons is important and analyzes its influence on the networks' performance. The mapping abilities of the BIP pretrained ELMs is investigated on a further robotics tasks where the scaling of the neurons' input weights is artificially tuned to undesired regimes. It is additionally analyzed how strong the results of BIP depend on the random initialization of the input matrix and the biases.

The hidden-layers have $R = 200$ neurons, the parameterized Fermi-function is used as activation function and the desired output used to produce the targets for the BIP learning is again the truncated exponential distribution $f_{\text{des}} = f_{\text{exp}}$ with a fixed mean $\mu = 0.2$. The networks for this robotics task had a ridge regression parameter of $\alpha = 10^{-5}$. The regularization parameter was identified as parameters leading to the lowest test error by line search in the range $\alpha \in [10^{-9}, 10^{-8}, \ldots, 10^{3}]$.

Again, the task is to learn the inverse kinematics mapping between joint and task space of a redundant six degrees of freedom (DOF) robot arm shown in Fig. 3.4. $N_{\text{tr}} = 200$ training samples and $N_{\text{tr}} = 2000$ test samples are generated and additionally corrupted with Gaussian-noise ($\sigma_N = 0.1$). The generated trajectory forms four circles where only the second and third circle is designated for training and the first and the fourth cycle for testing - see Fig. 3.4.

The figure shows the results for ELMs which hidden-layer neurons are tuned into a specific regime in order to clearly extract the effect of BIP. The figure is separated into two rows. The first row illustrates the robotics task and the respective reproduction performance. The second row shows the output distributions of two randomly selected hidden-layer neurons for the three different examples respectively. The left part of the figure demonstrates a situation where the neurons are saturated. This was achieved by drawing the input matrix weights and biases uniformly from $[-10, 10]$. The first two images in the second row show that the neurons in this case have an almost binary coding. The robot arm is supposed to follow an circular trajectory which is not approximated accurately. The extrapolation ability of the ELM is poor, strong overfitting occurs. Fig. 3.4 (right) shows the effect of hidden-layer neurons with almost constant activation functions producing peak-like output distributions. The input weights and biases are drawn from $[-0.1, 0.1]$. Therefore, the complexity of the network is to low and the mapping can not be approximated. The centered plots illustrate the mapping results for an ELM which was trained by BIP after the random initialization in a saturated or constant regime. The hidden-layer neurons approximate exponential distributions with fixed mean $\mu = 0.2$ illustrated by the third and the fourth histogram in the second row of Fig. 3.4 - the network performs well. The shaping of the output distribution clearly improves the generalization ability of the network. Note, that the desired distribution is not always produced with high accuracy (see

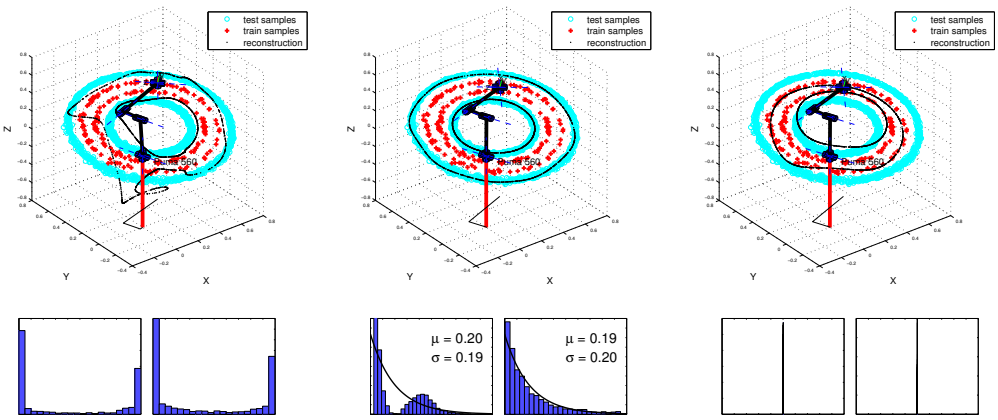

Fig. 3.4: The effect of saturated (left) and almost constant hidden-layer neurons (right) on the robotics interpolation task (RoboInter). BIP produces desired output distributions favorable for generalization (center).

third plot in the second row of Fig. 3.4). However, mean and standard deviation are approximated very well. These example demonstrates that the shaping of output distributions for the hidden-layer neurons separates into two effects: weights which are too large or too small for the given task are scaled and the encoding in the hidden-layer is sparsified due to the exponential distribution. The combination of these effects leads to a good generalization ability.

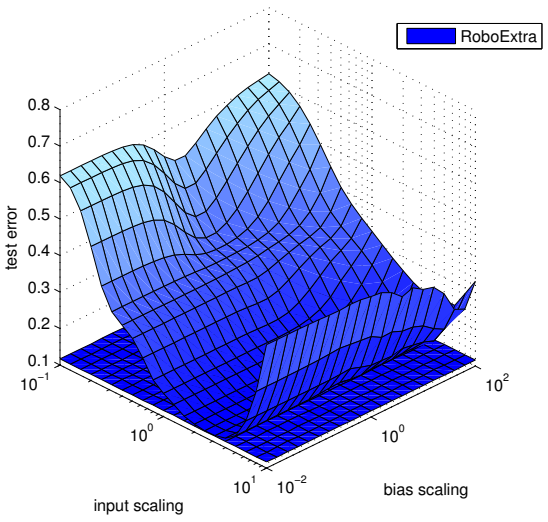

Fig. 3.5: Performance on the test set for different input matrix and bias scalings.

Fig. 3.5 shows how the test error measured as root mean square error changes when the input matrix W^{in} and the biases b are scaled with a factor. After scaling the input matrix and the biases, one part of the networks is trained with BIP. The test error for the randomly initialized ELM highly depends on the initialization indicated by the blue error surface which has a minimum for an input scaling of 1.25 and a bias scaling of 10^{-2}. The networks trained with BIP perform very well

independently of the initialization. BIP compensates the scaling of the incoming stimuli such that a good generalization capability occurs indicated by a flat error surface below the one for the randomly initialized ELMs.

3.5.3 Batch Intrinsic Plasticity vs. Random Initializations

The following section describes experiments focusing on the impact of BIP and ridge regression on the task performance on different real world tasks. The input matrix components W_{ij}^{in}, the biases b_i for a respective ELM are drawn from the uniform distribution in $[-1, 1]$. The Fermi-function $f(x) = 1/(1 + e^{-a_i x - b_i})$ is again used as activation function for the networks and the desired output producing the targets for the BIP learning is again the truncated exponential distribution $f_{\text{des}} = f_{\text{exp}}$ with a fixed mean μ separately drawn for each neuron from the uniform distribution $\mu \in [0.05, 1]$. BIP is applied after the random initialization of the networks as pretraining method. The results are averaged over 20 different network initializations for each data set. The statistics over the networks is contained in the following tables. The optimal ridge regression parameter and network size is determined by a line search.

The data sets used for investigation are: Abalone (Ab), CaliforniaHousing (Ca), CensusHouse8L (Ce), DeltaElevators (De), ComputerActivity (Co), and Elevators (El). All data sets are regression tasks from the UCI machine learning repository - detailed information can be found in [111]. The separation of the data sets into training and test data is predefined as specified in the UCI repository for the respective tasks. The input data has been normalized to $[-1, 1]$, while the output data was normalized into the range $[0, 1]$. The results of the learning for three of the tasks are illustrated in Fig. 3.6. The first row shows the development of the test error for growing hidden-layer size when trained with linear regression. The randomly initialized ELMs strongly overfit the training data when the network becomes large. This effect is significantly reduced when pretrained with BIP. Whereas the effect is present for all data sets, I show only three for illustration in Fig. 3.6. The second row in Fig. 3.6 displays the development of the test error when changing the ridge regression parameter for networks with $R = 500$ hidden neurons. The networks' show a typical generalization behavior where the models first overfit the data, reach an optimal value and then degenerate for too strong regularization. The BIP-pretrained ELMs perform better than the randomly initialized networks in a large range of output regularization parameters and are by far less sensitive to its choice. The results for the best performing ELMs are summarized in Tab. 3.3 and Tab. 3.4. Tab. 3.3 contains the performance results on the test sets for randomly initialized ELMs and BIP-trained ELMs with linear regression. The results are given for the best obtained hidden-layer size which was varied in $R \in [5, 10, \ldots, 500, 1000]$. The BIP-pretrained ELMs perform better than the purely random initialized networks for the majority of the tested regression tasks and lead to stable results with low variance. The results also show that the

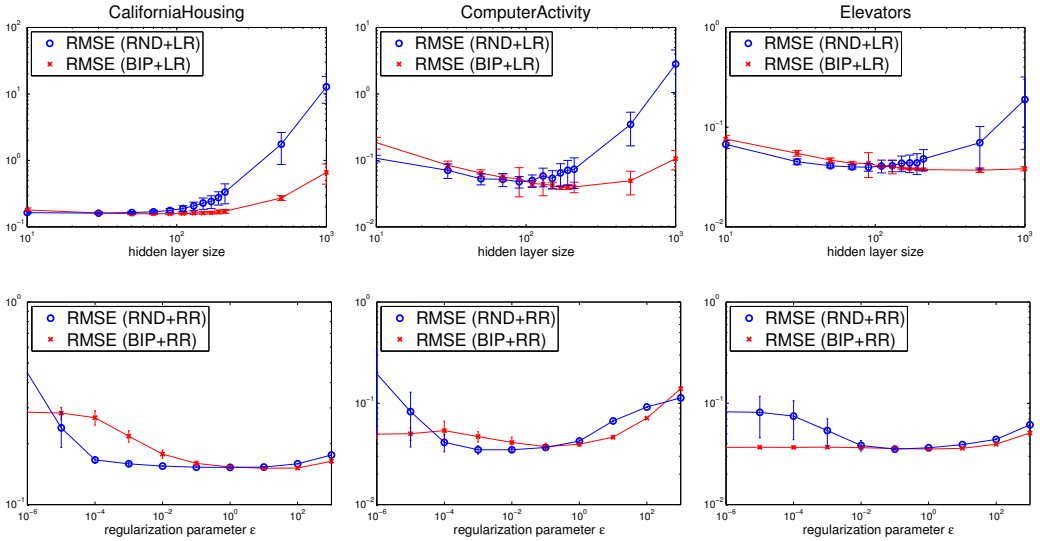

Fig. 3.6: RMSE development on the test set for growing hidden-layer size and different ridge regression parameters.

BIP-pretrained networks can handle larger hidden-layer without overfitting the data. The change in the hidden-layer encoding shows a significant impact on the networks' performance. Since large networks have a good generalization ability after training with BIP, ELMs with a $R = 500$ neuron hidden-layer are used for further experiments. The expectation is that the BIP networks, in this case, perform better than the random networks after supervised readout training with ridge regression, since the previous experiments show that BIP improves the generalization ability significantly for large networks. The test error results are stated for the best performing networks with the corresponding output regularization strength α which was found in $\alpha \in [10^{-9}, 10^{-8}, \ldots, 10^{3}]$ in Tab. 3.4.

The performance results from other methods are summarized in Tab. 3.5, in order to compare the results obtained for the BIP-pretrained ELMs with the state of the art optimization techniques for ELMs. The performance results for the RELM, EI-ELM, I-ELM and CI-ELM where taken from [33, 23, 37, 112] respectively. It is shown that the BIP-pretrained ELMs additionally trained with ridge regression can perform better than other state of the art models. However, the optimization is done in a complementary way, which is important to note. In such a case, it is possible to combine those methods with BIP. Interestingly, the RELM performs best on almost all of the tasks, which is not fully consistent with the results obtained in this thesis. The results demonstrate that the best performance of the BIP networks additionally trained with ridge regression in comparison to the randomly initialized networks is not only due to ridge regression or BIP. The combination of ridge regression and BIP leads to significantly better performing networks with

Task	**BIP**	R	**RND**	R
Ab	**.075±.001**	35	.075±.001	35
Ca	**.158±.005**	90	.159±.003	25
Ce	**.066±.001**	210	.068±.002	110
De	**.053±.001**	130	.053±.001	70
Co	**.039±.002**	170	.048±.009	90
El	**.037±.002**	500	.040±.004	90

Tab. 3.3: The results of BIP pretrained in comparison to randomly initialized ELMs for regression tasks. The networks are stated with the optimal hidden-layer size.

BIP	$\lg \alpha$	**RND**	$\lg \alpha$
.073±.001	1	.074±.001	-1
.149±.001	2	.153±.001	0
.063±.001	0	.064±.001	-3
.053±.001	1	.053±.001	-2
.034±.001	0	.035±.002	-2
.036±.001	0	.036±.001	-1

Tab. 3.4: BIP pretrained and randomly initialized ELMs. The networks are stated with the optimal ridge regression parameter.

Task	**RELM**	**EI-ELM**	**I-ELM**	**CI-ELM**
Ab	0.076 ± 0.006	0.082 ± 0.002	0.092 ± 0.005	0.083 ± 0.003
Ca	0.117 ± 0.003	0.149 ± 0.002	0.168 ± 0.005	0.155 ± 0.005
Ce	0.036 ± 0.002	0.083 ± 0.001	0.092 ± 0.867	0.087 ± 0.002
De	0.018 ± 0.003	0.058 ± 0.003	0.074 ± 0.013	0.060 ± 0.007
Co	0.019 ± 0.003	0.094 ± 0.003	0.120 ± 0.013	-

Tab. 3.5: Mean performance and standard deviation of other state of the art methods on the respective regression tasks used to benchmark BIP pretrained ELM with ridge regression.

a low variance in the test error for all tasks used in the experiments. In addition, the BIP trained networks have higher optimal ridge regression parameters. This is due to the fact that the features provided by the hidden-layer are more task suited. Less large output weights are needed to produce a good mapping. This is desired cause it makes the networks robust against noise in the hidden-layer state.

3.6 Conclusive Remarks

This chapter introduces batch intrinsic plasticity (BIP) and suggests its application as optimization method for the neural encoding of ELMs by shaping the output distributions of the hidden-layer neurons. This method represents a key ingredient for reliable learning because it makes the ELM technique robust to random initializations.

The method is used to initialize the network's input weights and biases without detailed knowledge about the task and also overcomes the problem of slow learning dynamics of the original IP algorithm by formulating a regression problem. The experiments show that the new learning method produces the desired output dis-

tributions despite that only two parameters per hidden-layer neuron are used for adaptation. It is demonstrated that the production of sparse codes in the hidden-layer of a neural network by producing exponential distributions with low fixed mean can lead to good generalization results in two different robotic scenarios. Especially networks with a large hidden-layer have a better generalization ability when pretrained with BIP.

In addition, the method is compared to other ELM techniques for different real world regression tasks from the UCI machine learning repository. Since every hidden-layer neuron is updated separately, BIP is well suited for methods optimizing ELMs by incrementally growing the network without re-computation of the already obtained biases and slopes. BIP complements those methods and could - combined with other optimization methods - lead to even better learning results for ELMs. In addition, every neuron can be trained by a different desired output distribution in order to produce a more diverse encoding in the hidden-layer. Only the desired distribution f_{des} and the inverse of the activation f^{-1} is needed for the method, which points out the high flexibility of the method. The generic formulation might be used to analyze the performance of the method with respect to other desired output distributions and activation functions. This leads to different codes in the hidden-layer and has a huge impact on the network's performance.

One important information to mention is that BIP is not well suited to learn from correlated data (e.g. from trajectories that only cover small parts of the workspace). Since BIP changes the activation functions of the neurons such that their output distributions become exponentially distributed, input data that is not close to the training data very likely leads to saturated neurons. This is not favorable for generalization. A possible solution to this issue is to draw input samples randomly from a desired distribution in the workspace that differs from the training data. Unfortunately, these different cases for unsupervised learning with BIP is yet not well investigated and is part of future research.

Chapter 4

Robustness to Drifts by Intrinsic Plasticity with Natural Gradient

Drift compensation is an important approach enhancing the reliability of machine learning techniques in long-term scenarios where measurements are made for long periods of time. The deterioration of an involved sensor or systematical shifting and scaling of the sensory information through other processes can lead to an inherit change of the underlying function which is referred to as input drift. The compensation of such drifts is tackled in this chapter by means of unsupervised adaptation through intrinsic plasticity which is optimized with the natural gradient technique and another weight scaling mechanism.

The remainder of this chapter is organized as follows. The first section of this chapter, Sect. 4.1, emphasizes the role of unsupervised learning by intrinsic plasticity via natural gradient descent for reliable learning results. In particular, the reliability to temporal changes of the mapping are discussed in detail. Sect. 4.2 revisits intrinsic plasticity (IP) as a stochastic gradient descent method whereas the following sections (Sect. 4.3 and Sect. 4.4) introduce an extension of IP through the use of the natural gradient technique. Sect. 4.5.2 provides a rigorous analysis on this new online adaptation rule in the context of drift compensation. Sect. 4.5.3 demonstrates the method in a real world scenario with the humanoid robot iCub and Sect. 4.6 provides a rigorous discussion of the chapter.

The natural gradient technique for IP was introduced in [15]. The results obtained by use of the natural gradient descent approach with application to drift compensation are taken from [16]. Part of these results are taken from Strub's master thesis [113].

4.1 Reliability and Drift Compensation

Drift compensation is highly useful in real world applications where measurements are made for long periods of time [114] or if inputs are influenced systematically through other processes like, for instance, in case of a change of illumination or a (sudden) displacement of a camera. The literature shows that a detection of

the drift before the compensation is a promising approach [115, 116]. In these cases a suitable compensation strategy needs to be chosen in order to cope with the drift successfully. One such strategy is to adjust the data accordingly, e.g. to recenter to compensate respective mean shifts or to rescale to compensate changes in variance. This requires external data analysis and appropriate measures but does not adapt the learned model to internalize the drift. Thereby the learned model does not actually encode for the current real world input, but rather for the input at learning time before the drift occurs.

This is opposed to an implicit approach to drift compensation that internalizes the drift into the model by continuous re-adaptation. In principle, continuous online learning of the weights though backpropagation can provide respective re-learning. This actually is a strong argument in favor of applying online-learning while already exploiting the learned model. But it requires that error feedback is continuously available to change the neural code that solved the learning task for the original data. In real applications, this may not be feasible. Consider for instance the application of a learned virtual sensor for which training data can be generated in the laboratory using a costly direct sensor. The goal is to replace this sensing in the real product, where supervised re-adaptation of the weights is consequently not possible by definition.

Drift compensation through IP learning in the presence of mean and variance changes offers a different and novel approach, which internalizes the drift in the network model so that the input data does not need to be analyzed. It simultaneously sustains the neural encoding, which was learned using the error feedback for the original data, so that there is also no need for continuous error feedback and retraining. IP achieves this by exploiting that the considered networks restrict error driven weight adaptation to the outputs of the network, whereas optimization of the input encoding in the hidden-layer is decoupled from the output weight adaptation and provided by the IP learning. The drift compensation is therefore in some sense a desired side effect of the local and unsupervised optimization of the encoding of each single neuron in the network. Note that we do not consider so-called concept-drifts, which means online changes in the desired output function [117], as opposed to compensation of online changes of the input signal.

However, it turns out that only the combination of a proposed modification of the IP rule and the usage of the natural gradient provides an IP learning dynamics that it is well suited for compensating such drifts. Therefore, the natural gradient for intrinsic plasticity learning is introduced and its impact is analyzed in rigorous detail. Also, a further novel mechanism to adaptively transfer persistent changes of the activation function caused by IP to the weights is presented in order to avoid numerical instabilities.

It is shown that the use of the natural gradient descent technique is indispensable for successful drift compensation because the learning dynamics potentially suffer all known drawbacks of standard stochastic gradient. The parameter estimates can lead to small gradient norms in some regions of the parameter space, so

called plateaus, where convergence is slow. One reason for this is that the parameterization and the corresponding output of a model are defined in different metric spaces. Most gradients defined on an error measure only utilize the Euclidean metric in the parameter space. But, generally, there is no reason to assume that the Euclidean metric is the preferential distance measure between solutions. It is well known that the parameter space has a Riemannian metric structure in many cases, for instance in the weight space of neural networks [118]. The parameter space can be analyzed by means of information geometry - a theory which employs differential-geometric methods in statistics [119, 120].

While the steepest direction in a parameter space with an Euclidean metric structure is given by the conventional gradient, the steepest direction in a parameter space with Riemannian metric structure is given by the so-called natural gradient. It is obtained by transforming the Euclidean metric in the output space by means of an often only locally defined metric tensor into the parameter space. The tensor needs to be well-suited to the Riemannian metric of the parameter space.

It has been shown that the natural gradient can be advantageous for different stochastic learning setups like blind-source separation or statistical estimation of probability density functions (see e.g. [118, 121, 122]). It has also been applied to improve the learning dynamics of multilayer networks [118, 123]. For instance in [124], the authors distinguish between a transient and an asymptotic phase in the learning dynamics which both show significant gains in performance over standard gradient descents. The concept of natural gradient has further been extended to more general classes of multidimensional regression and classification problems in [65]. An alternative derivation of the natural gradient is given in [125], together with the natural equivalent of batch learning, linked to Levenberg-Marquardt optimization. Recently, the special case of learning for non-linear discriminant networks was improved by use of natural gradient in [126].

As opposed to standard neural learning, where the input weights are adapted, IP learning adapts parameters of the activation function. In this chapter, the corresponding Riemannian metric tensor for IP, which was first introduced in [12], is defined and analyzed in detail. Thereby, the natural gradient for IP is introduced. Like in other domains, where natural gradients were previously explored, experiments reveal that the Riemannian metric and the associated natural gradient are more suited to describe distance relations between output distributions for IP and provide superior learning dynamics.

The following sections describe the natural gradient descent for IP and proposes techniques to tackle numerical issues of the IP learning rule. Experiments that analyze the differential-geometric properties of IP are provided in order to complement the theory of natural gradient. It is also shown how the learning dynamics change when following the natural gradient. The major part is to demonstrate the effects of natural gradient IP learning for compensating drifts in the input, including a real world learning task from robotics. In summary, this chapter shows that IP

learning simultaneously provides unsupervised learning and input drift compensation through sustainment of the neural encoding in the hidden-layer which is an important property of reliable learners in long-term learning scenarios.

4.2 Intrinsic Plasticity as Stochastic Gradient Descent

Intrinsic Plasticity (IP) was developed by Triesch in 2004 [45] and optimizes the information transmission of a single neuron strictly locally by adaptation of slope a and bias b of the activation function such that the neuron's output y becomes approximately exponentially distributed with a fixed mean μ. This is done with respect to the input sample distribution $f_x(x)$. IP-learning can be derived by using the insight from statistics that $f_y(y) = f_x(x) \cdot (\partial y / \partial x)^{-1}$ and the equation $\partial y / \partial x = ay(1-y)$, obtained by analyzing the Fermi function. Minimization of the difference $L(f_y, f_{\exp}) = L(\theta)$ between the output f_y and an exponential distribution $f_{\exp}$, quantized by the Kullback-Leibler-divergence [127] (KLD) delivers the following:

$$L(\theta) = \mathbb{E}[l(y,\theta)] = C + \int_\Omega \underbrace{-\ln\left(\frac{ay(1-y)}{e^{\frac{1}{\mu}y}}\right)}_{l(y,\theta)} f_y(y)\ \mathrm{d}y\ , \tag{4.1}$$

where C remains as a constant of the potential and can be neglected without loss of information. Since IP was introduced as a stochastic gradient rule, the respective online loss function can be identified with the integrand $l(y,\theta)$ (see Eq. (4.1)). Here the KLD is interpreted as expected loss $\mathbb{E}[l(y,\theta)]$ for the input samples x distributed according to $f_x(x)$. A separation of Eq. (4.1) into the entropy $H_x[y]$ and the expectation value of the output distribution $\mathbb{E}_x[y]$ is possible [107], which directly shows that a minimization of $L(\theta)$ for a fixed mean $\mathbb{E}_x[y]$ is equivalent to entropy maximization of the output distribution. Additional information about the KLD and its relation to exponential distributions can be found in [128].

Fig. 4.1 shows how four different input distributions (first row in the figure) are transformed into exponential-like distributions (second row in the figure) after training with IP. The figure clearly reveals that the best possible fit after IP learning is highly dependent on the input distribution. This is due to the fact that only two parameters in the Fermi function are adapted. These distributions are used as input for the experiments in the following sections.

4.3 The Natural Gradient for Intrinsic Plasticity

Given an input distribution $f_x(x)$, an analog neuron establishes a differentiable mapping between the parameter space $\Theta = \mathbb{R}^2$ and the manifold of possible output distributions Υ. The KLD comparing a given distribution to the exponential distribution with fixed mean μ in Eq. (4.1) can be used to derive a canonical distance measure on the output distribution space resulting in a Riemannian metric

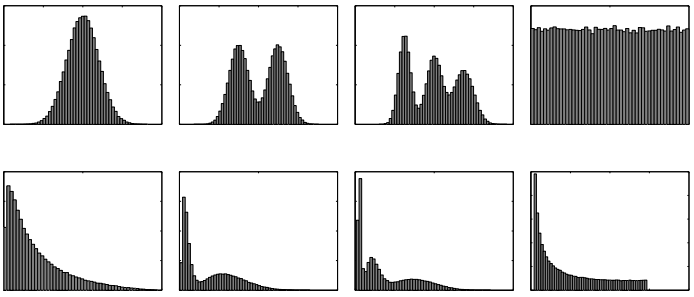

Fig. 4.1: Four input distributions $f_x(x)$ (1st row) and the learned exponential-like output distributions $f_y(y)$ for $\mu = 0.2$ (2nd row).

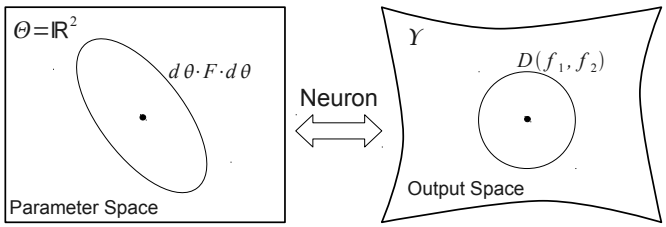

Fig. 4.2: Differentiable relation (neuron) and metrics F and D between parameter space $\Theta = \mathbb{R}^2$ and manifold of possible output distributions Υ.

F on the parameter space Θ. The metric determining the distance between two output distributions $y_1(x) = y(x, \theta_1)$ and $y_2(x) = y(x, \theta_2)$ in Υ defined by the parameter settings θ_1 and $\theta_2 = \theta_1 + \mathrm{d}\theta$ in Θ for an infinitesimal change of parameters $\mathrm{d}\theta$ is given by $D(y_1, y_2)$. This distance measure is transformed such that it induces the Riemannian metric tensor $F(\theta)$ - a 2×2 positive definite matrix given by the Fisher information [129] - as a pull-back onto the parameter space:

$$\begin{aligned} D(y_1, y_2) &= \mathbb{E}_x[(l(y_1, \theta_1) - l(y_2, \theta_2))^2] \\ &= \mathbb{E}_x[(l(y_1, \theta_1) - l(y_1, \theta_1) - \nabla l(y_1, \theta_1)\mathrm{d}\theta)^2] \\ &= \mathbb{E}_x[(\nabla l(y_1, \theta_1)\mathrm{d}\theta)^2] \\ &= \mathrm{d}\theta \cdot \mathbb{E}_x[\nabla l(y_1, \theta_1) \cdot (\nabla l(y_1, \theta_1))^T] \cdot \mathrm{d}\theta = \mathrm{d}\theta \cdot F(\theta) \cdot \mathrm{d}\theta. \end{aligned} \tag{4.2}$$

This idea guarantees that the distance between two parameter vectors θ_1 and θ_2 - as measured by the length of the geodesic with respect to the metric tensor $F(\theta)$ - is equal to the previously defined distance measure $D(y_1, y_2)$ on the corresponding output distributions y_1 and y_2 in Υ. The relation between parameters and output distributions established by a non-linear transfer function of a neuron and its' corresponding distance measures is schematically illustrated in Fig. 4.2.

The steepest descent direction of a potential with Riemannian structure is given by the natural gradient defined by the metric tensor. The following update

equation is obtained when using the natural gradient for IP:

$$\theta_{t+1} = \theta_t - \eta_{\text{IP}}(F(\theta) + \varepsilon I)^{-1}\nabla l(y,\theta) = \theta_t - \eta_{\text{IP}}\nabla_{\text{NIP}} l(y,\theta) \ , \tag{4.3}$$

where I is the 2×2 - identity matrix and $\varepsilon \geq 0$ is a positive scalar. We call $\nabla_{\text{NIP}} := (F(\theta) + \varepsilon I)^{-1}\nabla$ the natural gradient operator for IP. Typically ε can be set to zero to obtain a plain natural gradient formulation. But in the more general definition Eq. (4.3), ε introduces a blending between standard and natural gradient. Note that this blending influences the step width of the numerically applied gradient descent and stabilizes the inversion of the metric tensor F.

The main problem with this formulation is that the expected gradient with respect to the input is needed, but not available in an online framework. However, it was shown in [123] that is possible to estimate the metric tensor online by defining a proportional control law:

$$\dot{\hat{F}}(\theta) = \lambda \left(F(x,\theta) - \hat{F}(\theta) \right) \ , \tag{4.4}$$

where $\hat{F}$ is the estimate of the Fisher matrix and $F(x,\theta)$ the Fisher information for one input element x. The problem then reduces to finding a good λ, which must be small since the loss function is continuous and a good initial value $\hat{F}_0(\theta)$. The learning by natural gradient descent thus becomes online capable and computationally feasible.

4.4 Working-Point Transformation for IP

A closer inspection of Eq. (3.2) (left) reveals that the standard IP rule can suffer from numerical instabilities in particular for large input amplitudes which lead to small slopes a. In this regime of small slopes the discretization becomes problematic due to the singularity induced by the $\frac{1}{a}$-term, illustrated in Fig. 4.5 (A). The combination with the results of experiments in Sect. 4.5.2 reveals that IP has a "working point" at $a = 1$. It is therefore favorable to keep the parameter a close to this "working point", which can be achieved by a novel modification of IP learning. It proposes to scale the neuron's input weights with the slope in order to transform the working point appropriately:

$$\Delta\vec{w} = \eta_{\text{WS}} \cdot (-\vec{w} + a \cdot \vec{w}) \ . \tag{4.5}$$

with $\eta_{\text{WS}} < \eta_{\text{IP}}$ (η_{WS} is set to 10^{-5} in the experiments). With this additional adaptation rule, the slope tends to converge back to unity, as the weights converge to the former slope values. Hence the semantics of the IP learning rule remain the same while transferring the learned pertinent slope information from the slope parameter a to the synaptic weights. The collection of learning rules given by Eq. (4.3), Eq. (4.2) and Eq. (4.5) are used in the following experiments and called Natural IP (NIP).

4.5 Experimental Results

This section contains the experiments investigating the impact of the natural gradient and the working point transformation on the online IP rules with an additional application to drift compensation. Sect. 4.5.1 investigates the differential geometric properties of the natural gradient descent for intrinsic plasticity in two different scenarios. The novel learning rule for drift compensation is analyzed in Sect. 4.5.2. Sect. 4.5.3 demonstrates that drift compensation by NIP is possible in a real world scenario involving the humanoid robot iCub.

4.5.1 The Impact of the Natural Gradient on Intrinsic Plasticity

The following experimental results are obtained for a single-neuron model with parameterized Fermi function. The experiments are performed with different inputs: the first row in Fig. 4.1 shows the four different input distributions that are used for investigation. A Gaussian (1-G), a bipartite (2-G), a tripartite (3-G) Gaussian and an uniform (U) distribution. $N_{\text{tr}} = 100$ samples are independently drawn from each distribution and used for training. A step width of $\eta_{\text{IP}} = 10^{-3}$ and a numerical stabilization of $\varepsilon = 10^{-1}$ is used. For online estimation of the metric tensor a decay rate of $\lambda = 0.01$ is used.

Information Geometry

The following experiment visualizes how the geometry of the potential L changes

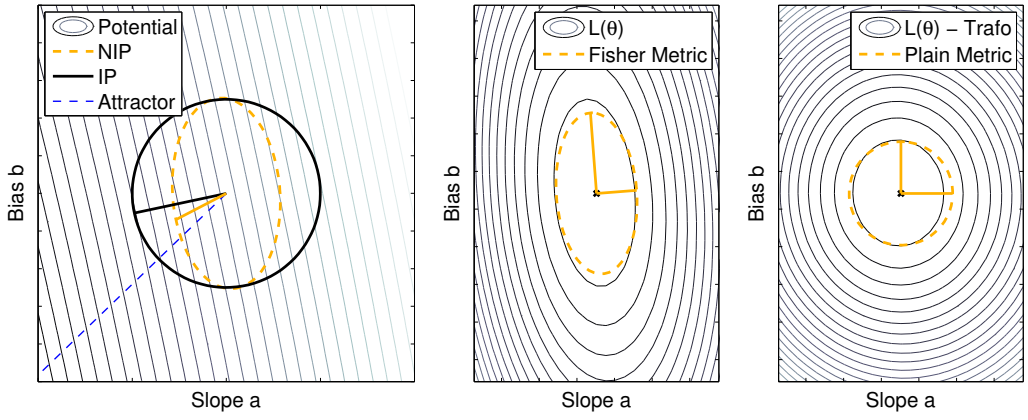

Fig. 4.3: Fisher metric at point θ for the 1-G distribution (left). The geometry change of the attractor basin using the natural gradient (center, right). IP potential and Fisher metric at the attractor (center). NIP potential and plain Euclidean metric (right).

by use of the metric tensor F at the attractor θ^*. The 1-G distribution is used as input to a single Fermi neuron model for illustration.

Fig. 4.3 (left) shows schematically the Euclidean and the Fisher metric at a given parameter configuration $\theta = (2, -0.5)^T$. Note that the gradient induced by the Fisher metric is not orthogonal to the equipotential lines anymore. The direction of the steepest descent therefore changes and points in a more direct way to the attractor than the standard gradient for IP which is using the plain Euclidean metric visualized as black circle in the figure. Fig. 4.3 (center) shows the potential $L(\theta)$ with a clearly visible plateau in b-direction, where the change in the KLD is small. The dashed line is the unit circle with a radius of η in the geometry defined by the metric tensor $F(\theta^*)$, which is well suited to the potential: the unit circle is stretched in b-direction. Fig. 4.3 (right) visualizes the distortion of the potential after transformation with $F(\theta^*)$. The induced landscape becomes "Euclidean-like" after transformation and loses the plateau - the transformed potential is isotropic.

Information Geodesy

The following experiment focuses on a more global analysis of the natural gradient descent. A gradient descent from a given starting point θ to the attractor θ^* is performed while the relative geodesic length (RGL) of the path is recorded. The RGL gives the length of the geodesic γ from starting point θ to the attractor θ^* with respect to the shortest way in the parameter space:

$$\mathrm{RGL}(\theta) = \frac{\int_\gamma \mathrm{d}s}{\|\theta - \theta^*\|} \ , \tag{4.6}$$

where $\mathrm{d}s = \sqrt{\mathrm{d}a^2 + \mathrm{d}b^2}$ is the infinitesimal arc length in parameter space.

Fig. 4.4 (left) shows the potential field $L(\theta)$ of the Gaussian input distribution (1-G) while the right hand side of the figure shows the potential field $L(\theta)$ of the uniform input distribution (U). It also shows four starting points θ_{1-4} for the learning of each input distribution. The black lines show gradient descents performed by IP, while the yellow lines are the geodesics from the NIP learning. Both approaches have the same fixed-point, but the geodesics of the NIP learning imply a more direct path to the attractor in parameter space. Thus the natural gradient method is better suited to the potential than the conventional IP gradient. Tab. 4.1 displays the results of an experiment where the RGL is measured for $N = 100$ different starting points

Task	$\mathbb{E}$[RGL] (**IP**)	$\mathbb{E}$[RGL] (**NIP**)
1-G	1.3493 ± 0.5730	**1.0748 ± 0.0526**
2-G	1.0473 ± 0.0300	**1.0234 ± 0.0342**
3-G	1.1209 ± 0.0753	**1.0505 ± 0.0506**
U	1.0219 ± 0.0206	**1.0056 ± 0.0099**

Tab. 4.1: Relative average length of the geodesics $\mathbb{E}$[RGL] and their standard derivation $\sqrt{\mathbb{E}[(\mathrm{RGL} - \mathbb{E}[\mathrm{RGL}])^2]}$ for IP and NIP learning.

drawn from a Gaussian distribution centered around the attractor with covariance matrix $\Sigma = I$. It shows the average RGL and its standard deviation.

Since the minimum value for the RGL is one (which corresponds to a straight line from the initial point to the attractor in parameter space), the values for the RGL in Tab. 4.1 show that the geodesic lines of NIP are almost straight for all tested input distributions (visualized in Fig. 4.4). In addition, the low standard

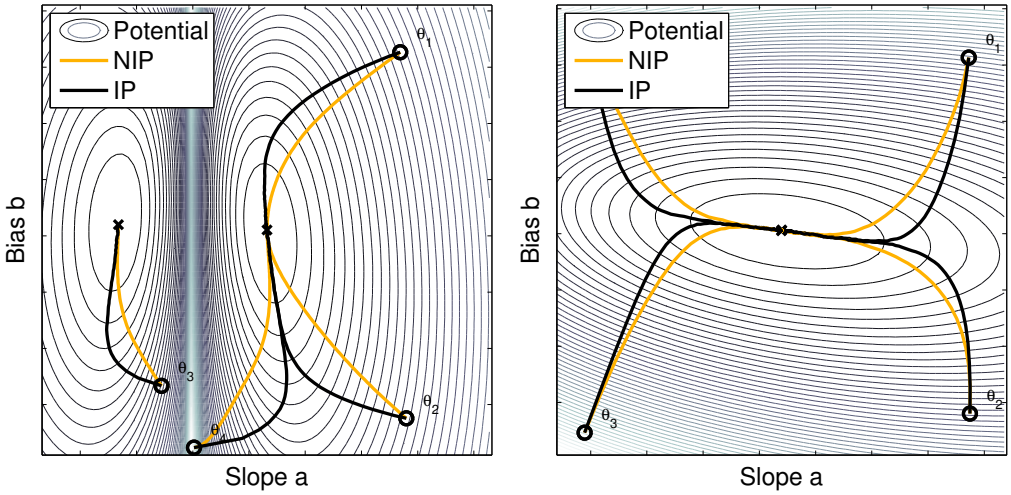

Fig. 4.4: Geodesics in the IP potential. Geodesic lines for the Gaussian (1-G) distribution (left). Geodesic lines for the uniform (U) distribution (right).

deviation demonstrates that the curvature of the geodesic is more independent from the initial point in the potential when using the natural gradient instead of the conventional gradient.

4.5.2 Drift Compensation with Intrinsic Plasticity

Drift compensation is a practically highly relevant issue for the application of machine learning algorithms, because in real plants sensors and actuators are typically subject to wear and other non-stationary effects, e.g. induced by temperature changes. As already discussed in the introductory part of this chapter, drift can be externally compensated by re-adjusting the data which requires an additional mechanism, or internalized into the learned model for continuous adaptation. IP provides a novel approach to the latter, because it internalizes the drift in an unsupervised and local way, while not relying on continuous error feedback learning. It optimizes the neural encoding by shifting the mean through the bias and scaling the variance through the slope parameter in the activation function. Drift compensation can thus be considered as an inherent side effect of the IP learning approach, which has not been analyzed in the IP literature yet. Obviously,

the learning dynamics of IP changes the effectiveness of drift compensation and the following sections show that the interplay of input drift and the standard IP learning dynamics leads to typical plateaus, which can be avoided when using NIP.

Analysis of IP Learning in the Presence of Input Drifts

The following experiment with a synthetic input signal provides an initial analysis of IP learning with and without the natural gradient in the presence of drifts. It demonstrates that standard IP always implies some drift compensation, but it is not fast enough and can be improved trough the proposed modifications. Experiments are performed using an input signal comprising a product of oscillations $x(t) = \sin(0.2t) \cdot \sin(0.053t) \cdot \sin(0.092t)$. In the beginning, IP learning is applied for 5×10^4 steps in order to let the parameters converge with a learning rate of $\eta_{\text{IP}} = 10^{-3}$. After learning, two manipulations of the input signal are carried out to analyze the impact of the natural gradient and weight scaling on the IP learning dynamics in the presence of drifts: (i) gradual scaling of the signal changing the variance and (ii) a gradual shift of the signal changing the mean. In the first experiment, a gradual linear scaling of the input signal up to a factor of 100 or $^1/_{100}$ respectively starting from 1 in 10^6 steps is applied. In the second experiment, a gradual linear shifting to 50 or -50 respectively starting from 0 in 5×10^5 steps is applied. Whereas these scaling and shifts are taken to the extremes, they are meant to show the full behavior of the learning algorithm and to allow to clearly display and discuss the effects of the NIP and weight scaling modifications to the original IP rule.

The plots in Fig. 4.5 show the IP (black) and NIP (yellow) learning dynamics for the described scalings and shifts. The blue-dashed lines show the computed ground truth target slope and bias, which are necessary to perfectly compensate the input signal manipulation. The left column summarizes the results for the scaling while the right column outlines the effects for the shifting of the input signal. The first row in Fig. 4.5 (A, B) displays the logarithm of the slope ratio for IP (black) and NIP (yellow), which is defined as follows:

$$\text{“Slope Ratio” (IP)} : \frac{a(t)}{a_{\text{train}}} \qquad \text{“Slope Ratio” (NIP)} : \frac{a(t) \cdot s(t)}{a_{\text{train}} \cdot s_{\text{train}}}$$

where $a(t)$ is the recorded slope at time t and a_{train} is the learned slope for the non-scaled and non-shifted input. The term $s(t)$ simply denotes the scaling factor of the weights at time step t with respect to the weight with unity norm, in order to make the results comparable. The second row (C, D) shows the shifting of the bias $b(t)$ with respect to the bias b_{train} for the non-manipulated input signal divided by the actual slope $a(t)$. Hence, the plots show the effective mean shift by IP (black) and the adapted NIP (yellow) learning rule.

$$\text{“Bias Shift” (IP):} \frac{b(t) - b_{\text{train}}}{a(t)} \qquad \text{“Bias Shift” (NIP)} : \frac{b(t) - b_{\text{train}}}{a(t) \cdot s(t)}$$

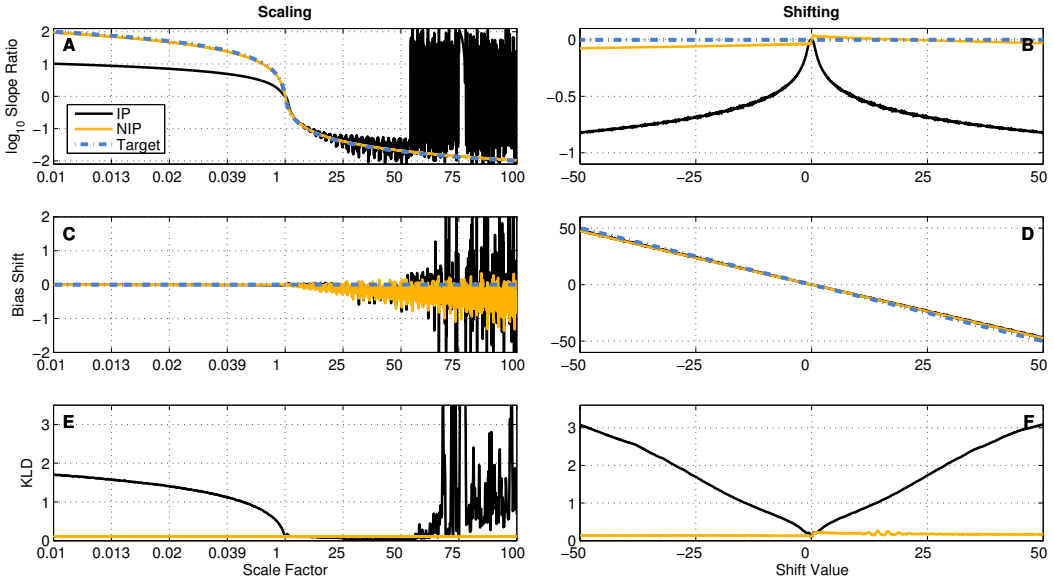

Fig. 4.5: IP and NIP in a single neuron. The logarithm of the slope ratio for each scaling and shifting value, respectively (A, B). The bias shift divided by the slope for the according scaling and shifting value (C, D). The KL-divergence for scaling and shifting (E, F). These experimental results are from [16, 113].

The KLD is computed to measure the success of the respective adaptation by IP or NIP (see E and F).

Variance Shift Reveals Working Point. Fig. 4.5 (A) shows that the IP learning rule is not able to counteract the signal manipulation for an increasingly small scaling within the given time constraints. While compensation is good for small variations, IP learning degenerates when the scaling decreases further. In the case of a linearly increasing scaling, the slopes get very small until numerical instabilities occur due to discretization. The plot also shows that the adapted learning rule from Eq. (4.5) suffices to achieve the target slope ratio and resolve the numerical instabilities.

Basically, the shift of the working point by the Δw learning rule is responsible for the good matching of the target - see the NIP line (yellow) in Fig. 4.5 (A). Fig. 4.5 (C) shows that IP as well as NIP react suitably and hardly adapt the bias when scaling the input signal. The oscillations of the "Bias Shift" in the plot are mainly induced by the division with the slope - small variations in the bias get magnified for very small slopes. This oscillation is an effect of discretization, although not really a problem because the actual bias changes itself are very small. The KLD for the IP and NIP cases are shown in Fig. 4.5 (E) and confirm the increased performance - the KLD is small for the new learning rule.

Mean Shift Leads to Plateaus. Fig. 4.5 (B) illustrates the occurrence of a systematic overestimation of the input variance by IP when the signal is shifted. The reason for the overestimation is that decreasing the slope leads to an increased effect of the bias shift. In fact, IP drives the neuron into a parameter regime where the gradient nearly vanishes which leads to very slow convergence, e.g. a typical plateau that prevents efficient learning. This suboptimal behavior is rectified by using NIP which gives a much better estimation of the gradient direction in parameter space. Therefore the slope ratio with NIP hardly changes during a pure shift of the input signal, although a small underestimation can still be seen. Fig. 4.5 (D) shows that IP as well as NIP achieve a good compensation for the shift by use of the bias. However, the KLD for NIP stays close to the optimal value for shifting of the input signal - in contrast to the results for IP adaptation (see Fig. 4.5(F)).

4.5.3 Learning to Point with the Humanoid Robot iCub

In this section, a real world task involving the humanoid robot iCub demonstrates that drift compensation is possible by sustaining the neural encoding with the proposed learning scheme. Humanoid robots are designed to solve service tasks in environments where a high flexibility is required. To cope with various kinds of drifts in the input, e.g. with changing illumination or a displacement of a sensor, is a prerequisite for such systems. We use a hand-eye coordination task that is inspired by [130] and is discussed in depth in a further contribution [131]. The humanoid robot iCub learns to point towards an object based on the raw visual input from his head cameras. We investigate, if NIP learning can cope with the input shift that is associated with a small displacement of the head, a typical problem if there is wear in the mechanical mechanism. As before, we explore quite extreme and even exaggerated displacements to challenge the NIP algorithm.

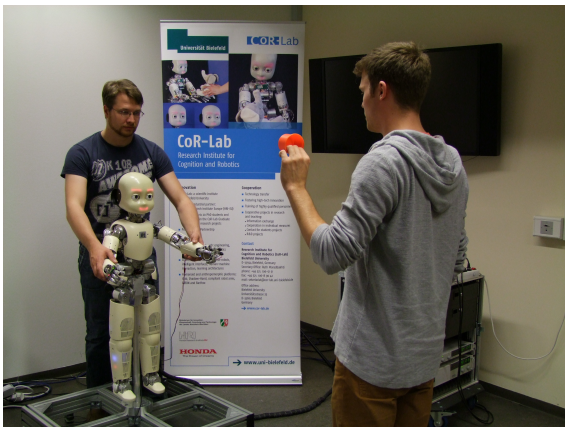

Fig. 4.6: Kinesthetic teaching of the humanoid robot iCub. One tutor guides iCub's left arm and another holds a red cup to point at in his left hand.

The experimental setting is illustrated in Fig. 4.6. A human tutor physically guides iCub's arm by means of kinesthetic teaching using a recently established force control on the robot. The tutor can actively move all joints of the arm to place the end-effector at the desired position. To create training data, a second person in an approximate distance of two meters to the robot moves an object in the visual field of iCub while the human tutor is guiding its arm to point at the object by means of a laser pointer attached to the robot's hand. The 2D-pixel coordinates of the object in both eye-cameras are extracted by a tracking system and recorded together with the joint angles of the arm, for further details see [131].

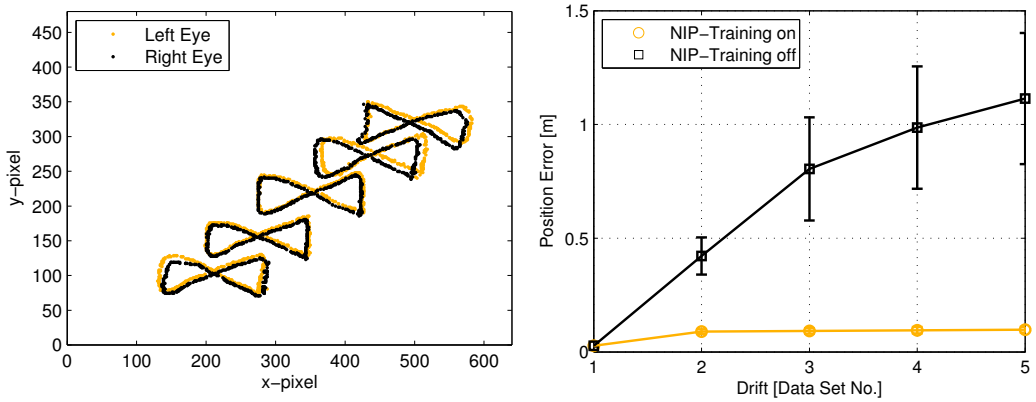

Fig. 4.7: Drift compensation by sustaining the neural encoding with NIP for the humanoid robot iCub.

NIP learning is applied online in order to adapt the hidden-layer of ELMs. The networks consist of $R = 100$ neurons, the network's weights and biases are initialized randomly from an uniform distribution in the interval $[-1, 1]$, and the slopes are initially set to one. The regression parameter is $\alpha = 10^{-3}$ in the following experiments. The task is to learn the mapping from the 2×2D pixel coordinates received by both cameras (with a resolution of 640×480 pixels) onto the end effector configuration of the robot's left arm, i.e. the respective joint angle configuration. $N = 5$ data sets are recorded where the tutor in front of iCub painted eight-like figures. The data sets comprise 458 to 506 samples where $N_{\text{tr}} = 300$ where used for training and the remaining samples were used for testing. For each data set, a different head configuration of pan and tilt-angle of the neck was chosen from $\theta_{\text{pan}} \in [-20, 20]$ and $\theta_{\text{tilt}} \in [-20, 20]$ divided equally in 5 steps. This represents a relatively strong displacement of the head, which leads to a shift in the input data as visualized in Fig. 4.7 (left). The pointing task remains invariant. Note that in this case it is infeasible to cope with the input shift by continuous online supervised error learning, because no error feedback is available when exploiting the learned model to realize the actual pointing on the robot.

Fig. 4.7 (right) compares the ELM network's performance with and without NIP on the shifted data without re-adapting the output weights. The network is pre-trained on the first data set by means of NIP for 1000 epochs and then trained by ridge regression. Then the network is tested on the shifted data sets 2-5, either directly or after additional 1000 NIP epochs on each new data set to compensate the shift. All results are averaged over 10 network initializations. The test error significantly increases for data sets 2 to 5, if no NIP training is applied. This is expected, since the data varies with changing head configuration and the network can not arbitrarily generalize. However, the NIP training can compensate the shift and keep the error low, despite relatively large displacements and without re-training the output weights.

4.6 Conclusive Remarks

This chapter presents two interconnected contributions to IP learning which was previously introduced as a biologically plausible and effective means to optimize the encoding of inputs in neural networks of various types.

First, the well known natural gradient is introduced and analyzed for intrinsic plasticity. The significant impact of the natural gradient for this learning dynamics is shown and an additional modification of IP learning is introduced, which targets to further optimize IP by keeping the parameters close to a suitable working point. These modifications improve IP learning over the previous learning scheme and can be applied to any of the known applications of IP learning.

Second, the implicit capability of IP learning to cope with drifts in the input is analyzed for the first time and identified as a very special mechanism and novel approach to drift compensation. It internalizes the effects of drift into the learned model by adaptation of the activation function parameters without the need to change the input data. This adaptation is achieved without the usage of online error feedback. This is a novel and highly useful approach and can be applied when using ELM feed-forward type networks. However, it turns out that the drift compensation effect works well only in connection with the proposed modifications and improvements of the IP learning dynamics.

Further work shall be directed towards a closer comparison of possible drift compensation schemes and towards the identification of problem domains where this scheme works well. In particular the interplay of IP learning speed and drift speed deserves attention in order to reliably compensate input changes. Nevertheless, the novel learning scheme provides a first account on drift compensation with IP learning and yields encouraging results on synthetic and first real world data obtained through improved natural gradient learning.

Chapter 5

Reliability via Incorporation of Continuous Constraints

The black-box character of neural networks in general and extreme learning machines (ELM) in particular is one of the major concerns that repels engineers from application in unsafe automation tasks. This particular issue is approached by incorporation of continuous constraints into the learning process of ELMs which are derived from prior knowledge about the specific task. This is reasonable, because machine learning solutions have to guarantee a safe operation in many application domains. This chapter reveals that the special form of ELMs, with its functional separation and the linear read-out weights, is particularly well suited for the efficient incorporation of continuous constraints in predefined regions of the input space.

The remainder of this chapter is organized as follows. The first section of this chapter, Sect. 5.1, explains the relation between reliability of ELMs and the incorporation of continuous constraints with additional focus on applications. Sect. 5.2 discusses related work. Sect. 5.3 and Sect. 5.4 describe how to incorporate discrete constraints into the learning of ELMs with certain guarantees on their satisfaction. Sect. 5.5 contains three different experiments obtained to investigate the impact of continuous constraints on the learning of ELMs. The first experiment analyzes the learning of a monotonically increasing one-dimensional mapping. The second experiment investigates an upper-bounded mapping in two dimensions. Finally, learning of the redundant kinematics of the Puma560 robot in a dynamical systems formulation with respect to previously defined joint limits is examined. Most of the content of this chapter is related to [16].

5.1 Reliability via Continuous Constraints

Modern machine learning solutions often need to guarantee certain properties of the learned function to ensure a safe operation of the technical system while simultaneously providing the full power of data-driven modeling. The behavior of classical neural networks is solely determined by the data subject to the particular process synthesizing the data. But this lack of process depending internal structure might be an issue if the network is faced with sparse and noisy data, which

Fig. 5.1: The reliable integration of prior knowledge in form of discrete constraints can be separated into three main aspects: (i) the integration of the constraint into the learning algorithm in discrete points, (ii) the generalization of the point-wise constraints towards the continuous region by means of the generalization capability of the underlying ELM, and (iii) the mathematical ex-post verification that the constraint holds in the learned function.

can lead to inconsistencies with respect to the physics of the underlying process. In cases where predictions are made beyond the training data (extrapolation), inconsistent predictions become even more likely and severe. It is obvious that the consideration of prior knowledge about the task in form of continuous constraints is useful to overcome data limitations and enhances the reliability of the learner.

The acceptance of applied machine learning algorithms usually scales with the degree to which the constraints can reliably be incorporated into the learner, where "reliable" can refer to a high probability, or even a mathematical proof, of success. This task is not trivial because many machine learning algorithms gain their very power from the universal approximation capabilities of their underlying models. The restriction of the approximation capability, i.e. model complexity, to enhance the reliability is thus no adequate solution. In addition, data in real world applications are typically noisy, which leads to an natural risk of overfitting corrupted data. A constraint that is present in the ideal function, from which the training data are sampled, may therefore be violated by the learned function. A general approach to incorporate continuous constraints efficiently is missing.

Yet, model selection strategies are entirely specific for these constraints and can not easily be found for complex constraints. The issue of incorporation becomes even more complex due to the large variety of such constraints. Constraints can appear only locally in a possibly lower dimensional subregion of the input space or globally in the whole input space. Constraints may be defined only at certain discrete points or in a continuous region. The latter is of course the more complex case: only a discrete number of data points can be input to the learner, but the constraint must be generalized to the full region under question. The reliable integration of prior knowledge in form of constraints can be separated into three main aspects: (i) the integration of the constraint into the learning algorithm in discrete points, (ii) the generalization of the point-wise constraints towards the continuous region by means of the generalization capability of the underlying learner, and (iii) the mathematical ex-post verification that the constraint holds for the learned function in the continuous region. (i), (ii) and (iii) are interconnected if the learning algorithm directly guarantees the constraint. The key idea is thus

to iterate steps (i) and (ii) for learning and (ii)-verification successively. Fig. 5.1 schematically illustrates this idea.

This thesis shows that the particular form of the extreme learning machine with its fixed input weights and high-dimensional hidden-layer representation, allows for an efficient and flexible incorporation of continuous constraints in the learned function without losing the universality of the ELM approach. To this aim, the actual incorporation of the continuous constraint is done by an incremental sampling method which successively implements constraints in discrete points via quadratic programming in a highly efficient manner, i.e. learning refers to minimization of the standard square error under additional linear constraints. The goal is that the discrete constraints are then generalized by the ELM towards a continuous input space region called workspace[1]. It turns out that all constraints which can be expressed as linear inequalities involving arbitrary derivatives of the learned functions and multiple dimensions can be incorporated. This is possible because partial differentiation of a function which is implemented by an ELM, can directly be performed due to the special form of the ELM with its fixed input weights. It is furthermore shown that the inherent generality of the proposed approach can be used to implement complex forms of prior knowledge such as stability of the to-be-learned dynamics. This is achieved by connecting multiple outputs via a continuous constraint induced by a Lyapunov function and demonstrated practically in Chap. 7. Besides these features, the locality of the approach prevents a direct guarantee of the satisfaction of the constraint in the continuous region. However, as the neural network model allows analytical differentiation, it is shown that the verification can be constructively and effectively proven ex-post if needed.

5.2 Related Work

The incorporation of specific kinds of prior knowledge was tackled before in several machine learning models. Many of those deal with the embedding of bounds on the derivatives [132, 133] or minimum/maximum output control [134] into feed-forward learning structures. Some approaches use a-priori model selection to guarantee certain properties, like non-negativity in non-negative matrix factorization (NMF) [135], or limits of the model complexity. In automatic control, for example, constraints like maximum energy of the control signal, monotonicity or smoothness are often required to ensure safe operation of the plant [136]. This is prior knowledge in the sense that certain desired relations between inputs and outputs are known in advance.

There are also attempts to structure the embedding of prior information into classes. For instance in [137], three main classes are distinguished: structural, data, and the algorithm class. An example for structural methods is the so called

[1]The workspace is the set of inputs that can appear in typical situations during application of the model.

hybrid model [138, 139], which integrates two sub models: a partial first principles model, which incorporates the available prior knowledge about the process being modeled and a neural network which serves as an estimator of unmeasured process parameters that are difficult to model from first principles. Niyogi et al. presents a data driven approach in [140] which creates virtual samples and is element of the data class. Also Burges uses virtual samples to bias a support vector machine [141]. This is reasonable because most of the learning algorithms assume noise-free samples which can cause a poor generalization performance. Selections of specific kernels [142] and constraints on the search bias [143] are in the algorithm class.

Interestingly, Yu et al. defines prior knowledge similar to Abu-Mostafa's hypotheses as "all the auxiliary information about the learning task [...] that[2] can be used to guide the learning process" ([144], p. 8), while the information typically "comes from either related discovery processes or domain experts" ([144], p. 8). They give an additional but similar structuring for approaches incorporating prior domain knowledge into inductive machine learning. Existing methods are classified due to the use of prior domain knowledge, which is either used 1) to prepare training samples, 2) to initiate the hypothesis space, 3) to alter the search objective, or 4) to augment the search. Since inductive learning is investigated, the authors essentially state that "[...] any learning algorithm already makes assumptions apart from the training data [...] which cannot replace prior[3] knowledge, and can be expressed mathematically as: Generalization = Data + Knowledge" ([145], p. 6). It is nevertheless clear that the prior knowledge implemented by the algorithm is task specific according to the no free lunch theorem [146] and needs to be chosen carefully. Learning by constraints as proposed in this thesis uses prior knowledge to alter the search objective and therefore belongs to the third class of Yu's taxonomy.

One big problem is that most methods can only integrate specific types of prior knowledge in continuous workspace regions. In order to embed prior knowledge in a more generic way, continuous constraints on the optimization process appear to be promising. The estimation of parameters $\mathbf{w}$ minimizing a non-linear (error) function $f(\mathbf{x}^i, \mathbf{w})$ subject to a constraint $c(\mathbf{w}, \mathbf{x}) \leq 0$ for training input stimuli $\mathbf{x}^i$ in a continuous region Ω is called semi-infinite programming (SIP) problem, because, in principle, the solution must fulfill the constraint for infinitely many inputs $\mathbf{x} \in \Omega$. Yet, general solutions to the SIP problem are either computationally expensive because f or c are highly non-linear in $\mathbf{w}$ or strongly restricted because f is only linear or quadratic in $\mathbf{x}$. [147] gives a comparative review on several solutions to the SIP problem.

First efforts towards neural networks, encoding a-priori information, can be found in [148]. The paper proposes two different approaches for incorporation: constraints on the architecture and/or constraints on the connection weights. The

[2]The word "that" is an additional comment by the author and only added for the sake of readability. It is not part of the original comment.

[3]The word "prior" is an additional comment by the author and not in the original comment.

authors show that it is possible for monotonic and concave behavior to transform the infinitely many constraints of the SIP problem into a finite number of constraints which only involve the network weights. The method applies backpropagation rules for learning and forces all network outputs to this specific behavior which restrict learning to rather simple and low-dimensional domains.

Abu-Mostafa and Sill connected so-called "hints" to constraints on the objective function via virtual samples [149, 150]. Such hints are used as auxiliary information about the target function which is learned in a data-driven process. Hints are also used as additional regularization when added to the objective function which punish the violation of previously defined prior knowledge. The ill-posed problem is thus transformed into a well-posed problem, however, without regards to continuous constraints. A neural network solution that is able to embed general prior knowledge by learning is introduced in [151]. The authors present backpropagation rules for MLPs and also for single hidden-layer networks that respect additional terms in the error function to constrain arbitrary partial derivatives of the network's output. However, training these networks is computationally very expensive. A neural network architecture that includes functional prior knowledge in form of non-linear functions to enhance the generalization capability is proposed in [152]. Learning is done by a gradient based l_2-optimization technique and thus potentially suffer all known drawbacks of gradient descent. [153] investigates prior knowledge integration with a particular focus on the universal approximation of the model. Learning is achieved by means of conjugate gradient descent. The method introduced in [154] also allows the incorporation of prior knowledge, however, with application of standard non-linear programming techniques which renders learning computationally expensive as well. Attempts to use prior knowledge to initialize neural networks can, e.g., be found in [155]. The paper investigates specific prior knowledge expressed in form of proportional rules for so-called rapid backpropagation networks. The generalization ability and convergence speed of the networks is enhanced significantly in comparison to networks initialized without prior knowledge. In [156], Lauer et. al. introduce an approach to incorporate various kinds of prior knowledge into support vector regression. However, the method only considers discrete constraints without generalization towards a continuous workspace region and can therefore not implement continuous constraints. One of the predecessors of this method is presented by Mangasarian et al. in [157]. This method introduces a learning problem in the form of a linear program which also includes the prior knowledge in the learning as a finite set of inequalities. In this case, simpler constraints as for the proposed approach are considered: Mangasarian's method does not include prior knowledge on the derivatives of the function.

A model based on radial basis function (RBF) networks focusing on continuous constraints is introduced by Hu et al. in [137] and further investigated in [158]. This approach implements prior knowledge by a sampling method comparable to the method in [156]. The model also suggests a strategy to incorporate continuous priors in an architectural way, i.e. changing the feature pool by addition of

basis functions, based on a global maximization step which is essential but not further described. Another method also dealing with continuous constraints on a theoretical level was proposed in [159] and is based on kernel expansion and regression. Sampling is also used for the implementation of continuous constraints. However, the main disadvantage of this abstract formulation is that no specific information about the optimization problem is available. This implies that optimization is, in general, only achievable with non-linear programming techniques which renders learning computationally expensive. Furthermore, guarantees are restricted to probability ranges because an efficient verification algorithm for this method is missing. Another drawback of this method is that the outputs are treated independently which restricts the class of implementable prior knowledge strongly. Also the prior knowledge integration into ELMs was focused earlier in a very limited manner [160, 161]. The ELM model introduced in [160] is not capable of approximating universal functions and does not handle continuous constraints. The symmetric ELM [161] only considers prior symmetry assumptions. Tab. 5.1 shows a summary of the mentioned related work, respective features, and citations.

The following section shows that the proposed approach to incorporate continuous constraints via successively applied quadratic programming into ELMs is an effective means to represent the SIP problem. The quadratic error functional $f(\mathbf{x}^i, \mathbf{w}) = E_{\text{tr}}(\mathbf{x}^i, W^{\mathbf{out}})$ for training inputs $\mathbf{x}^i$ subject to continuous constraints $c(\mathbf{x}, \mathbf{w}) = C(\mathbf{x}, W^{\mathbf{out}}) \leq 0$ appears to be quadratic in the learning parameters $\mathbf{w} = W^{\mathbf{out}}$ and highly non-linear in the input $\mathbf{x}, \mathbf{x}^i \in \Omega$, whereas the constraint function is linear in the learning parameters $\mathbf{w} = W^{\mathbf{out}}$ and also highly non-linear in the input $\mathbf{x}$. This distinctiveness is due to the inherent functional separation between random hidden-layer projection and read-out weights and thus implies that the proposed ELM approach is a general, feasible and computationally efficient solution for SIP.

Tab. 5.1: Overview: related methods applying prior knowledge.

Method	**Model Basis**	**Prior Knowledge**	**Safety**	**Learning**	**Citations**
Monotonic MLP	MLPs, RBF nets	Monotonicity (continuous) via architectural bias (model selection)	Proof	Backpropagation	[132, 136, 133]
Hints	MLPs, RBF nets	Hints via virtual samples (discrete)	Probability	Backpropagation	[149, 150]
Prior knowledge MLP	MLP, Single layer networks	Bounds on derivatives via error function (continuous)	Probability	Backpropagation	[151]
Prior knowledge SVM	Support vector machine	Linear inequality constraints (discrete)	Proof	Linear programming	[156]
Generalized constraint neural network	Hybrid: RBF net, constrained RBF net	Linear inequality constraints (continuous/discrete) via architectural bias	Probability	Quadratic programming	[137]
Cutting plane method	Conceptional basis (kernel methods)	Inequality constraints on single output dim. (continuous/discrete)	Probability	Non-linear programming	[159]
Symmetric ELM	ELM (model selection)	Symmetry (continuous)	Proof	Linear regression	[161]

5.3 Embedding Discrete Constraints into ELMs

This section describes how discrete constraints are incorporated into the learning of extreme learning machines (ELM). The error functional used to obtain the closed form for the read-out weights via ridge regression is interpreted as a quadratic program which explicitly embeds constraints in discrete points $\mathbf{u}$. Afterwards, the effective generalization of the constraints towards a continuous region is explained.

5.3.1 Incorporation of Discrete Constraints via Quadratic Programming

Many constraints on a desired function can be expressed by bounding the output $\mathbf{y}(\cdot)$ (see Eq. (2.5)) or its partial derivatives. Hence, I generally refer to a constraint $C(W^{\mathbf{out}}, \mathbf{u})$ for input $\mathbf{u} \in \Omega$ with respect to the read-out parameters $W^{\mathbf{out}}$ as a linear combination of partial derivatives with parameters $\gamma_i \in \mathbb{R}$ and bound $c \in \mathbb{R}$ of the form:

$$C(W^{\mathbf{out}}, \mathbf{u}) = \sum_i \gamma_i D^{\mathbf{m}^i} \hat{y}_i(\mathbf{u}) = \sum_i \gamma_i W_i^{\mathbf{out}} \cdot D^{\mathbf{m}^i} \mathbf{h}(\mathbf{u}) \leq c. \tag{5.1}$$

$D^{\mathbf{m}^i} = \partial^M / \partial u_{m_1^i} \dots u_{m_M^i}$ is the component-wise differential operator, whereas the vector $\mathbf{m}^i = (m_1^i \dots m_M^i) \in [1, 2, \dots, I]^M$ defines the input dimensions with regard to which the differentiations are carried out. Interestingly, Eq. (5.1) shows that the constraint can be rephrased in terms of the hidden state and is linear in the learning parameters $W^{\mathbf{out}}$ due to the fixed input weights and the functional separation. This defines a linear inequality for each discrete sample $\mathbf{u}$.

If a sample set $U = (\mathbf{u}^1, \dots, \mathbf{u}^{N_\mathrm{s}})$ comprising N_s discrete inputs is given, incorporation of these constraints into the function learned by the ELM is phrased as solving a quadratic program [162, 163] optimizing the read-out weights $W^{\mathbf{out}}$ subject to a system of linear inequalities $C(W^{\mathbf{out}}, U) \leq$. First, a collection of the hidden states $H(X)$ for input stimuli X is phrased in a block-diagonal matrix $\hat{H}$ according to the following definition:

$$\hat{H} := \begin{pmatrix} H(X)^T & 0 & 0 & 0 & \dots \\ 0 & H(X)^T & 0 & 0 & \dots \\ 0 & 0 & H(X)^T & 0 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix} \in \mathbb{R}^{O \cdot N_\mathrm{tr} \times O \cdot R} \ . \tag{5.2}$$

Also the read-out weights $W^{\mathbf{out}}$ and the target values Y are rephrased for appli-

cation in the quadratic program.

$$\hat{W}^{\mathbf{out}} := \begin{pmatrix} (W^{\mathbf{out}}_{1\cdot})^T \\ (W^{\mathbf{out}}_{2\cdot})^T \\ \dots \end{pmatrix} \in \mathbb{R}^{O\cdot R} \text{ and } \hat{Y} := \begin{pmatrix} y_1(1) \\ y_1(2) \\ \dots \\ y_2(1) \\ y_2(2) \\ \dots \end{pmatrix} = \begin{pmatrix} Y_{1\cdot} \\ Y_{2\cdot} \\ \dots \end{pmatrix} \in \mathbb{R}^{O\cdot N_{\text{tr}}} \quad , \tag{5.3}$$

where $Y_{i\cdot}$, e.g., denotes the ith row of matrix Y. These equations allow a redefinition of the quadratic program applied for learning the data set, where Tikhonov regularization is directly attached to the already defined matrices:

$$\begin{gathered} \left|\left| \begin{pmatrix} \hat{H} \\ \sqrt{\alpha} \cdot I \end{pmatrix} \cdot \hat{W}^{\mathbf{out}} - \begin{pmatrix} \hat{Y} \\ \hat{0} \end{pmatrix} \right|\right|^2 \to \min \\ \text{subject to: } A^{\mathbf{ieq}} \cdot \hat{W}^{\mathbf{out}} \leq \mathbf{c} \ , \end{gathered} \tag{5.4}$$

where α is the ridge regression parameter, $I \in \mathbb{R}^{O\cdot R \times O\cdot R}$ is the identity matrix, $\hat{0} := (0, 0, \dots)^T \in \mathbb{R}^{O\cdot R}$ is a collection of zero values, and $A^{\mathbf{ieq}} \in \mathbb{R}^{N_s \times O\cdot R}$ and $\mathbf{c} \in \mathbb{R}^{N_s}$ define the constraint matrices (N_s is the number of point constraints $\mathbf{u}^j$):

$$A^{\mathbf{ieq}} := \begin{pmatrix} \gamma_1 D^{\mathbf{m}^1} \mathbf{h}^T(\mathbf{u}^1), & \gamma_2 D^{\mathbf{m}^2} \mathbf{h}^T(\mathbf{u}^1), & \dots \\ \gamma_1 D^{\mathbf{m}^1} \mathbf{h}^T(\mathbf{u}^2), & \gamma_2 D^{\mathbf{m}^2} \mathbf{h}^T(\mathbf{u}^2), & \dots \\ & \vdots & \end{pmatrix} \text{ and } \mathbf{c} = (c, c, \dots)^T \quad . \tag{5.5}$$

However, to make this approach feasible, also the multi-dimensional differentiation has to be carried out. Fortunately, the special form of the ELM allows to compute a closed form of arbitrary partial derivatives of the different output components analytically as:

$$\begin{aligned} \frac{\partial^M \hat{y}_i(\mathbf{u})}{\partial u_{m_1} \dots u_{m_M}} &= \sum_j W^{\mathbf{out}}_{ij} \frac{\partial^M h_j(\mathbf{u})}{\partial u_{m_1} \dots u_{m_M}} \\ &= \sum_j W^{\mathbf{out}}_{ij} f^{(M)}(a_j \sum_k W^{\mathbf{inp}}_{jk} u_k + b_j) \cdot a_j^M W^{\mathbf{inp}}_{jm_1} \dots W^{\mathbf{inp}}_{jm_M}, \end{aligned} \tag{5.6}$$

where $f^{(M)}$ denotes the Mth derivative of f. Note, that the output of the network can be interpreted as 0th derivative when choosing $M = 0$. Solving the quadratic program now guarantees satisfaction of the given constraints with respect to the discrete samples $\mathbf{u} \in \Omega$ in the input space, which is already useful in many applications.

Independent Outputs. If it is possible to assume independent outputs (also be means of prior knowledge), then the previously defined quadratic program

in Eq. (5.4) becomes more simple. Independent outputs in mathematical terms means that only one or less of the γ's is not equal to zero, i.e. $(\gamma_1, \gamma_2, \dots \gamma_O) = (0, \dots, 0, \varepsilon, 0, \dots, 0)$. The learning program can be reduced to the following program:

$$\begin{gathered} \left\| \begin{pmatrix} H \\ \sqrt{\alpha} \cdot I \end{pmatrix} \cdot W_{i\cdot}^{\mathbf{out}T} - \begin{pmatrix} Y \\ \hat{0} \end{pmatrix} \right\|^2 \to \min \\ \text{subject to: } A^{\mathbf{ieq}} \cdot \hat{W}_{i\cdot}^{\mathbf{out}T} \leq \mathbf{c} \ , \end{gathered} \tag{5.7}$$

where the constraint matrices are defined according to the following redefinition of Eq. (5.5)

$$A^{\mathbf{ieq}} := \begin{pmatrix} \gamma_i D^{\mathbf{m}^i} \mathbf{h}^T(\mathbf{u}^1) \\ \gamma_i D^{\mathbf{m}^i} \mathbf{h}^T(\mathbf{u}^2) \\ \vdots \end{pmatrix} \text{ and } \mathbf{c} = (c, c, \dots)^T \ , \tag{5.8}$$

where i is the only output dimension who's γ_i is not equal to zero. Learning is then applied for each dimension i separately. Note, that finding the solution of the quadratic program is more computationally efficient in this case because of the reduced dimensionality. However, complex forms of prior knowledge include all dimensions and thus needs the connection of multiple output dimensions as in Eq. (5.4). Stability is such a form of prior knowledge and is discussed in detail in Sect. 5.5.3 and Chap. 7

5.4 From Discrete to Continuous Constraints

The next step is to target constraints in a continuous, compact region Ω of the input space, i.e. to generalize the point-wise constraints $\mathbf{u}^i$ to a continuous region Ω. Discrete inputs $\mathbf{u}^i \in \Omega$ are regarded as discrete samples of the continuous constraint. In principle, no finite number of discrete samples $\mathbf{u}^i$ can implement the constraint and at the same time guarantee its generalization to hold in the continuous region without additional verification effort. In fact, the quadratic program only guarantees the satisfaction of the continuous constraint in the points $\mathbf{u}^i$. It can be expected that the generalization ability and the implicit smoothness of the used ELM enables an implementation of the constraint in the whole region Ω with only a limited number of discrete samples. The expectation is therefore that sampling is sufficient for generalization. But then, it is necessary to verify ex-post that the constraint $C(\mathbf{u}) = C(W^{\mathbf{out}}, \mathbf{u}) \leq 0$ (the read-out matrix is omitted and the constraint $C(\mathbf{u}) \leq c$ is equivalently changed to $C(\mathbf{u}) - c \leq 0$ in the following sections for notational simplicity) holds for all $\mathbf{u} \in \Omega$ and not only for the discrete samples $\mathbf{u}^i$.

It is impossible to verify the satisfaction of a given constraint in a continuous region in the input space in closed form, because the universal approximation

capability of the ELM [10] implies that the learned function can in principle be arbitrarily complex. An algorithm to verify the fulfillment of the constraints which is based on a worst case analysis of the Taylor approximation of the learned function is provided. The algorithm's result $P(C, \hat{\mathbf{y}}, \Omega, \varepsilon)$ with respect to the constraint C, reliability margin ε, and the ELM's output $\hat{\mathbf{y}}$ is *true* for region Ω if and only if the constraint C is satisfied by the function $\hat{\mathbf{y}}(\cdot)$ in region Ω with reliability margin ε.

5.4.1 Constraint Sampling

Assume that an arbitrary number of continuous constraints $C_i : i = 1, \dots N$ is given. The constraints are present in the workspace Ω. The goal is to construct N different sample pools $U_i = (\mathbf{u}_i^1, \dots, \mathbf{u}_i^{N_\mathrm{s}^i})$ that are sufficient to generalize the discrete constraints towards the continuous region Ω. As a first step ($k = 0$), the network is initialized randomly and trained without any constraints (the sample matrices U_i^k are empty in the beginning): $U_i^0 = \emptyset : \forall i = 1, \dots, N$. In this case learning can be done in computationally cheap fashion by ridge regression, which is a standard technique for ELMs.

In the next step, N_C samples $\hat{U} = \{\hat{\mathbf{u}}^1, \hat{\mathbf{u}}^2, \dots, \hat{\mathbf{u}}^{N_\mathrm{C}}\}$ are randomly drawn from an uniform probability distribution in Ω. Afterwards, the number of samples ν_i in $\hat{U}$ that fulfill the continuous constraint C_i are determined according to the constraint formulation in Eq. (5.4). The sampling algorithm stops if more or equal than p_i percent (defined by the user) of these samples fulfill the continuous constraints with additional reliability margin $\varepsilon_i > 0$, i.e. $\nu_i / N_C \geq p_i$. Otherwise, the most violating samples $\hat{\mathbf{u}}_j$ for each constraint are added to the sample pool $U_i^{k+1} = U_i^k \cup_j \hat{\mathbf{u}}_j$ to constitute the new sample set U_i^{k+1}. The obtained set of samples is then used for training according to Eq. (5.4). For $\varepsilon_i > 0$ the constraint $C_i \leq 0$ is ε_i-fulfilled in the sense that a violation of size ε_i is tolerated. This guarantees convergence of the sampling algorithm. A pseudo code of the learning procedure is provided in Alg. 5.2.

Algorithm 5.2 Sampling discrete constraints

Require: data set D, region Ω, counter $k = 0$, empty sample pools $U_i^k = \emptyset$
Require: ELM trained with D according to Eq. (2.5)
  repeat
    draw samples $\hat{U} = \{\hat{\mathbf{u}}^1, \hat{\mathbf{u}}^2, \dots, \hat{\mathbf{u}}^{N_\mathrm{C}}\}$
    ν_i = no. of samples in $\hat{U}$ fulfilling $C_i(\hat{\mathbf{u}}) \leq \varepsilon_i$
    determine most violating samples $\hat{\mathbf{u}}_j \leftarrow \arg\max_{\mathbf{u} \in \hat{U}} C_i(\mathbf{u})$
    if $p_i > \frac{\nu_i}{N_\mathrm{C}}$ **then** $U_i^{k+1} = U_i^k \cup_j \hat{\mathbf{u}}_j$
    train ELM with D, and U_i^{k+1} according to Eq. (5.4)
  until $p_i > \frac{\nu_i}{N_\mathrm{C}} : \forall i$

5.4.2 Reliability Verification

To effectively verify the satisfaction of the continuous constraint in the workspace region Ω, a second order Taylor approximation of the constraint C and its corresponding remainder are calculated. The Taylor approximation in $\mathbf{u}^0 \in \Omega$ is given as:

$$C(\mathbf{u}) = T(\mathbf{u}, \mathbf{u}^0) + rem(\mathbf{u}, \mathbf{u}^0) \tag{5.9}$$

$$= K + J^T(\mathbf{u} - \mathbf{u}^0) + \frac{1}{2}(\mathbf{u} - \mathbf{u}^0)^T H(\mathbf{u} - \mathbf{u}^0) + rem(\mathbf{u}, \mathbf{u}^0) \ , \tag{5.10}$$

where $K = C(\mathbf{u}^0)$ is the constant term, $J = \nabla C(\mathbf{u})|_{\mathbf{u}^0}$ is denoting the Jacobian vector, $H = (\nabla\nabla^T)C(\mathbf{u})\big|_{\mathbf{u}^0}$ is the Hessian matrix evaluated at point $\mathbf{u}^0$, and $rem(\mathbf{u}, \mathbf{u}^0)$ is the remainder term. An upper bound for the remainder is computed:

$$rem := \sum_i \gamma_i \sum_j W_{ij}^{\mathbf{out}} M_j \frac{|\underline{u}_j - \overline{u}_j|^3}{6} \ , \tag{5.11}$$

where i is the dimension of the constrained output and γ_i are the coefficients defined as in Eq. (5.1). The used variables are:

$$\underline{u}_j = \min_{\mathbf{u}\in\Omega} W_j^{\mathbf{inp}}\mathbf{u} \ , \ \overline{u}_j = \max_{\mathbf{u}\in\Omega} W_j^{\mathbf{inp}}\mathbf{u} \ , \text{ and} \tag{5.12}$$

$$M_j = \max_{u_j \in [\underline{u}_j, \overline{u}_j]} \left[f^{(M+3)}(a_j u_j + b_j) \cdot a_j^{M+3} W_{jm_1}^{\mathbf{inp}} \dots W_{jm_M}^{\mathbf{inp}} \right] \ . \tag{5.13}$$

The maximization steps in (5.12) can be performed by evaluating the vertices of Ω if the region Ω is a convex polyhedron, e.g. a regular hypercube. The maximization in Eq. (5.13) uses that f can be differentiated analytically if the Fermi-function is used which is one-dimensional w.r. to its argument. This is only feasible due to the simple and elegant form of the ELM. The remainder of the Taylor approximation $rem(\mathbf{u}, \mathbf{u}^0)$ in Eq. (5.10) is bounded from above by rem in Eq. (5.11) in region Ω:

$$rem \geq rem(\mathbf{u}, \mathbf{u}^0) : \forall \mathbf{u} \in \Omega. \tag{5.14}$$

A proof of the proposition in Eq. (5.14), is given in the appendix, see Sect. A.2. Since T (see Eq. (5.10)) is a polynomial of second order, it is possible to find its global maximum and minimum in Ω analytically. In order to find a worst case approximation of the learned function, the following is tested and a recursive step is performed:

$$P(\Omega) = \begin{cases} 1 & : \text{if } \max\limits_{\mathbf{u}\in\Omega} \left[T(\mathbf{u}, \mathbf{u}^0)\right] < \varepsilon - rem \\ 0 & : \text{if } \min\limits_{\mathbf{u}\in\Omega} \left[T(\mathbf{u}, \mathbf{u}^0)\right] > \varepsilon + rem \\ \bigwedge_i P(\Omega_i) & : \text{otherwise} \end{cases} \tag{5.15}$$

where $\mathbf{u}^0 \in \Omega$ is the base of the Taylor approximation and $\Omega = \bigcup_i \Omega_i$ is divided into the pairwise disjoint subregions Ω_i.

A schematic view of the decision process in Eq. (5.15) is given in Fig. 5.2. The figure shows an one-dimensional mapping with constraint $C \leq 0$ and safety margin ε and three subregions (Ω_{i-1}, Ω_i, and Ω_{i+1}) where the satisfaction of the constraint within the reliability margin is tested. The center of each region $\mathbf{u}^0$ defines the respective Taylor polynomial and the remainder estimate rem of L. The regions show the possible cases of Eq. (5.15):

1. The maximum of the Taylor polynomial in region Ω_{i-1} is below $\varepsilon - rem$. The constraint is fulfilled and the ouput the the verification algorithm is $P = 1$. In mathematical terms: $\max_{\mathbf{u} \in \Omega_{i-1}} \left[T(\mathbf{u}, \mathbf{u}^0)\right] < \varepsilon - rem$.

2. The minimum of the Taylor polynomial in region Ω_{i+1} is above $\varepsilon + rem$ and therefore $P = 0$ - there exists at least one point in this region where the constraint is violated. In mathematical terms: $\min_{\mathbf{u} \in \Omega_{i+1}} \left[T(\mathbf{u}, \mathbf{u}^0)\right] > \varepsilon + rem$.

3. It is not clear whether the constraint in region Ω_i is satisfied or not - a division into smaller subregions is necessary.

An intrinsic feature of Taylor approximations is that the quality of the estimated approximation is the best close to the approximation point $\mathbf{u}^0$ (locality feature): the smaller the region Ω, the better the estimation. This separation is performed until convergence of the algorithm, where an early stop is possible whenever the constraint is definitely not fulfilled in some subregion. For $\varepsilon > 0$, the constraint is ε-fulfilled such that a violation of size ε is tolerated, very similar to the ε-sensitive loss functions frequently employed in support vector regression. In engineering applications, one might also set $\varepsilon < 0$ to guarantee some safety margin, e.g. to balance numerical errors.

5.5 Experimental Results

This section investigates the learning with different continuous constraints for extreme learning machines. The particular focus lies on the learning of sparse and noisy data.

5.5.1 One-Dimensional Mapping with Monotonicity

An ELM with $R = 100$ hidden-layer neurons is supposed to learn an one-dimensional $\mathbb{R} \to \mathbb{R}$ mapping. The input weights and the biases are randomly drawn form an uniform distribution in $[-10, 10]$ and trained by BIP with $\mu_{\text{BIP}} = 0.2$ beforehand. The data is generated by a stairs-like function which is monotonically increasing and subject to Gaussian noise with varying amplitude σ_{noise}. A visualization of this function can be seen in Fig. 5.4. The continuous constraint for learning is

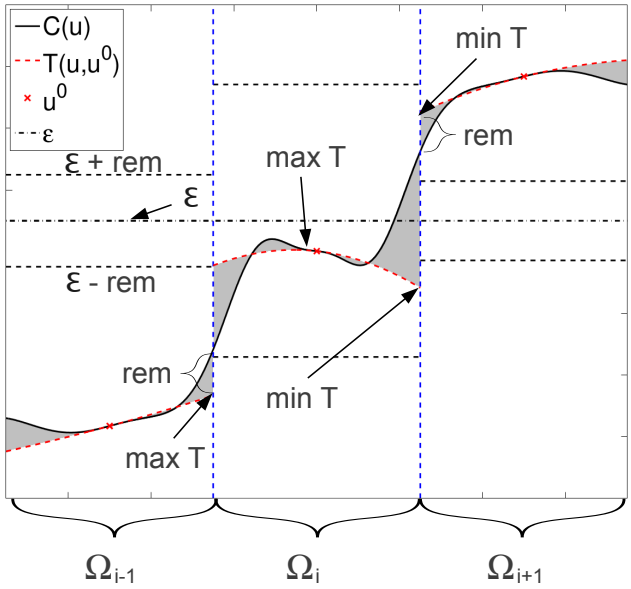

Fig. 5.2: Illustration of the reliability margin verification for an one-dimensional input. The plot shows a constraint $C(\mathbf{u}) \leq 0$, the safety margin ε with a Taylor polynomial $T(\mathbf{u}, \mathbf{u}^0)$ of second order, and subregions Ω_i, Ω_{i-1}, and Ω_{i+1}.

monotonicity which is equivalent to a positive first derivative of the network's output $y(x)$:

$$\frac{\partial}{\partial x} y(x) \geq 0 : \forall x \in \mathcal{I} \quad \Leftrightarrow \quad -\frac{\partial}{\partial x} y(x) \leq 0 : \forall x \in \mathcal{I} \ , \tag{5.16}$$

where $\mathcal{I} = [-1, 1]$ is the input interval. The constraint sampling algorithm searches in a sample pool with $N_\mathrm{C} = 10^4$ samples in each step until the continuous constraint is satisfied with a probability of $\mathrm{p} = 1.0$. Three different experiments are conducted. The first experiment investigates the impact of the data set size, the second experiment analyzes the role of noise, and the third experiment examines the influence of the regularization parameter for the implementation of the continuous constraint.

In the first experiment, learning is conducted for data sets with different training set sizes reaching from $N_\mathrm{tr} = 30$ to $N_\mathrm{tr} = 300$ samples in a logarithmic scale. The number of constraint samples needed to successfully implement the continuous constraint according to Eq. (5.16), training, and test errors are recorded. The experiment is averaged over one-hundred network initializations. The noise level for this data set is $\sigma_\mathrm{noise} = 0.15$ and the regularization parameter $\alpha = 10^{-8}$ is previously defined. Fig. 5.3 (first row, left plot) shows the experimental results for the recorded number of constraint samples. It is clearly visible that a decreasing number of discrete constraints is necessary to implement monotonicity into the network's mapping if the data set size is increasing. This is due to the fact

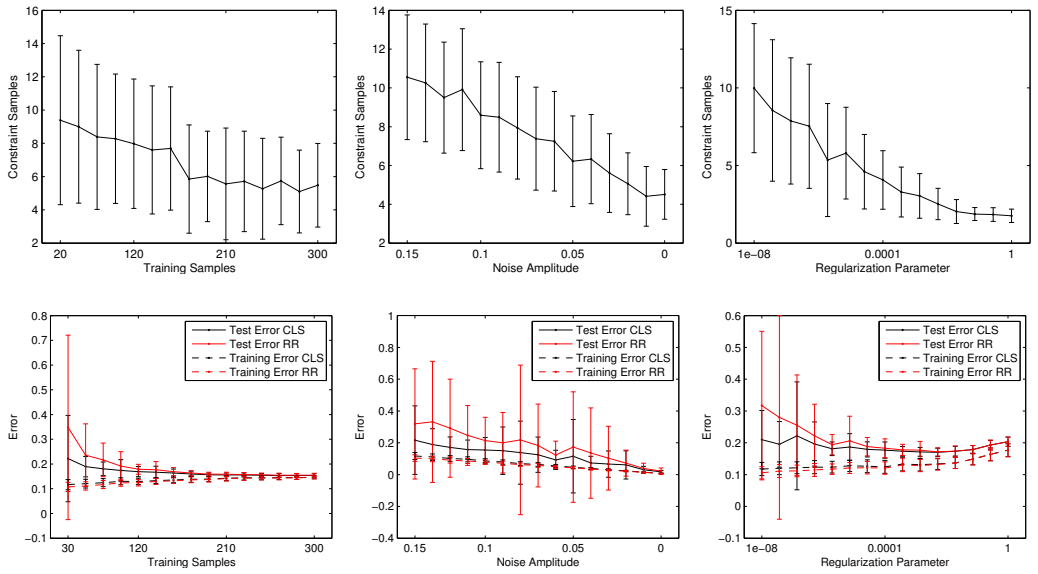

Fig. 5.3: Investigations on an one-dimensional mapping. The number of samples needed in order to implement a continuous constraint (first row) and corresponding training and test errors (second row) for data sets with different sizes (left), noise levels (center), and regularization parameters (right).

that the noise can be eliminated by averaging of the target data. It is also visible in Fig. 5.3 (second row, left plot) that the errors for networks with (denoted by CLS in the plots) and without (denoted by RR in the plots) continuous constraint decrease with growing size of the data set. The networks provided with small data sets highly profit from the implementation of the continuous constraint which enhances the generalization capability of the ELMs indicated by the low test error.

A corresponding statement holds for the learning procedure when noise is considered, see Fig. 5.3 (center column). In this experiment, the number of training samples $N_{\mathrm{tr}} = 30$ and the regularization parameter $\alpha = 10^{-8}$ are fixed. The noise amplitude is varied from $\sigma_{\mathrm{noise}} = 0.15$ to $\sigma_{\mathrm{noise}} = 0.01$. Fig. 5.3 (first row, center plot) demonstrates that a decreasing number of constraints are needed to implement the continuous constraint with decreasing noise level of the input data. Also the errors of the networks are shrinking for lower noise levels indicated by the results contained in Fig. 5.3 (second row, left plot). The influence of the constraints enhances the generalization capability in particular for the networks coping with noisy data sets.

The results of the third experiment are depicted in Fig. 5.4 (third column). The experiment investigates the influence of regularization on the performance of the networks. $N_{\mathrm{tr}} = 30$ samples, contaminated by Gaussian noise with an

amplitude of $\sigma_{\text{noise}} = 0.15$, are used for training. The regularization parameter is varied from $\alpha = 10^{-8}$ to $\alpha = 10^{0}$ logarithmically. It is shown that the number of constraints needed for implementation is decreasing with increasing regularization. This is expected because the model complexity is lower and the generalization of the discrete constraints towards the continuous region becomes easier. Networks with low regularization have a significantly better test error when provided with the continuous constraint and thus profit more form the prior knowledge than networks with high degree of regularization. Networks with strong regularization are not capable to capture the structure of the data to a high degree and thus already implement the continuous constraint which is intrinsically encoded in the data.

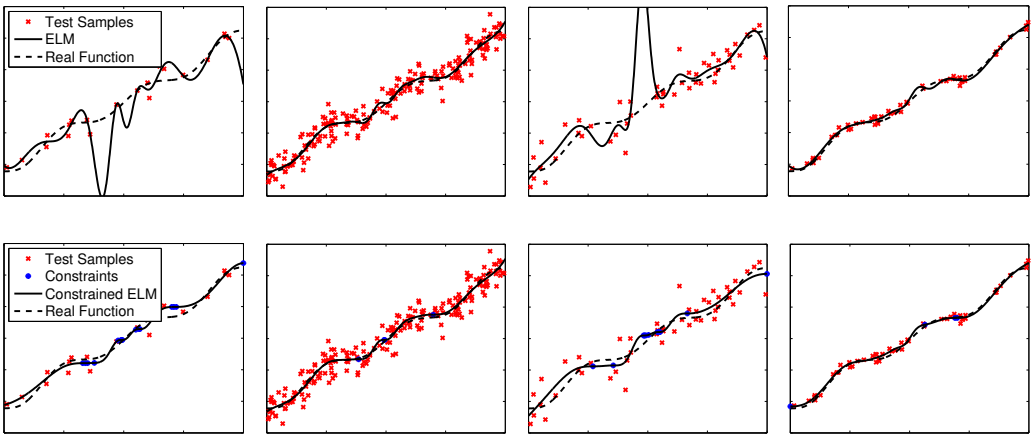

Fig. 5.4: Investigations on an one-dimensional mapping. Comparison of the learning without (first row) and with (second row) continuous constraint (monotonicity) on a noisy and small data set (first column). Learning of a big data set (second column). The learned mapping for a noisy and medium size data set (third column). Learning on a medium size data set with low noise level (fourth column). Extreme learning machines coping with sparse and noisy data clearly profit from incorporation of continuous constraints.

The different stages of the results are illustrated in Fig. 5.4. The first row of the figure contains the results of the networks without respect to the continuous constraint. The second row shows the results with an particular consideration of monotonicity in $\mathcal{I}$, see Eq. (5.16). The first column of Fig. 5.4 shows two network mappings that are trained with a moderate noise level of $\sigma_{\text{noise}} = 0.15$ and only very few samples $N_{\text{tr}} = 30$. The classical ELM networks overfit the data set indicated by strong oscillations visible in Fig. 5.4 (first column, first row). The incorporation of monotonicity according to Eq. (5.16) leads to a smooth solution, see Fig. 5.4 (first column, second row). It is clearly visible that the implementation of constraints leads to a better generalization ability because the overfitting of the samples is

suppressed. The second column of the plot illustrates that this effect becomes smaller with a growing data set size which is already supported by the previous experiments. In this plot, the same noise level of $\sigma_{\text{noise}} = 0.15$ was used and the data set was equipped with $N_{\text{tr}} = 200$ samples. Both solutions start to coincide in the densely sampled regions of the data set. Even the networks without the use of a continuous constraint perform very well. The third column of Fig. 5.4 shows networks that are trained with a high noise level of $\sigma_{\text{noise}} = 0.2$ and a medium size data set of $N_{\text{tr}} = 50$ samples. Fig. 5.4 (first row, third column) shows that the networks without respect to monotonicity are subject to strong overfitting. The positive effect induced by the implementation of the continuous constraint leads again to a good generalization capability and eliminates overfitting, see Fig. 5.4 (second row, third column). Unsurprisingly, the learning results for both networks become significantly better and also coincide when the noise level is decreased (noise is changed to $\sigma_{\text{noise}} = 0.05$ for the same data set size as in the third column) visualized by the fourth column of the figure.

This experiment illustrates the results obtained for the constrained learning procedure for an one-dimensional example where monotonicity is explicitly implemented in form of a continuous constraint. It is shown that the properties of the data set are important to consider when dealing with reliable learning and that the implementation of continuous constraints has a strong positive impact on the generalization of data that only comprise *few* and *noisy* samples. Note, that data sets that are provided with many samples and low noise level only need a small number of constraints in order to implement the continuous constraint in a continuous region. Additionally, the experiments show that the regularization of the networks has a significant impact on the learning with prior knowledge. The higher the regularization, the lower the model complexity, the less samples are needed for successful implementation of the continuous constraint.

5.5.2 Two-Dimensional Mapping with Bounded Maximum Output

An ELM with $R = 100$ hidden-layer neurons is supposed to learn an $\mathbb{R}^2 \to \mathbb{R}$ map of four super-positioned two-dimensional Gaussian functions. The centers of the single distributions are $\mu_{1\ldots4} = (\pm 0.5, \pm 0.5)^T$ and the respective variance of the distributions are $\sigma_{1\ldots4} = \frac{1}{5}$. $N_{\text{tr}} = 500$ samples are used for training, $N_{\text{te}} = 500$ for testing, and $N_{\text{C}} = 10^4$ for the sample pool. Also Gaussian noise with an amplitude of 0.1 is added to the training data. The network is trained by BIP with $\mu_{\text{BIP}} = 0.2$ beforehand. The regularization parameter $\alpha = 10^{-4}$ is obtained by line search. Simultaneously, an artificial constraint is supposed to be satisfied by the network after learning: the maximal output of the network is restricted to

$$y(\mathbf{x}) \leq 0.4 = c \;\; \Leftrightarrow \;\; y(\mathbf{x}) - c \leq 0 \text{ for all } \mathbf{x} \in \Omega = [-1, 1]^2 \;\; , \tag{5.17}$$

where c is the output bound. Note, that the output of the network is interpreted as 0th derivative according to Eq. (5.1) and Eq. (5.6) and is thus consistent with the proposed framework when choosing $M = 0$. In each step, the algorithm adds the 3 most violating sample constraints to the learning pool U until no violation can be assessed anymore ($p = 1.0$). The reliability margin for the discrete constraints is set to $\varepsilon = 0.01$ which enforces the output in the discrete samples to stay below $c = 0.4$ and permits a maximal output of $c+\varepsilon = 0.4+0.01 = 0.41$ in the remaining continuous region. A cell is divided along its longest side into two equal parts, if the satisfaction was not verified in the respective step to a desired degree. This defines the structure of the subregions.

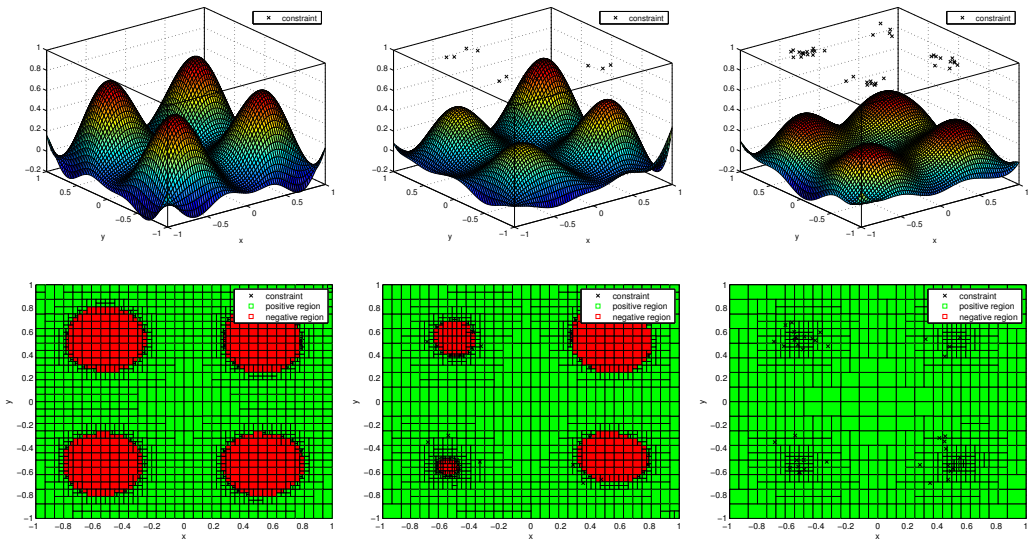

Fig. 5.5: Iteration of the constrained learning algorithm. Green boxes show regions where the constraint is verified, red boxes where the constraint is violated and crosses mark samples in U obtained by the iteration process. The maximal output bound is strongly violated by the unconstrained learner (left). The output of the network shrinks in each step of the iteration (center). The constraint is satisfied and proofed (right).

Fig. 5.5 summarizes the results of the task. The first row shows the outputs of the network in the first, fourth and the last (12th) iteration of the learning, while the second row visualizes the corresponding outcome of the verification algorithm. In the plots of the second row: green boxes show regions where the constraint is verified, red boxes where the constraint is violated and crosses mark samples in U obtained by the iteration process. Obviously, the constraint is violated before the iteration starts - see Fig. 5.5 (left column). After some iterations of the algorithm, the maximal output of the network in Ω shrinks due to the addition of samples of

the constraint. The right hand side of the figure demonstrates that the constraint is fulfilled after twelve iterations, e.g. on basis of 36 discrete $\mathbf{u}_i$ sampling the constraint. Note that the samples are only placed where the mapping produces high output values and thus violate the continuous constraint in Eq. (5.17). Moreover, the cells produced by the recursive step are small in the region where the network's output y is close to the upper bound c. This is due to the fact that the remainder rem becomes big in comparison to the difference of network output y and the output bound $c = 0.4$. Note, that it was already shown in [17] that the proposed sampling strategy is superior to a random sampling of the continuous constraint.

5.5.3 Learning the Redundant Kinematics of the Puma560 Robot

Practicable representations of robot kinematics is an essential tool in the field of robotics. Especially, if the robot is capable of using many degrees of freedom, which allows the robotic system to solve tasks in a highly flexible way using the redundancy of the robot. The kinematics of a robotic platform can be separated into a well-defined forward model (mapping from joint to task space) and it's inverse mapping (mapping from task to joint space), namely the inverse kinematics. This mapping is highly ill-posed for redundant robots and therefore difficult to perform due to the non-convexity of the solution sets.

One way to deal with the control of robots is to learn the kinematics. In particular, the learning of inverse kinematics was studied in the past with different focuses. Basically, the approaches differ in one main aspect: either the encoding of the kinematics is done by solving the direct inverse kinematics, i.e. on the position level or the differential inverse kinematics are solved on the so-called velocity level. In [164], the differential inverse kinematics is learned by locally weighted projection regression, where global consistency is achieved by selectively choosing one of several trained linear models by partitioning the input space. A comparable idea is given by [165]. The approach uses a gating network combining local models. The approach in [166] focuses on a mapping approximation on the positioning level, where structured output learning is applied. A probabilistic model in the input-output space is learned and used for prediction of target outputs. An approach implementing dynamical systems with attractors for using robots with redundant manipulators was introduced in [167]. However, this model cannot guarantee stability and was not used for real robot applications. In contrast, [168] applies a recurrent neural network controller. The network is stabilized by an integral condition which needs to be tuned precisely.

Besides the fact that learning-driven methods represent appealing new approaches for the control of robots in comparison to analytical models, most tasks in robotics still need safety in many respects. However, it is inherent to many learning methods, that the worst case while application cannot be estimated accurately. It is still challenging to use the advantages of learning while simultaneously ensuring a safe application.

Very recently, Reinhart et al. in [169] proposed a method highly inspiring the approach introduced in this illustrative example. The kinematic information is stored in a bi-directionally used network structure with output feedback resolving ambiguity of the inverse kinematics by means of multi-stable attractor dynamics. However, the approach in [169] lacks a direct stabilization of the dynamics in order to enhance the systems safety. Therefore, a relative encoding of the dynamics is introduced which can be shaped explicitly by additional constraints and guarantee a stable behavior. It is shown that the stabilization through additional constraints does not affect the performance of the inverse estimate in a negative way and leads to stable and accurate results for robot control. A simulation of the Puma560 robot is applied, using Corke's tool box [110].

Neural Learning Approach:
Extreme Learning Machine, State Prediction, and Stability

Assume that a data set $D = (\mathbf{x}_i, \mathbf{q}_i) : i = 1, \dots N_{\text{tr}}$ comprising N_{tr} samples is given. Each data pair $(\mathbf{x}_i, \mathbf{q}_i)$ encodes a valid combination of end-effector position $\mathbf{x}_i$ and corresponding joint angles $\mathbf{q}_i$. Multiple solutions of the inverse kinematics can also appear in this data set because of the robot's redundancy, i.e. there might exist pairs $(\mathbf{x}_i, \mathbf{q}_i)$ and $(\mathbf{x}_j, \mathbf{q}_j)$ with $\mathbf{x}_i = \mathbf{x}_j$ and $\mathbf{q}_i \neq \mathbf{q}_j$.

The idea is to use the data set D in order to learn a dynamical system to solve the redundant inverse kinematics of the robot. The dynamical system is parameterized by the desired end-effector position $\mathbf{x}$ and forces the initial state $\mathbf{q}^0$ to converge towards a valid solution of the inverse kinematics $\mathbf{q}$ for end-effector configuration $\mathbf{x}$. This allows the network to encode several solutions $\mathbf{q}_i$ to one end-effector position $\mathbf{x}$ in order to cope with the redundancy of the Puma560. The selection of this solution is dependent on the initial state $\mathbf{q}^0$ of the dynamical system.

Desired dynamics - in particular the contraction of the state towards the desired attractor which encodes the inverse kinematics of the robot - is implemented by state prediction [169]. The state prediction approach extends the training data with synthesized sequences to facilitate attraction to the data samples.

An important issue to cope with is that learning of dynamical systems is typically prone to instabilities. The system's state might diverge due to unstable behavior. The network is stabilized by an additional continuous constraint which forces the state of the system to stay in the robot's joint limits such that it is always close to the training data.

Network Architecture: Extreme Learning Machine. The network comprises three layers: $\mathbf{u} \in \mathbb{R}^I$ collects the input, $\mathbf{h} \in \mathbb{R}^R$ the hidden, and $\mathbf{y} \in \mathbb{R}^O$ the output neurons. The input matrix $W^{\mathbf{in}} \in \mathbb{R}^{R \times I}$ semantically separates the input $\mathbf{u}^k = (\mathbf{x}, \mathbf{q}^k)^T$ into parameters $\mathbf{x} \in \mathbb{R}^X$ and network state $\mathbf{q}^k \in \mathbb{R}^Q$ at time step k, where the input is satisfying $X + Q = I$. The output of the ELM is then

used to compute the next state of the network by discretization of the continuous dynamics:

$$\mathbf{q}^{k+1} = \mathbf{q}^k + \Delta t \cdot \hat{\mathbf{y}}(\mathbf{u}^k) \ , \tag{5.18}$$

where Δt is a time constant, k is the current time step of the iteration, and $\hat{\mathbf{y}}(\mathbf{u}^k)$ is the output of the ELM with input $\mathbf{u}^k$.

Programming Dynamics: State Prediction. In order to imprint suitable dynamics into the ELM, the programming dynamics approach recently introduced in [169] called state prediction (SP) is used and adapted for the model introduced in the previous paragraph. This approach extends the training data with synthesized sequences to facilitate attraction to the data samples. Sequences $\hat{\mathbf{q}}_n^s(k) = \mathbf{q}_n + (1 - \frac{k}{K})^2 \nu^s$ are generated for each input sample pair $\mathbf{x}_n$ and $\mathbf{q}_n$ in the data set; s denotes the index of the S sequences, $\nu^s \in \mathbb{R}^Q$ is a small perturbation, $k = 1, \ldots, K$ are the time steps, and $n = 1, \ldots, N$ is the index of the training samples. This imprints $\mathbf{q}_n$ as a valid attractor of the dynamical system.

The corresponding hidden network states $\mathbf{h}(\mathbf{x}_n, \hat{\mathbf{q}}_n^s(k))$ for each sequence are collected. Attraction to the desired outputs is enforced by mapping states $\mathbf{h}(\mathbf{x}_n, \hat{\mathbf{q}}_n^s(k))$ to outputs $\hat{\mathbf{q}}_n^s(k+1) - \hat{\mathbf{q}}_n^s(k)$. This can be accomplished efficiently by applying ridge regression:

$$W^{\text{out}} = (H^T H + \alpha \mathbb{I})^{-1} H^T \hat{Q} \ , \tag{5.19}$$

where

$$H = \begin{pmatrix} \mathbf{h}(\mathbf{x}_1, \hat{\mathbf{q}}_1^1(1))^T \\ \vdots \\ \mathbf{h}(\mathbf{x}_n, \hat{\mathbf{q}}_n^s(k))^T \\ \vdots \\ \mathbf{h}(\mathbf{x}_n, \hat{\mathbf{q}}_N^S(K-1))^T \end{pmatrix} \text{ and } \hat{Q} = \begin{pmatrix} \hat{\mathbf{q}}_1^1(2)^T - \hat{\mathbf{q}}_1^1(1)^T \\ \vdots \\ \hat{\mathbf{q}}_n^s(k+1)^T - \hat{\mathbf{q}}_n^s(k)^T \\ \vdots \\ \hat{\mathbf{q}}_N^S(K)^T - \hat{\mathbf{q}}_N^S(K-1)^T \end{pmatrix} \tag{5.20}$$

collects the hidden states and the desired outputs in the respective matrices and α modulates the trade-off between weight decay and training error. The perturbations $\nu^s \in \mathbb{R}^Q$ are chosen randomly by a multivariate Gaussian distribution with $\Sigma = \sigma \cdot \mathbb{I}$ centered around the origin:

$$\nu^s \leftarrow \mathcal{N}(\mathbf{0}, \Sigma) \ , \tag{5.21}$$

The presented approach differs from the original work in [169] by using a relative instead of an absolute encoding of the output. The network therefore induces continuous dynamics whose discretization is explicitly controlled through Δt. This yields $S = 7$ sequences with $K = 5$ time steps for training in the experiments, which enlarges the training set by a factor of $S \cdot (K-1)$. However, learning is still feasible due to the efficient learning scheme of the ELM approach.

Stabilization of Neural Dynamics by Continuous Constraints. It was already shown in [169] that the presented approach is able to learn the kinematics of a complex robot. However, this method lacks a direct stabilization mechanism which is desired in order to control the robot in a safe manner. Therefore, a continuous constraint which forces the programmed dynamics to stay in the predefined joint limits is introduced. It is shown in the following section that this technique stabilizes the dynamics while simultaneously sustaining the performance of the kinematic control.

The joint limits of the Puma560 robot are set to $[-\pi, \pi]$ in the experiments. This defines a hypercube where each point $\mathbf{x} \in \Omega$ on the surface of this cube $\partial\Omega$ is mapped onto an uniquely defined normal vector $\mathbf{n}(\mathbf{x}) \in \mathbb{R}^d$ with $\|\mathbf{n}(\mathbf{x})\| = 1$. The normal vector is pointing towards the outside of the cube. Samples are drawn from an uniform distribution on the surface Ω of the cube. The resulting continuous constraint is expressed as a scalar product between the normal vector $\mathbf{n}(\mathbf{x})$ and the network's output $\mathbf{y}(\mathbf{x})$:

$$\mathbf{n}(\mathbf{x})^T \cdot \mathbf{y}(\mathbf{x}) \leq 0 : \mathbf{x} \in \Omega \ . \tag{5.22}$$

This continuous constraint is valid because the scalar product is linear and forces the network's dynamics to stay inside the hypercube and thus stabilizes the dynamical system by avoiding diverging behavior.

6-DOF Robot Arm Kinematics

The neural model is trained to learn the inverse kinematics of the Puma560 Robot whereas the network is restricted to control the end effector position only. The architecture of the Puma560 robot arm provides four different solutions in the workspace for the inverse kinematics in this setting. Fig. 5.6 (second row) visualizes the Puma560 robot performing a movement in the four different joint settings. The tool box [110] which provides a framework to obtain the solutions analytically is applied in the experiments and for visualization. $N_{\text{tr}} = 500$ data points are generated for training and $N_{\text{te}} = 500$ for testing by sampling the first three joints according to an uniform distribution in between the joint limits. The application of state prediction in order to program the multi stable dynamics yields $S \cdot (K - 1) \cdot N_{\text{tr}} = 7 \cdot (5 - 1) \cdot 500 = 14000$ data points in the training set. These $N_{\text{tr}} = 500$ samples are equally divided into four parts. The samples of each part are provided with a single solution of the inverse kinematics which is different to the solutions of the other data set parts. This ensures that samples with ambiguous solutions to the inverse kinematic mapping are included in the training set. The continuous constraint is sampled according to the previous section. Only one sample is added per iteration. The network is set up with six input neurons $(\mathbf{x}, \mathbf{q})$ for the end effector position and the actual joint angles and three neurons $\dot{\mathbf{q}}$ which encode the change of the dynamical joint state of the robot. The new state of the robot is obtained by integration according to Eq. (5.18). Networks with $R = 300$ neurons

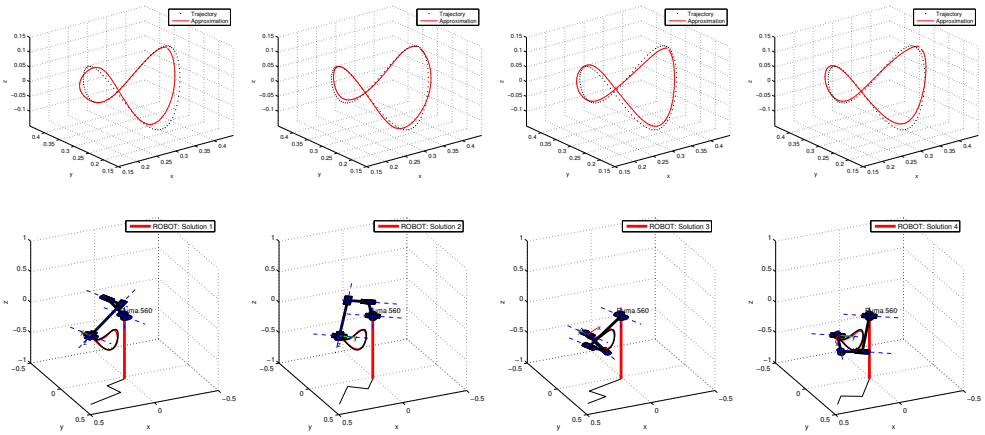

Fig. 5.6: Task space trajectory (first row) vs. joint space trajectory (second row).

in the hidden-layer are used. The input weights W^{inp} and biases b are initialized uniformly in $[-1, 1]$, respectively, and trained by BIP.

Fig. 5.6 shows a setup where the trained networks are used to reproduce a closed loop task space trajectory. Each column visualizes the result obtained for one solution of the inverse kinematics. The first row shows the goal (black dashed line) and the actually reproduced trajectory (red line). The second row shows the robot performing the movement while sustaining one solution of the inverse kinematics. The plot demonstrates that the redundant inverse kinematics are learned with a high degree of precision and that the different solutions are stable in a comprehensive part of the workspace.

The capability of the network to encode several solutions simultaneously with a particular respect to stability is evaluated and the results are presented in Fig. 5.7. The three main learning parameters are varied in order to analyze their impact on the performance. The first row summarizes the results for a change of the regularization parameter α. The second row visualizes the performance and stability change for different variances Σ of the basin size, see Eq. (5.21). Finally, the last row investigates the effect of sampling the continuous constraint in Eq. (5.22) in the process of learning. The left column plots shows the estimate of the relative violation $\frac{\#\mathbf{x}_a}{N}$, where $N = 10000$ denotes the number of samples $\mathbf{x}$ drawn from an uniform distribution on the surface of the hypercube Ω, and $\#\mathbf{x}_a$ is the number of samples that are above zero according to Eq. (5.22). The second column visualizes the average distance between desired and actually implemented attractor for the four different solutions. For this purpose, the network dynamics in Eq. (5.18) are iterated until convergence, i.e. that no significant change in the state is recognizable anymore.

The first experiment, results shown in Fig. 5.7, demonstrates, on the one hand, that the violation of the continuous constraint is decreasing for an increasing reg-

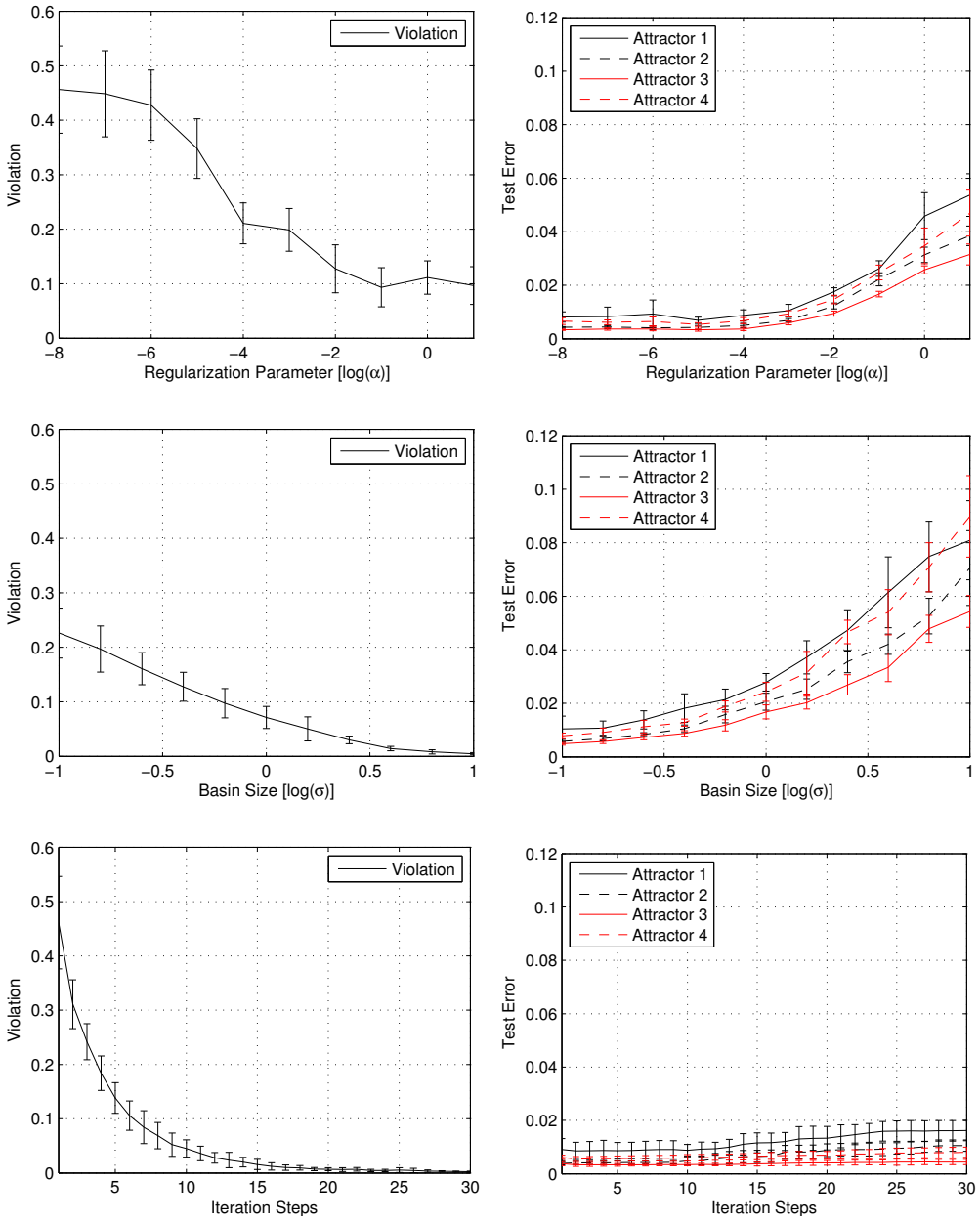

Fig. 5.7: Violation of the continuous constraint (left column) and the error of the attractor implementation (right column). Results for different regularization parameters α (first row) and variations of the basin size σ. The change of the measurements during the iteration of the sampling strategy (third column).

ularization parameter which leads to a stabilization of the dynamics. On the other

hand, the average error of the attractors encoding the redundant solutions of the inverse kinematics is also increasing. The main reason is that the regularization of the read-out weights affects each part of the workspace equally: the surrounding surface of the hypercube, where stability is demanded, and the inside of the cube, where the attractors are located. The same applies to the variance of the Gaussian distribution in Eq. (5.21). The stability is increasing with growing variance, because the basins of attraction are growing with this parameter. However, the borders of attraction between the four solutions are blurred such that the attractors are encoded with less precision - the errors are increasing. The last experiment shows the results for the iteration of the proposed constrained learning procedure during iterative sampling. The dynamics are stabilized within the progress of sampling which is shown by the decreasing violation of the continuous constraint. Besides this stabilization, also the attractors are incorporated with constant accuracy. The sampling of the continuous constraint on the hypercube does not affect the attractors inside the hypercube. This shows that the constrained learning approach is sufficient to learn redundant kinematics of the Puma560 stably and accurate at the same time.

5.6 Conclusive Remarks

The presented approach shows that the ELM is particularly well suited to include prior knowledge into the learned function by means of adding linear constraints, which are functions of arbitrary derivatives and linear in the read-out weights of the considered network (see Sect. 5.3). Generalization of the constraint towards the continuous region is achieved by successively sampling the discrete constraints at positions where the continuous constraint is violated (see Alg. 5.2). Where no direct data are available, constraints quasi substitute actual data and can be generated artificially from prior knowledge. This is the key for faithful interpolation between sparse data points. However, quadratic optimization can guarantee the fulfillment of such constraints after learning in the discrete samples, whereas the step towards continuous constraints needs additional effort. The ELM architecture motivates an efficient verification algorithm, which is based on a worst-case approximation of the Taylor expansion of the learned function and presented in Sect. 5.4. To derive the respective bounds on the remainder of the Taylor approximation is only applicable because derivatives and maxima can be analytically determined due to the special form of the ELM - the random and untrained projection of the inputs into the networks state space.

It is shown that this approach provides all the tools to combine the full representational power of the ELM with guarantees on the performance of the learned function, which are crucial in many engineering and automatic control applications. The flexibility to define constraints locally and selectively in continuous regions and the computational efficiency due to successively applied quadratic programming scheme distinguishes the presented ELM approach from other schemes that for

instance guarantee monotonicity by means of a-priori model selection. The generality of the proposed approach to incorporate prior knowledge is one of its major advantages. This chapter shows, that besides simple cases of prior knowledge such as partial monotonicity, convexity, or bounds on the output, also complex types are expressible such as stability of the to-be-approximated dynamics as for the illustrative Puma560 example.

This chapter shows several illustrative examples where data-driven learning is applied with additional prior knowledge about the task. Several "rules of thumb" can be deduced from these experiments. First, the impact of prior knowledge is higher if the data set is *noisy* and/or *sparse*. This is clear because less and unreliable data means that more prior knowledge is required. In fact, the networks profit mostly from continuous constraints in the regions where only few data samples are present. Second, the strength of the Tikhonov regularization is directly linked to the constraint incorporation. The experiments show that the number of needed constraint samples can be reduced by stronger regularization, which, in return, means that too weak regularization leads to a bigger demand of constraints due to the possible overfitting of noisey data samples. The regularization parameter thus appears as an important control variable to cope with the curse of dimensionality. In summary, generalization of the discrete constraints to continuous regions becomes easier with lower model complexity of the network. Finally, continuous constraints present in one part of the workspace does not affect the model complexity of the network in other parts of the workspace too much. This is demonstrated in Sect. 5.5.3, where the dynamical system representing the inverse kinematics of the Puma560 robot is stable (due to the continuous constraint) close to the joint limits while sustaining the four redundant solutions in between the joint limits.

Obviously, the iterative verification and sampling algorithm becomes expensive with increasing input dimensionality of the learned function and is not practical in very high dimensions. However, in many applications, the output is rather low-dimensional. In such cases, the proposed scheme is very efficient, optimally exploits the special architectural setting of the ELM to compute all necessary quantities, and provides guarantees on particular constraints without introducing additional architectural bias. However, also higher dimensional problems can be tackled, if the demand for constraint samples is decreased by increasing the regularization parameter.

Chapter 6

Reliable Control of the Bionic Handling Assistant

The bionic handling assistant is a novel robot platform manufactured by Festo. The morphology of the robot is inspired by elephant-trunks and actuated by a pneumatic control. This chapter shows how the control of the bionic handling assistant is accelerated by means of learning.

The remainder of this chapter is organized as follows. The first section of this chapter, Sect. 6.1, introduces the bionic handling assistant as a robotic platform where reliable learning is essential. Sect. 6.2 states the architecture of the low-level control of the BHA. The data set used for investigation of the proposed ELM approach is described in Sect. 6.3. The methods used for comparison are stated in Sect. 6.4. The experimental results are given in Sect. 6.5. Sect. 6.6 demonstrates the advantages of the learned model when applied in the control loop of the bionic handling assistant in a reactive robotics scenario. The conclusive remarks on this chapter are given in Sect. 6.7. Part of the experimental results concerning the bionic handling assistant are taken from [16].

6.1 Controlling the Bionic Handling Assistant

The bionic handling assistant (BHA) [170, 171] is a pneumatically actuated, award-winning [172] robot platform [173, 174] manufactured by Festo and inspired by elephant-trunks. The actuation principle with several segments of continuous parallel components has gathered increasing interest in robotics research over the last decade. It belongs to a new class of soft and lightweight robots with pneumatic chambers which allows for a safe physical interaction between robots and humans. A picture of the BHA is shown in Fig. 6.1 (left), the structure of the BHA separated into three segments is depicted in Fig. 6.1 (right).

The downside of the biologically inspired design of the BHA is that hardly any analytic models are available for control, which qualifies learning as an essential tool for its application. The actuation itself operates with pneumatics, which is not sufficient for a reliable positioning of the robot: air pressure only describes a force, and friction with the addition of physical hysteresis effects causes different outcomes for the same pressure. In order to avoid this problem, the BHA has cable

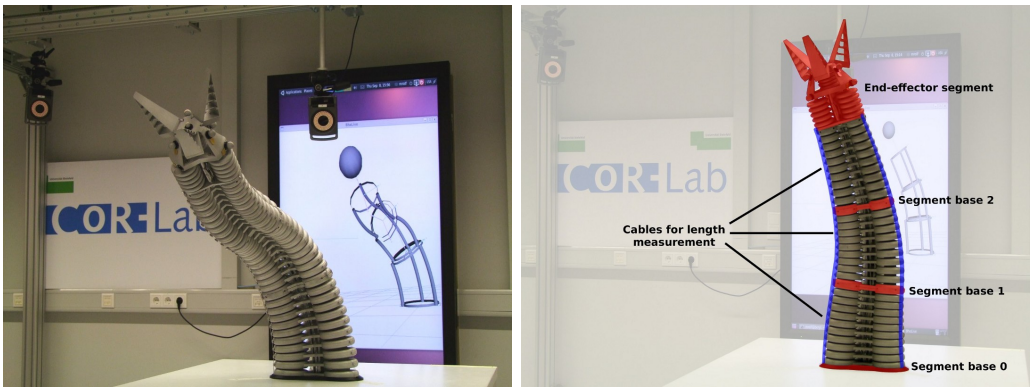

Fig. 6.1: The bionic handling assistant (BHA) (left). The segments and respective length sensors of the BHA (right).

potentiometers (see Fig. 6.1 (right)) that allow to sense the outer length of the actuators providing geometric information about the robot shape. Controlling these lengths can, in principle, be done with standard schemes like proportional integral derivative control (PID). The fundamental problem is that these approaches rely on quick and reliable feedback from the robot, while the BHA only provides very delayed and noisy feedback due to its pneumatic actuation. Consequently, the control would need to be applied with very low gains, which corresponds to only slow movements. At this point, learning can largely help to leverage the opportunities provided by the BHA. The control of the actuator lengths is improved by learning the inverse mapping from some desired actuator length to the pressure necessary to reach it in a mechanical equilibrium. This model thus allows a direct estimation of a reasonable control signal that can be applied on the robot very aggressively, without waiting for delayed feedback.

However, learning the inverse mapping for length control from scratch is difficult. On the one hand, data sampling is very expensive and must be done with great efficiency. For each ground truth point of training data, some pressure needs to be applied on the real robot. This pressure must be active until the physical deformations of the robot have reached a mechanical equilibrium, which can take up to 20 seconds for a single data point. On the other hand, the resulting samples are very noisy due to strong physical hysteresis effects induced by the soft materials applied. During the physical equilibration process, the visco-elastic properties of the elastic material result in slightly different equilibrium lengths every time. This process can be seen as highly inhomogeneous noise and requires multiple repetitions of the same experiment to obtain a reasonable basis for data driven learning. Exhaustive and precise sampling of the nine-dimensional input space of length values is impossible and a feasible machine learning algorithm has to cope with such a sparse and noisy data set.

It is nevertheless possible to derive continuous constraints from prior knowledge about the physical plant applicable in the learning process. In fact, it is known in advance that the ground-truth behavior per axis is strictly monotonous, since higher pressure in one actuator physically leads to an extension of the actuator itself. Formulated as a constraint, it is necessary for the learner to reflect this behavior in order to be applicable in a closed control loop without leading to an amplification of errors. This is particularly important for those parts in the set of inputs where only few samples are provided.

It is shown in this chapter, that the control of the BHA can be accelerated significantly by application of the proposed ELM approach. Learning is only feasible because continuous constraints derived from prior knowledge about the actual robot are used to guide the learning process. The control of the BHA is therefore a prototypical engineering application, where learning subject to continuous constraints is the method of choice, because no analytic model is available.

6.2 Low-Level Control of the BHA

Controlling the lengths can, in principle, be done with standard schemes like PID control. But since the BHA cannot provide a quick and reliable feedback due to the visco-elastic mechanics, the control would need to be applied with very low gains, corresponding to slow movements. A feed-forward controller is added in

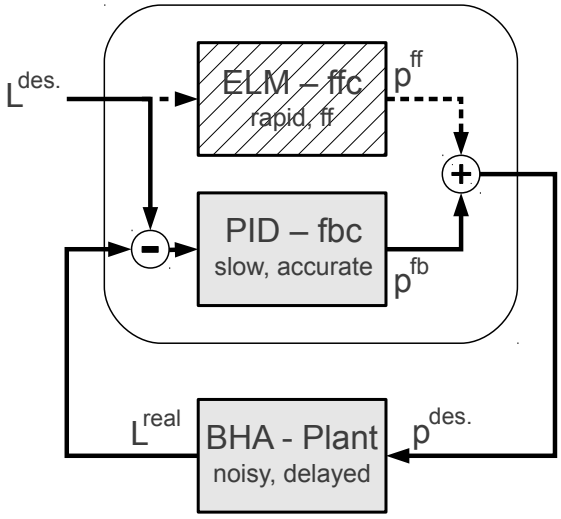

Fig. 6.2: The control loop: combination of the learned feed-forward and feedback controller for the BHA.

parallel to the feedback signal of the PID control. The feed-forward controller is supposed to represent the direct relation between length and pressures in a mechanical equilibrium and thus serves as a basis for the control of the BHA. The

PID controller is reduced to balance the small error between feed-forward control and actual output and can then be applied with low gains without losing the fast response of the combined controller.

Fig. 6.2 visualizes the control loop of the BHA schematically. The image shows the BHA plant with its noisy and delayed feedback, the PID as a feedback controller that works accurate but can only be applied with low gains, and the ELM as inverse model used as feed-forward controller. The BHA is provided with pressure values and replies with sensor values obtained by the cable potentiometers measuring the expansion of the respective pressure chamber. The PID controller is provided with the difference of the desired length values and the length sensor values while the ELM is fed by the desired length values. This ensures that the PID controller acts in a small range correcting the small errors of the ELM model in the feedback loop. The outputs of ELM and PID controller are added together and used to control the BHA. The following sections show the construction of the data set for learning the feed-forward controller by means of the ELM approach with additional use of continuous constraints.

6.3 Learning the BHA Data Set

The BHA data set contains the relation between geometric length and pressure in a mechanical equilibrium. The recorded data units are milli-bar for pressure and meters for length. The goal is to learn a partially monotonous model

$$\hat{\mathbf{p}}(\mathbf{l}) : \mathbb{R}^3 \rightarrow \mathbb{R}^3 : \forall \text{segments} \ , \tag{6.1}$$

which approximates the mapping from desired lengths $\mathbf{l}$ (meter) of the three actuators for the respective segment to the three necessary pressures $\mathbf{p}$ (milli-bar) in the mechanical equilibrium. Further, the observation that the entire mapping from length sensor values to chamber pressures can be separated into three independent three-dimensional problems for the different segments (see Fig. 6.1 (right)) is used in order to simplify the learning scenario. For each segment, the pressure space is explored by applying pressures between minimum and maximum value in five equidistant steps. This results in a pressure grid comprising $5 \times 5 \times 5 = 125$ samples. For each pressure, the resulting combination of three lengths was recorded after a waiting phase of 20 seconds in order to reach the mechanical equilibrium. In order to deal with the inherent variation due to the visco-elastic material, this process is repeated five times with different traversal orderings, such that 625 samples per segment are available for learning. The minimum and maximum pressures, and the resulting length ranges are collected in Tab. 6.1.

The grid for the applied pressures of segment one is illustrated in Fig. 6.3 (left). The corresponding length values recorded on the robot are shown in Fig. 6.3 (right). The data are clearly non-linear, with strong interactions of components and has huge gaps in the middle part of the target data, for which generalization is critical. All this makes the task evidently difficult to learn.

Seg.	Max. Pres.	Min. Pres.	Max. Len.	Min. Len.	#Samples	#Trials
1	800 mbar	0 mbar	0.32 m	0.16 m	625	5
2	1000 mbar	0 mbar	0.33 m	0.15 m	625	5
3	1200 mbar	0 mbar	0.29 m	0.16 m	625	5

Tab. 6.1: Properties of the BHA data set for the different segments.

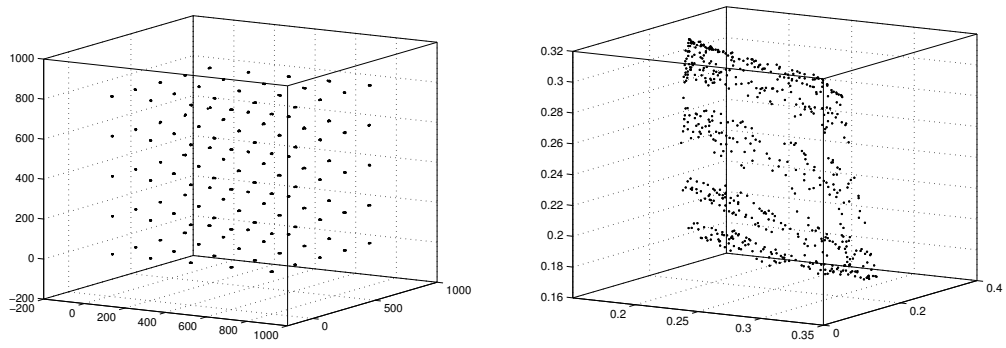

Fig. 6.3: Data set for segment 1. Pressure grid with five samples per dimension (left) and the corresponding length values (right). The highly non-linear relation between lengths and pressures leads to big gaps in the input space of the data.

Error Measure. Let $\mathcal{D} = (\mathbf{p}^i, \mathbf{l}^i)_j : i = 1 \dots N_{\text{tr}}, j = 1 \dots 5$ be the recorded data set, for segment j, learning with $N_{\text{tr}} = 125$ training samples. An appropriate error measure is to compute the per-axis average-deviation from the measured ground truth value:

$$E = \frac{1}{N_{\text{tr}}} \sum_i \frac{1}{D} \sum_d \| p_d^i - \hat{p}_d(\mathbf{l}^i) \| \ , \tag{6.2}$$

where N_{tr} is the number of samples in the evaluated data set and $D = 3$ is the input and output dimensionality.

Continuous Constraints (Partial Monotonicity). The ground-truth behavior is supposed to be strictly monotonous per axis for each segment. Rephrased in mathematical terms, the learner $\mathbf{p}(\mathbf{l})$ for one segment therefore needs to fulfill the following three properties:

$$C_d(\mathbf{l}) = -\frac{\partial}{\partial l_d} \hat{p}_d(\mathbf{l}) < 0 : \forall \mathbf{l} \in \Omega \ , \tag{6.3}$$

where $d = 1, 2, 3$ is the input dimension, $\mathbf{l}$ is an length input for the respective segment, and $\Omega = [0.16, 0.32] \times [0.15, 0.33] \times [0.16, 0.29]$ is the three-dimensional input space defined by the minimal and maximal length of the segment (see Tab. 6.1).

6.4 Models for the BHA Data Set

This section provides detailed information about the learning of the BHA data set and two additional models for comparison on the BHA data set.

6.4.1 Linear model

The linear fit $\mathbf{p}(\mathbf{l}) = M \cdot \mathbf{l} + \mathbf{o}$ with regression matrix M and offset $\mathbf{o}$ is applied to the BHA data. The parameters are obtained by regression of the data and monotony is checked by looking for positivity of the diagonal elements of regression matrix M. Partial monotonicity is not enforced by learning but checked. However, the continuous constraints are typically satisfied by the linear model due to the low complexity of the model.

6.4.2 Partially monotonic MLP

The partially monotonic multi-layer perceptron (PMMLP) model is a special case of the model in [132, 175] where only one hidden-layer is used. Partial monotonicity is implemented by initializing certain input weights positive. In order to implement the monotonic behavior, the respective weights in the output matrix are restricted to be positive. In total, $R = 300$ Neurons are used for the hidden-layer in total. An MLP is known as an universal approximator of monotonous functions, but the constraints defined for monotonicity reduce the number of degrees of freedom for the weights. The capability to approximate arbitrary, partially (but also globally) monotonic functions was nevertheless proven in [132].

6.4.3 ELM for the BHA data set

The goal for each segment is to learn an ELM model $\mathbf{p}(\mathbf{l}) : \mathbb{R}^{I=3} \rightarrow \mathbb{R}^{O=3}$ that maps the desired length $\mathbf{l}$ (meter) of the three actuators to the three necessary pressures (milli-bar) $\mathbf{p}$ in the mechanical equilibrium with additional use of continuous constraints:

$$\hat{p}_i(\mathbf{l}) = \sum_{j=1}^{R} W_{ij}^{\mathbf{out}} f(a_j \sum_{k=1}^{I} W_{jk}^{\mathbf{inp}} l_k + b_j) \ , \tag{6.4}$$

where i is the index for the respective pressure chamber. The ELMs used in the experiments have $R = 300$ hidden-layer neurons and are optimized by BIP with $\mu_{\text{BIP}} = 0.2$. The regularization parameters $\alpha = 10^{-5}$, 10^{-3}, and 10^{-2} are obtained by line search for the segments 1,2, and 3 respectively. The input space of the ELM where the constraint is implemented is defined by the morphology of the BHA and its physical restrictions stated in Tab. 6.1. $N_{\text{re}} = 10^6$ samples are used for rejection sampling. Additionally, a lower bound of $c = 100$ mbar/meter (defined with some tolerance $\varepsilon = 100$ mbar/meter) is implemented. Only networks where the algorithm terminates successfully were taken into account in the experiments.

6.5 Experimental Results on the BHA Data Set

The performance of the three different constrained models is evaluated on the BHA data set. The linear model (LM), a partially monotonic multi layer perceptron (PMMLP) [132] and the ELM model with additional use of prior knowledge are tested.

The results are obtained by cross-validation over the five trials measured by the error function in Eq. (6.2). For each fold, four trials are used for training and one trial is used for testing the generalization ability of the models. Afterwards, the errors are averaged over the five folds. 100 different network initializations are tested and the training and test errors of the best performing models are stated in Tab. 6.2. The mapping ability of the LM is too poor to capture enough structure

Seg.	**LM**	**PMMLP**	**ELM**
1	50.1/54.0	39.84/46.24	**30.77/39.76**
2	64.5/69.0	47.69/56.19	**36.29/49.77**
3	60.9/68.2	50.03/60.40	**43.80/58.23**

Tab. 6.2: Best training/test errors for the different models out of 100 initializations.

encoded in the BHA data - the errors are large. The PMMLP performs better than the LM since it induces non-linearity. Unfortunately, the performance of the PMMLP is depending on the experience of the tuning expert. Especially, the tuning of the input scaling remains difficult. The ELM - in contrast - performs significantly better although it was provided with the same number of hidden-layer neurons.

In order to show the quality of the produced mapping by the PMMLP and ELM, also the mean and standard deviation over the different initializations are stated in Tab. 6.3. In order to make the results comparable, both networks are provided with the same number of neurons. The mean performance of the ELM

Seg.	**PMMLP**	**ELM**
1	43.60±2.09/49.96±1.91	**31.01±0.11/39.92±0.10**
2	51.04±1.93/59.39±1.86	**36.61±0.15/50.13±0.14**
3	55.31±3.20/65.78±3.05	**43.89±0.06/58.40±0.09**

Tab. 6.3: Mean training errors ± standard deviation of the training errors / mean test errors ± standard deviation of the test errors for 100 initializations.

is close to the best performance (compare results summarized in Tab. 6.2 and Tab. 6.3). The standard deviation of the errors for the ELM is very low in comparison to the standard deviation for the PMMLP. The main reason for this is the

use of BIP (see Chap. 3) producing a task-specific representation in the hidden-layer irrespective of the random initialization and ridge regression additionally regularizing the read-out layer.

The experiments show that the ELM is outperforming the other models significantly. One essential reason is that the two competing models (LM, PMMLP) fulfill the constraint globally by means of model selection, although only a dimension-wise monotony is required. This too conservative model selection of LM and PMMLP reduces their approximation capability too much. There is no simple way to implement the desired partial constraint through a model selection mechanism, which underscores the flexibility of the presented approach. The effort of quadratic optimization and verification of the learned ELM function pays off, because the local implementation of the continuous constraint (if enforced only in the previously defined region Ω of the input space) leaves more degrees of freedom in the model for approximation of the actual non-linear mapping.

6.6 Experimental Results for Closed Loop Application

Three experiments to test the influence of the ELM model onto the performance of the actual closed loop control of the BHA were conducted. The results of the experiments are summarized in Tab. 6.4. In the first experiment, the value of each

ELM Model	**Jump [s]**	**Fall [s]**	**Trajectory [s]**
deactivated	13.16±0.61	5.69±0.08	38.63±0.59
activated	**3.37±0.18**	**2.66±0.04**	**6.23±0.03**

Tab. 6.4: Speed tests of the control loop with and without the application of the ELM as feed-forward control.

pressure chamber is first set to 0.19 m and then jumps to 0.24 m. A posture is considered as reached until the difference between real and desired length of each segment is less the 1 cm. The time until the posture is reached is recorded. The values are averaged over ten repetitions. Tab. 6.4 (Jump) shows that the posture is reached almost four times faster in the cases where the ELM is activated. A visualization of this jump process for one chamber in segment one is presented in Fig. 6.4 (left). The plot clearly shows that the robot is able to follow the desired length with a high precision, while the simple PID control reacts idle caused by the low gains.

In the second experiment, the value of each pressure chamber is first set to 0.24 m and then falls to 0.19 m. It is important to investigate this separately to the jump experiment because the physics of this process is different: the chambers has to be deflated instead of pressurized. Tab. 6.4 (Fall) shows the results of the experiment. The activation of the ELM model increases the performance of the

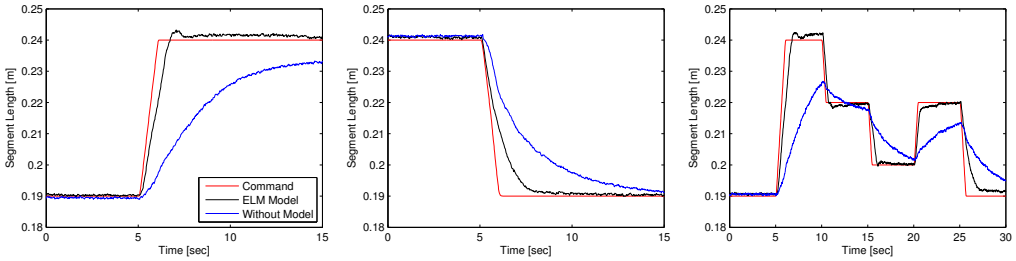

Fig. 6.4: Illustration of the three experiments conducted with and without ELM model. The reaction of the BHA on a jump (left), on a fall of the pressure (center), and a trajectory (right).

robot from 5.69 to 2.66 seconds which is still a significant enhancement. Fig. 6.4 (center) visualizes the process for the fall experiment. Also the deflation process is accelerated with the application of the inverse model. The posture is reached with a precision of one percent already after two and a half seconds.

The last experiment comprises five movements merged to a trajectory. The next posture is submitted after the previous posture is reached. The time until the last posture is reached is recorded. Tab. 6.4 (Trajectory) shows that the activation of the ELM leads to a significant speed-up of the movement trajectory. The delays for the simple PID controller aggregate such that this strategy takes more than half a minute to process this trajectory. The strategy with activated ELM takes around six seconds. Fig. 6.4 (right) shows the processing of the trajectory where each posture lasts for a time of five seconds. The BHA is able to closely follow the trajectory with activated ELM model. Without ELM model, the BHA reactes slowly and is highly over-challenged with the fast posture changes.

6.7 Conclusive Remarks

The bionic handling assistant (BHA) represents a robotic platform where learning is indispensable for a fast and accurate control. Manual modeling remains difficult due to several intrinsic properties of the underlying mapping. The main reason for these difficulties is the pneumatic actuation which results in complex friction effects and a strongly delayed feedback. However, learning has to cope with the downsides of the pneumatic design: data sampling is very expensive and the resulting samples are very noisy due to the soft materials applied. But it is nevertheless possible to derive continuous constraints from the physical plant applicable in the learning process which makes learning only then feasible. Without a learned low-level controller, appropriate human-robot interaction would not be imaginable. The results are very encouraging and the learned controller was actually used as basis for a recently developed higher process learning scheme inspired by infant-learning called

"goal-babbling" [176, 177, 178] to obtain an inverse mapping through exploration on the BHA robot ([178], video [179]). The learned controller of the BHA is also a demonstrator for self-organization and cognitive learning for intelligent technical systems in the "it's owl" leading-edge cluster[1].

This example shows that the ELM is particularly well suited to include prior knowledge into the learned function by means of adding constraints which are linear functions of arbitrary derivatives. Quadratic optimization can guarantee fulfillment of such constraints after learning in discrete points, whereas the step towards continuous constraints is achieved through an efficient verification algorithm, which is based on a worst-case approximation of the Taylor expansion of the learned function. This allows enhanced reliability in the sense that certain physical properties are satisfied. It is furthermore shown that unsupervised learning by means of batch intrinsic plasticity leads to a higher robustness of the learning results. The standard deviation of the experimental results is more than one order of magnitude lower for the models trained with unsupervised optimization through BIP and continuous constraints in comparison to the results for the PMMLP networks. This guarantees good and trustworthy learning results and makes a validation set unnecessary. It is also clear that the BHA is prone to non-stationary behavior due to the visco-elastic properties of the used material. The physical equilibration process therefore results in different equilibrium lengths with respect to the history of the arm because of hysteresis effects. It is undoubted that systematic changes in the underlying mapping appear in long term scenarios. Also rapid changes are possible such as a defect pressure chamber because the BHA is susceptible to damages in the exterior hull. In this cases, no target data is available and strongly motivates unsupervised compensation of such changes. Online training by IP with natural gradient as introduced in Chap. 4 can be used as a basis for compensation.

This approach maintains the universal approximation capability of the ELM approach and at the same time achieves guarantees on the performance of the learned function, which are crucial in this engineering and automatic control application. The flexibility of the implementation of continuous constraints which defines constraints locally and selectively in continuous regions allows good generalization on the collected BHA data. Additional experimental results show that the learned controller accelerates the BHA significantly in comparison to standard PID control without the danger of error amplification in the closed control loop. This ensures a save and fast application of the BHA, which is an indispensable prerequisite for proper human-robot interaction.

[1] See www.its-owl.de or www.cor-lab.de for further details.

Chapter 7

Stable Estimation of Dynamical Systems and Reliability

The data-driven approximation of vector fields that encode dynamical systems is a persistently hard task in machine learning. When data is sparse and given in form of velocities derived from few trajectories only, there are state-space regions where no information on the vector field and its induced dynamics are available. Reliable generalization towards such regions is meaningful only if strong biases are introduced, for instance assumptions on global stability properties of the to-be-learned dynamics. This chapter therefore introduces a novel learning scheme based on the constrained ELM approach introduced in this thesis that represents dynamical systems by means of vector fields, where asymptotic stability is explicitly enforced through utilizing techniques from Lyapunov's stability theory. In particular, the learning of the vector field is constrained through a Lyapunov function candidate, which in turn is first adjusted towards the training data. The significance of optimized Lyapunov function candidates is pointed out and the approach is analyzed in a scenario where trajectories are learned and generalized from human handwriting motions.

The remainder of the chapter is organized as follows. At first, Sect. 7.1 describes the role of dynamical system estimation with additional focus on stability for the reliable generation of complex movements for robots and mentions related work. Then, Sect. 7.2 introduces neurally imprinted vector fields as an approach to learn dynamical systems in a data-driven manner based on extreme learning machines. Sect. 7.3 explains the relevance of Lyapunov function candidates for the stability and accuracy of the resulting estimates. The obtained experimental results are summarized in Sect. 7.4, which contains three different experiments also comprising the humanoid robot iCub. Finally, Sect. 7.5 gives conclusive remarks on this chapter.

The concept of neurally imprinted stable vector field was introduced in [18] and further developed in [20]. The idea of highly flexible Lyapunov function candidates through learning with continuous constraints was proposed in [19]. Most of the content of this chapter is based on this publications.

7.1 Stable Estimation of Dynamical Systems and Reliable Movement Generation

The approximation of vector fields from sparse data, that, e.g., encode quantitative flow visualization [180], optical flow in computer vision [181], or force fields in motor control [182], is an important but also challenging task for learning algorithms. Such vector fields represent non-linear dynamical systems which were recently also applied in cognitive robotics and appear as one of the most promising candidates as computational basis for exploitation of flexible motor capabilities of modern robots [183, 184] and also humanoids [185]. For instance, point-to-point movements modeled by autonomous dynamical systems can provide a library of basic building blocks called movement primitives [186] which are very successfully applied to generate movements in a variety of manipulation tasks [187, 188]. However, in such scenarios, training data typically consist of only few trajectories and thus leave many regions in the state space with no information of the desired vector field. Generalization towards such regions is particularly challenging because small errors in the approximation of the vector field can get amplified during numerical integration and can lead to divergence of the dynamical system.

Learning is obviously an appropriate means to enable adaptive behavior, but the learning of controllers is valid only if certain demands on reliability can be guaranteed. Recently, several studies emphasized that stability plays an important role for the reliability of such tasks [189]. A widely known approach to tackle this issue is called dynamic movement primitives (DMP) [190] which is a successful technique to generate reaching motions towards a target attractor with dynamical systems. It provides an accurate non-linear estimate of a given trajectory robust to spatial perturbations while ensuring global stability at the target attractor. The stability is enforced through a stable linear dynamical system which suppresses a non-linear perturbation at the end of the motion. The smooth switch from non-linear to linear dynamics is controlled by a phase variable. The phase variable can be seen as external stabilizer which in return distorts the temporal pattern of the dynamics. This leads to the inherent inability of DMP to generalize well outside the demonstrated trajectory [191].

This fact directly motivates methods which are capable of generalizing to unseen areas. Such methods are time-independent and thus preserve the spatio-temporal pattern. They became of special interest by focusing on the "what to imitate" problem [192, 193]. However, the incorporation of stability into non-linear dynamical systems is inherently difficult due to their high complexity, in particular, if systems are supposed to approximate given data sets. On the one hand, an integration through conservative stability constraints often leads to a poor reproduction performance. On the other hand, a too strong focus on the accurate reproduction of the data lead to a weak robustness to perturbation which might end in divergence. The trade-off between stability and performance is often resolved for the benefit of stability in return for losing accuracy. This resolution is

undesired if the complex parts of underlying dynamics are of major interest which is the case for humanoid robotics. How to imprint stability, while simultaneously sustaining the complexity of the dynamical system is thus a key question to tackle.

Several solutions to that question have been developed in previous years. The common basis of these approaches is Lyapunov's theory which is a powerful tool to analyze the stability of such systems. One statement is that asymptotic stability of a fixpoint is equivalent to the existence of a Lyapunov function. However, most of the approaches for estimation of dynamical systems only deal with simple and data-independent Lyapunov functions which makes the finding of a satisfactory solution to the trade-off difficult. Artstein and Sontag [194, 195] generalized Lyapunov's theory of stability by defining conditions on a so-called control Lyapunov function, which stabilizes non-linear dynamical systems through online corrections during runtime. The construction of a control Lyapunov function, which guarantees stability without interfering with the data, was so far only solved for special cases and remains difficult [196].

An appealing approach that aims at ensuring robustness to temporal perturbations by learning vector fields from demonstrations is the stable estimator of dynamical systems (SEDS) [193]. It is based on a mixture of Gaussian functions and respects correlation across several dimensions. It is shown, that SEDS is globally asymptotically stable but restricted to approximate contractive dynamics corresponding to a quadratic Lyapunov function [193]. The extension of SEDS called SEDS-II was very recently published in [197]. It is grounded on the idea of online corrections by means of control Lyapunov functions and implements less conservative stability conditions compared to SEDS. However, the learning of the dynamical systems via SEDS-II is not informed by stability assumptions, where stability is only achieved by online perturbations of the dynamics triggered by a violation of the control Lyapunov function. Hence, SEDS-II cannot guarantee direct constructive stability. In [198], Reinhart et al. applied a neural network approach to generate movements for the humanoid robot iCub [199]. The performance and the stability are addressed by two separately trained but superimposed networks. The first network approximates the data, while the second network addresses stability by learning a velocity field, which implements a contraction towards the desired movement trajectory. However, also this approach cannot directly guarantee the stability of the motion. In this cases, the trade-off between the globally enforced stability constraint and the data approximation prevents an accurate learning of complex movements. Yet, SEDS nor superposition approaches can resolve this.

The application of standard candidates, i.e. quadratic Lyapunov candidates as in Fig. 7.1 (left) or matrix parametrized candidates visualized in Fig. 7.1 (center) is not satisfactory when the task demands an appropriate complexity. In theory, much more complex functions are possible and also desired, see Fig. 7.1 (right), but there is no constructive or analytic way to derive such a candidate function directly.

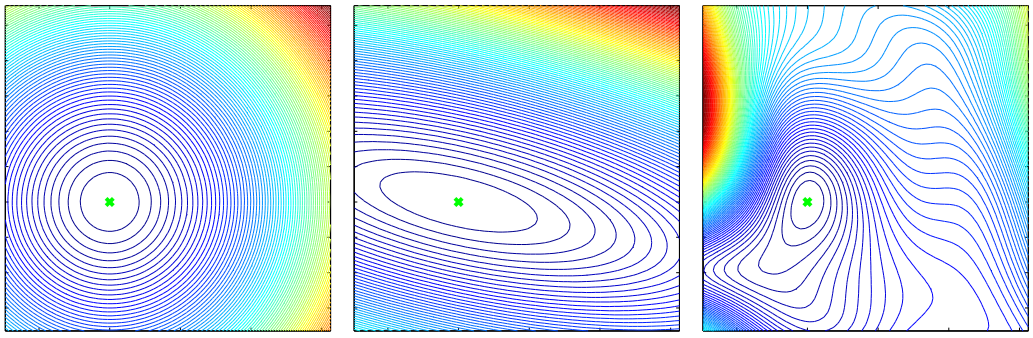

Fig. 7.1: Level sets of three different Lyapunov function candidates $L = (\mathbf{x} - \mathbf{x}^*)^2$, $L = (\mathbf{x} - \mathbf{x}^*)^T P (\mathbf{x} - \mathbf{x}^*)$, and $L =?$, surrounding an asymptotic fixed-point attractor, but which one to apply for learning?

For this reason a method called "neurally imprinted stable vector fields" [18] to learn time independent vector fields that lead to asymptotically stable dynamics is introduced. The method also allows to learn highly flexible Lyapunov candidates from data. The necessary continuous constraint to enable successive quadratic programming is deduced from stability assumptions based on Lyapunov's stability theory. The learning itself is separated into two main steps, see Fig. 7.2 for illustration of this process for Lyapunov candidates of type: $L = (\mathbf{x} - \mathbf{x}^*)^T P (\mathbf{x} - \mathbf{x}^*)$. First, a Lyapunov candidate is constructed through parameter optimization towards the data. It is used to obtain inequalities constraints on the parameters of

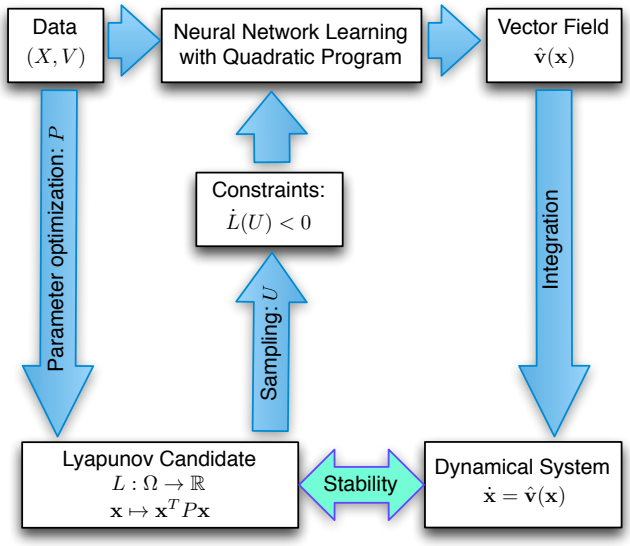

Fig. 7.2: Schematic view of the proposed approach to learn vector fields. The learning is separated into two steps: i) predefine a Lyapunov candidate L through optimization of parameters P and ii) use this function L to sample constraints U that are implemented by a successively applied quadratic program learning the data. The resulting dynamical system $\dot{\mathbf{x}} = \hat{\mathbf{v}}(\mathbf{x})$ approximates the data (X, V) and be asymptotically stable in $\mathbf{x}^*$ after learning.

the trained ELM in discrete points of the workspace. Second, the inequality constraints are implemented by a quadratic program minimizing the error between training data and output of the network. The constraints are sampled until their satisfaction in the predefined continuous workspace. Finally, the resulting vector field induces stable dynamics by construction which are linked to the previously defined Lyapunov candidate. Note, that the locality of this approach prevents a direct stability guarantee, but as the ELM allows analytical differentiation, it can be constructively and effectively proven ex-post if needed, see Chap. 5. During this chapter, different Lyapunov candidates are introduced and their impact on the estimation of dynamical systems for movement generation is investigated. In addition, experimental results that analyze the proposed method in detail is presented which show that the incorporation of the flexible learned Lyapunov candidates strongly reduces the trade-off between stability and accuracy. They also emphasize that the obtained dynamical systems are suited to allow robust and flexible movement generation for robotics.

7.2 Neurally-Imprinted Stable Vector Fields

This section briefly introduces the technique to implement asymptotic stability into extreme learning machines via sampling of constraints based on my recently published paper [18]. Stability is enforced by sampling linear constraints obtained from a Lyapunov candidate. We suggested to use quadratic functions for stabilization which are in fact relatively simple when considering complex robotic scenarios and thus raises the question: what Lyapunov candidate to apply instead?

7.2.1 Problem Statement

Point-to-point motions as driven by vector fields are considered: a state variable $\mathbf{x}(t) \in \Omega \subseteq \mathbb{R}^d$ in time $t \in \mathbb{R}$ defines a discrete motion of a robotic system, where this variable $\mathbf{x}(t)$ could be the robot's joint angles or the position of the arm's end-effector at time t:

$$\dot{\mathbf{x}} = \mathbf{v}(\mathbf{x}) \ . \tag{7.1}$$

It is assumed that $\mathbf{v} : \Omega \to \Omega$ is non-linear, continuous, and continuously differentiable with a single asymptotically stable fixed-point attractor $\mathbf{x}^* = \mathbf{v}(\mathbf{x}^*) = 0$ in Ω without loss of generality[1]. The limit of each trajectory satisfies:

$$\lim_{t \to \infty} \mathbf{x}(t) = \mathbf{x}^* : \forall \mathbf{x} \in \Omega \ . \tag{7.2}$$

The focus is the estimation of $\mathbf{v}$ as a function of $\mathbf{x}$ by using demonstrations for training and ensure its asymptotic stability at target $\mathbf{x}^*$ in Ω. The estimate is denoted by $\hat{\mathbf{v}}$ in the following.

[1] An arbitrary fixed-point attractor can be shifted to the origin of the coordinates' system by means of the diffeomorphism $\tau(\mathbf{x}) = \mathbf{x} - \mathbf{x}^*$. The transformed dynamical systems is equivalent to the original system.

7.2.2 Extreme Learning Machine for Estimating $\mathbf{v}(\mathbf{x})$

Consider an ELM for estimation of $\mathbf{v}$ comprising three layers of neurons, where the input $\mathbf{x} \in \mathbb{R}^d$ and the output $\hat{\mathbf{v}} \in \mathbb{R}^d$ have the same dimensionality d. For input $\mathbf{x}$ the output of the ith neuron is given by:

$$\hat{v}_i(\mathbf{x}) = \sum_{j=1}^{R} W_{ij}^{\mathbf{out}} f(\sum_{n=1}^{d} W_{jn}^{\mathbf{inp}} x_n + b_j) \ , \tag{7.3}$$

where $i = 1, \ldots, d$, b_j is the bias for neuron j, and f denotes the Fermi activation function applied to each neuron in the hidden-layer.

Let $D = (\mathbf{x}^i(k), \mathbf{v}^i(k)) : k = 1 \ldots N^i, i = 1 \ldots N_{\text{traj}}$ be the data set for training where N_{traj} is the number of demonstrations. N_{ds} denotes the overall number of samples in D. The evolution of motion can be computed by numerical integration of $\dot{\mathbf{x}} = \hat{\mathbf{v}}(\mathbf{x})$, where the term $\mathbf{x}(t = 0) \in \mathbb{R}^d$ denotes the starting point of the motion.

7.2.3 Stabilization through Lyapunov's Theory

Dynamical systems implemented by neural networks have been advocated as a powerful means to model robot motions [198, 169], but "the complexity of training these networks to obtain stable attractor landscapes, however, has prevented a widespread application so far" [190]. One reason for this is that learning a vector field from a few training trajectories gives only sparse information on the shape of the entire vector field. There is thus a desperate need for generalization to spatial regions where no training data reside.

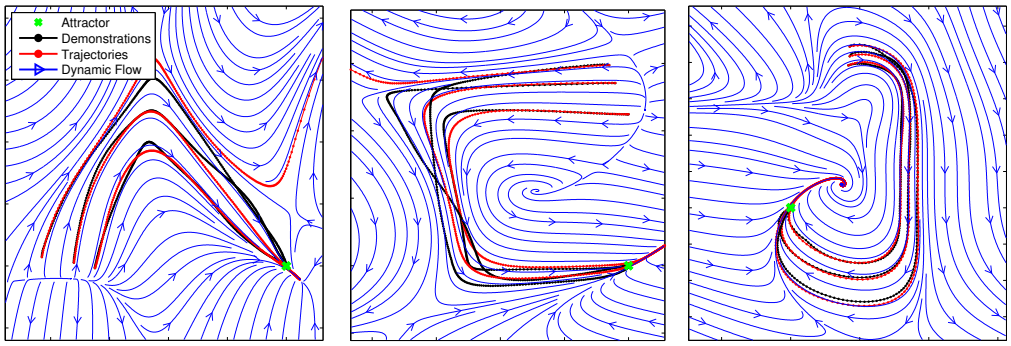

Fig. 7.3: Unstable estimation of dynamical systems through demonstrations by an ELM. Shapes are from the LASA data set: A-shape (first), sharp-C (second), and J2-shape (third).

Fig. 7.3 illustrates three examples of unstable estimation of non-linear dynam-

ical systems using ELMs. The data are from the LASA data set[2] comprising 20 handwritten motions by humans. The networks are each trained with three trajectories obtained from a human demonstrator. The first plot shows that the reproduced trajectories converge to a spurious attractor in the immediate vicinity of the target or diverge. The second plot demonstrates that the motion either converges to other spurious attractors far away from the target or completely diverge from it. The third plot show an example motion also subject to unstable behavior. This illustration shows that learning this attractor landscape is particularly hard without explicit stabilization because the training data comprises only a few training samples that encode the target.

In order to stabilize the dynamical systems estimate $\hat{\mathbf{v}}$, the conditions for asymptotic stability of arbitrary dynamical systems discovered by Lyapunov are recalled: a dynamical system is asymptotically stable at fixed-point $\mathbf{x}^* \in \Omega$ in the compact and positive invariant region $\Omega \subset \mathbb{R}^d$ if and only if there exists a continuous and continuously differentiable function $L : \Omega \to \mathbb{R}$ (called Lyapunov function) which satisfies the following conditions:

$$\begin{aligned} &(\mathbf{i})\ \ L(\mathbf{x}^*) = 0 \qquad (\mathbf{ii})\ \ L(\mathbf{x}) > 0 : \forall \mathbf{x} \in \Omega, \mathbf{x} \neq \mathbf{x}^* \\ &(\mathbf{iii})\ \ \dot{L}(\mathbf{x}^*) = 0 \qquad (\mathbf{iv})\ \ \dot{L}(\mathbf{x}) < 0 : \forall \mathbf{x} \in \Omega, \mathbf{x} \neq \mathbf{x}^* \ . \end{aligned} \tag{7.4}$$

A function L that only satisfies condition (**i**)-(**iii**) is called: *Lyapunov candidate*. Such a function can be used to obtain a learning algorithm that also satisfies condition (**iv**) w.r.t. the estimated dynamics $\hat{\mathbf{v}}$. Condition (**iv**) can be re-written by using the ELM learner defined in Eq. (7.3):

$$\begin{aligned} \dot{L}(\mathbf{x}) &= \frac{\mathrm{d}}{\mathrm{d}t} L(\mathbf{x}) = (\nabla_{\mathbf{x}} L(\mathbf{x}))^T \cdot \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = (\nabla_{\mathbf{x}} L(\mathbf{x}))^T \cdot \hat{\mathbf{v}} \\ &= \sum_{i=1}^{d} (\nabla_{\mathbf{x}} L(\mathbf{x}))_i \cdot \sum_{k=1}^{R} W_{ij}^{\mathbf{out}} \cdot f(a_j \sum_{k=1}^{d} W_{jk}^{\mathbf{inp}} x_k + b_j) < 0 \ . \end{aligned} \tag{7.5}$$

Note that $\dot{L}$ is linear in the output parameters $W^{\mathbf{out}}$ irrespective of the form of the Lyapunov function L. Eq. (7.5) defines a linear inequality constraint $\dot{L}(\mathbf{x}) < 0$ on the read-out parameters $W^{\mathbf{out}}$ for a given point $\mathbf{u} \in \Omega$, which can be implemented by the previously defined successive quadratic programming technique defined in Chap. 5. The training of the read-out weights $W^{\mathbf{out}}$ is rephrased as a quadratic program subject to constraints given by L:

$$\begin{aligned} W^{\mathbf{out}} = \arg\min_{W} (&\|W \cdot H(X) - V\|^2 + \alpha \|W\|^2) \\ &\text{subject to: } \dot{L}(U) < 0 \ , \end{aligned} \tag{7.6}$$

where the matrix V contains the corresponding target velocities of the demonstrations and $U = (\mathbf{u}(1), \ldots, \mathbf{u}(N_\mathrm{s}))$ is the matrix collecting the discrete samples obtained by sampling.

[2]The data were taken from [200] called LASA data set.

7.3 What is a good Lyapunov Candidate for Learning?

The previous section describes how a Lyapunov candidate $L : \Omega \to \mathbb{R}$, which satisfies condition (**i**), (**ii**), and (**iii**) is used to draw inequality constraints implemented by the quadratic program in Eq. (7.6). Fig. 7.4 illustrates the incorporation effect by showing the same demonstrations as in Fig. 7.3 but with additional stabilization by using the simple quadratic Lyapunov candidate $L_q = \mathbf{x}^2$. It is shown that the

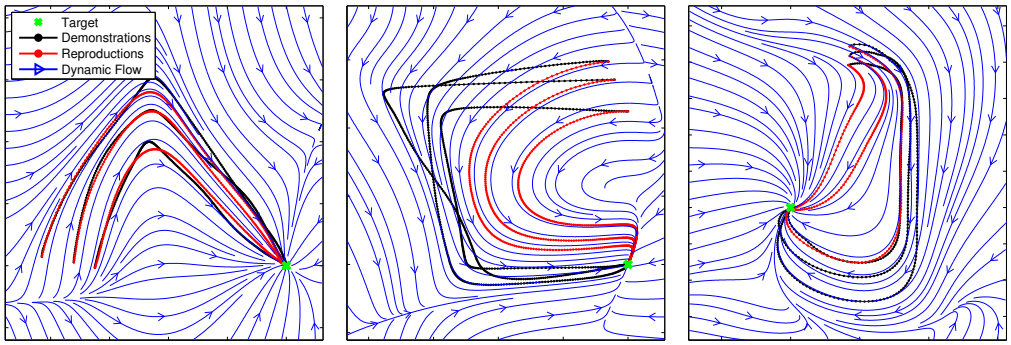

Fig. 7.4: Stable estimation of A-shape (first), sharp-C (second), and J2-shape (third) by applying L_q as Lyapunov candidate. Compare with Fig. 7.3

form of the candidate has a great impact on the resulting shape of the estimated dynamics. The data-independent candidate function L_q is sufficient to estimate the A-shape (Fig. 7.4, first) with good generalization. It is less suited to approximate the sharp-C shape (Fig. 7.4, second). The third shape also remains difficult for the learning with such a simple Lyapunov candidate.

Note that it is impossible to find an estimate for the sharp-C that approximates the data accurately while simultaneously fulfilling the quadratic constraint given by the Lyapunov candidate function L_q. Already some of the training samples $(\mathbf{x}^i(k), \mathbf{v}^i(k))$ are violated by L_q: $(\nabla L_q(\mathbf{x}^i(k)))^T \cdot \mathbf{v}^i(k) > 0$. This is in contradiction to condition (**iv**) of Lyapunov's theorem.

In order to prevent a violation of the training data D by a given Lyapunov candidate L, the measure M in Eq. (7.7) is defined. The following functional is called measure of violation:

$$M(L) = \frac{1}{N_{\text{ds}}} \sum_{i=1}^{N_{\text{traj}}} \sum_{k=1}^{N^i} \Theta \left[(\nabla L(\mathbf{x}^i(k)))^T \cdot \mathbf{v}^i(k) \right] \quad , \tag{7.7}$$

where $\Theta : \mathbb{R} \to \mathbb{R}$ denotes the ramp function[3], which guarantees that only those samples $(\mathbf{x}^i(k), \mathbf{v}^i(k))$ are counted in M where the scalar product between $\nabla L(\mathbf{x}^i(k))$ and $\mathbf{v}^i(k)$ is positive and thus violating Lyapunov's condition (**iv**).

[3]The ramp function is zero for inputs below zero and the identity function for input values above or equal to zero.

A good Lyapunov candidate does not violate the training data too much (i.e. M is small). This cannot be guaranteed for data-independent Lyapunov candidates in general. This raises the question: which Lyapunov candidate function is feasible for learning dynamical systems from complex data? The following sections answer this questions by proposing three different Lyapunov candidate functions compared in several experimental setups.

7.3.1 Quadratic Lyapunov Candidate

The most commonly used Lyapunov candidate is quadratic and given by the following function:

$$L_q = (\mathbf{x} - \mathbf{x}^*)^T (\mathbf{x} - \mathbf{x}^*) \ , \tag{7.8}$$

which is also used as a Lyapunov candidate in the experimental setup considered in this thesis. This function is data independent, quadratic and fulfills the conditions (**i**)-(**iii**). The function L_q is also a valid Lyapunov function for estimates produced by SEDS [193].

In [18], Lyapunov candidates of the following form are considered (the level sets of this Lyapunov candidate function are elliptic):

$$L_P(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}^*)^T \cdot P \cdot (\mathbf{x} - \mathbf{x}^*) \ . \tag{7.9}$$

Note that (**i**)-(**iii**) are fulfilled if P is positive definite and symmetric and that L_P equals L_q if P is equal to the identity matrix. It is defined as:

$$P = \underset{P \in \mathcal{P}}{\arg\min}\, M(L_P) \ . \tag{7.10}$$

M is defined according to the following sum over the training data which is derived from Eq. (7.7) by differentiation and injection of Eq. (7.9) as Lyapunov candidate:

$$M(L_P) = \frac{1}{N_{\text{ds}}} \sum_{i=1}^{N_{\text{traj}}} \sum_{k=1}^{N^i} \Theta \left[(\mathbf{x}^i(k))^T \cdot P \cdot \mathbf{v}^i(k) \right] \ , \tag{7.11}$$

The minimization operator in Eq. (7.10) can be formulated as a non-linear program, where sequential quadratic programming based on quasi Newton methods is used for optimization [201]. The possible matrices are restricted to be an element of $\mathcal{P} := \{P \in \mathbb{R}^{d \times d} : P^T = P,\ \lambda_i \in [\alpha, 1],\ \lambda_i \text{ is EV of } P\}$[4], where $\alpha = 0.1$ is a small and positive scalar.

7.3.2 Neural Architecture for Learning a Lyapunov Candidate

The candidate functions L_q and L_P are only able to capture a limited class of dynamics. Therefore, [19] suggested to learn a candidate function which is more flexible but in turn still feasible for implementation.

[4] EV is used as abbreviation of the mathematical term "eigenvalue".

Consider an ELM architecture which defines a scalar function $L_{\mathrm{ELM}} : \mathbb{R}^d \to \mathbb{R}$. Note, that this ELM also contains a three-layered and random projection structure with only one output neuron. The read-out matrix becomes $W^{\mathbf{out}} \in \mathbb{R}^R$. The main goal is to minimize the violation of the training data measured by Eq. (7.7) by making the negative gradient of this function follow the training data closely. Again, a quadratic program is defined:

$$\frac{1}{N_{\mathrm{ds}}} \sum_{i=1}^{N_{\mathrm{traj}}} \sum_{k=1}^{N^i} \| - \nabla L_{\mathrm{ELM}}(\mathbf{x}^i(k)) - \mathbf{v}^i(k)\|^2 \to \min_{W^{\mathbf{out}}} \ , \tag{7.12}$$

subject to the following equality and inequality constraints corresponding to Lyapunov's conditions (**i**)-(**iv**) such that L_{ELM} becomes a valid Lyapunov candidate:

$$\begin{aligned} &(\mathbf{a}) \quad L_{\mathrm{ELM}}(\mathbf{x}^*) = 0 \quad &&(\mathbf{b}) \ \ L_{\mathrm{ELM}}(\mathbf{x}) > 0 : \mathbf{x} \neq \mathbf{x}^* \\ &(\mathbf{c}) \ \ \nabla L_{\mathrm{ELM}}(\mathbf{x}^*) = 0 \quad &&(\mathbf{d}) \ \ - x^T \nabla L_{\mathrm{ELM}}(\mathbf{x}) < 0 : \mathbf{x} \neq \mathbf{x}^* \ , \end{aligned} \tag{7.13}$$

whereas the constraints (**a**) and (**c**) define equality constraints, the constraints (**b**) and (**d**) define inequality constraints which are implemented by sampling these constraints according to Alg. 5.2. The gradient of the scalar function defined by the ELM is linear in $W^{\mathbf{out}}$ and therefore suited for implementation. It is given by:

$$(\nabla L_{\mathrm{ELM}}(\mathbf{x}))_i = \sum_{j=1}^{R} W_j^{\mathbf{out}} f'(a_j \sum_{k=1}^{d} W_{jk}^{\mathbf{inp}} x_k + b_j) \cdot (a_j W_{ji}^{\mathbf{inp}}) \ , \tag{7.14}$$

where f' denotes the first derivative of the Fermi function. This technique allows the learning of a flexible Lyapunov candidate function that is optimized towards the data.

7.4 Experimental Results

This section contains the experimental results obtained for different Lyapunov function candidates. First the impact of stability is investigated in Sect. 7.4.1. In Sect. 7.4.2, different Lyapunov candidates are tested and compared on the LASA data set comprising 20 different hand writing motions by humans. Finally, the approach is evaluated in a real world imitation learning scenario involving the humanoid robot iCub in Sect. 7.4.3.

7.4.1 The Impact of Regularization and Stability Incorporation

This section contains experiments regarding the regularization of the parameters. All experiments in this section are performed with L_P as Lyapunov candidate.

Stable vs. Unstable Estimation. In order to analyze the impact of the stabilization mechanism, shapes learned from human-demonstrated movements are applied. The used data is composed of three S-like trajectories with 250 samples each and the end-point located at the origin (see Fig. 7.5). The ELM is initialized with $R = 100$ neurons in the hidden-layer. The slopes a_i are initialized with ones, biases b_i and components of $W^{\mathbf{inp}}$ are initialized randomly drawn from the uniform distribution on $[-1, 1]$. The regularization parameter is $\alpha = 10^{-5}$. The constraint samples are drawn from the uniform distribution on the set $\Omega = [-1, 2.5] \times [-1, 2.5]$, which covers the relevant region of the task space. Lyapunov functions $L_P(\mathbf{x})$ with elliptic form, where P is obtained according to Eq. (7.10) are considered. Fig. 7.5 (left) illustrates an example of unstable estimation of a non-linear dynamical system by an ELM trained without the usage of explicit stability constraints. In the

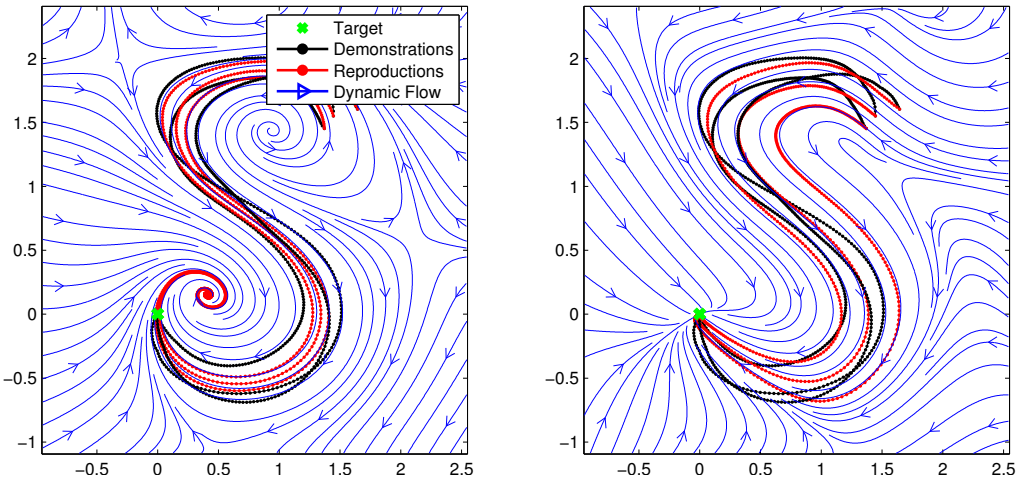

Fig. 7.5: The impact of the incorporation of asymptotic stability into the learning. Visualized dynamics of a network trained without stability constraints (left) and the same network trained with constraints for stabilization (right).

areas close to the demonstrations, the trajectories converge to a spiral attractor close to the target. In other regions of the space, they either converge to spurious attractors or diverge. In contrast, Fig. 7.5 (right) shows the same network setup but trained with the stabilization method. The generated trajectories converge to the target, because the learning process enforces asymptotic stability of the fixed point attractor in $\mathbf{x}^*$. This ensures that the target is reached when starting from any point in the workspace. In the following, an evaluation of the new stability mechanism is shown, using two performance measures. The first measure is the mean square error $E_{\text{tr}} = \frac{1}{N_{\text{tr}}} \sum_k \|\mathbf{v}(k) - \hat{\mathbf{v}}(\mathbf{x}(k))\|^2$ evaluated on the training data, which quantifies the ability to approximate the training data. The second measure quantifies the stability of the dynamical system. For this measure, $N_{\text{s}} = 100$ starting points uniformly drawn from $[-1, 2.5] \times [-1, 2.5]$ are chosen. The start-

ing points are used to perform movements with $N_{\max} = 1000$ steps with step size $\Delta_t = 0.1$; the resulting states $\mathbf{x}(N_{\max})$ are recorded. If the end-point $\mathbf{x}(N_{\max})$ of the reproduced trajectory is in the vicinity of the desired attractor $\mathbf{x}^* = \mathbf{0}$, the end-point is recognized as converged (i.e. $\|\mathbf{x}(N_{\max}) - \mathbf{x}^*\| < \delta = 1$), otherwise as diverged. The distance $\text{dist} = \|\mathbf{x}(N_{\max}) - \mathbf{x}^*\|$ from the converged points to the attractor and the number of converged points S are stored. All results are averaged over $N_{\text{ni}} = 10$ different network initializations. In Tab. 7.1 the results of the experiments for networks with and without stabilization for different regularization parameters α are shown. The precision is given in the distance (dist) of the end-

	Without constraints			**With constraints**		
$\ln \alpha$	dist	S/N_{s}	E_{tr}	dist	S/N_{s}	E_{tr}
-8	.431±.067	.686±.053	.208±.0006	.049±.014	1	.263±.0036
-6	.431±.018	.813±.031	.218±.0008	.043±.016	1	.283±.0043
-4	.401±.013	.917±.053	.235±.0027	.055±.014	1	.313±.0029
-2	.382±.010	.965±.044	.304±.0028	.020±.012	1	.377±.0026

Tab. 7.1: Network learning with and without constraints compared for different regularization parameters α.

point to the desired attractor after convergence. The ratio S/N_{s} shows how many generated movements according to the numerical integration process converged to the desired target. Note that $S/N_{\text{s}} = 1$ holds for all constrained networks. The results show that the stability ratio increases with growing regularization for networks trained without constraints. This induces a trade-off between stability and accuracy for the unconstrained ELM. For the explicitly stabilized ELMs, the trade-off is resolved, because all networks converge to the target independent of the regularization. Also the target is imprinted with a higher degree of precision (cf. Tab. 7.1 dist).

Constraints: Sampling and Generalization. This paragraph investigates the generalization of the discrete constraints towards a continuous region and the effects on the stability of the estimated dynamical system. For this region, only parts of the workspace are subject to sampling. Fig. 7.6 shows the results of the experiment; please compare with Fig. 7.5 left and right. In this experiment, only the region below the black stroke is subject to constraint sampling. Three regions in this experiment are interesting: the spiral attractor in region 1, that appears in Fig. 7.5 (left), is deleted due to the sampling of constraints. The spiral repeller in Region 3 is still active since it is close to the demonstrations and not subject to sampling. Region 2 shows that the stability constraint can be generalized far beyond the border of the workspace towards regions not subject to sampling, since no data is present in this region, thanks to the strong bias induced by the

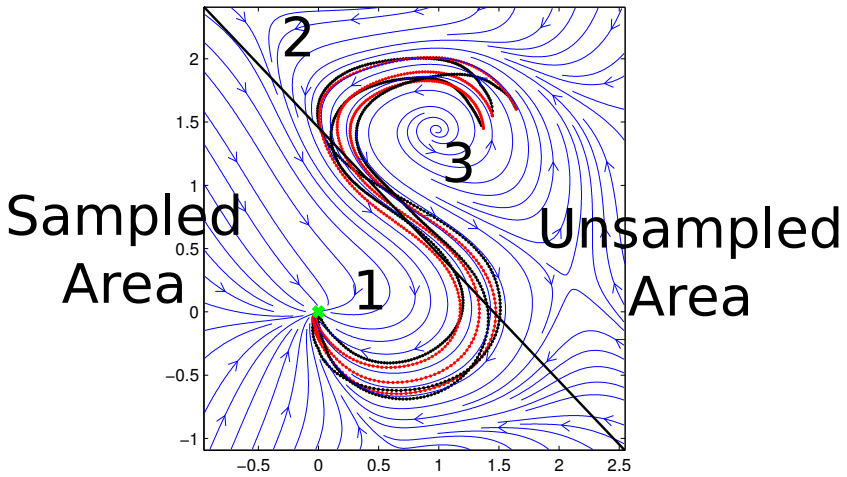

Fig. 7.6: Investigation of the S-shape where a network is trained with constraints that are only sampled in the lower half of the workspace.

stability constraints. This example shows that the locality of the approach is no disadvantage. Instead, locality adds high flexibility to the approach. This is not possible with global constraints such as SEDS [193] or the superimposed vector fields approach [198] introduce.

Regularization vs. Violation. This paragraph shows an investigation on each of the twenty motion shapes separately. The results are visualized in Fig. 7.9. The plot in the upper part of the figure shows the value for M defined in Eq. (7.7) and Eq. (7.11) for each shape of the data set. It is revealed that the simple Lyapunov candidate $L_P(\mathbf{x})$ strongly violates the training data for some of the shapes such as the G, the S, and the J2-shape. The second plot in Fig. 7.9 shows the number of samples drawn in the learning phase until implementation of asymptotic stability. This plot summarizes the results for networks with different regularization parameters trained with L_P. Two results can be deduced from this experiment. First, there is a clear correlation between difficult shapes - in the sense of M - and the number of applied samples: difficult shapes need more samples for implementation of stability. Second, networks with stronger regularization need less constraints for enforcing stability. This is advantageous, because it significantly reduces the time needed for learning. Note, that the number of samples until implementation of stability for some shapes can be reduced significantly by increasing the regularization parameter from $\alpha = 10^{-8}$ to $\alpha = 10^{-2}$.

Fig. 7.9 (third plot) shows the training error obtained in this experiment. It is observable that the training error is high for shapes which also produce a high value of violation M. This is expected because the networks are forced to implement the Lyapunov candidate. Thus, if the Lyapunov candidate violates the training data,

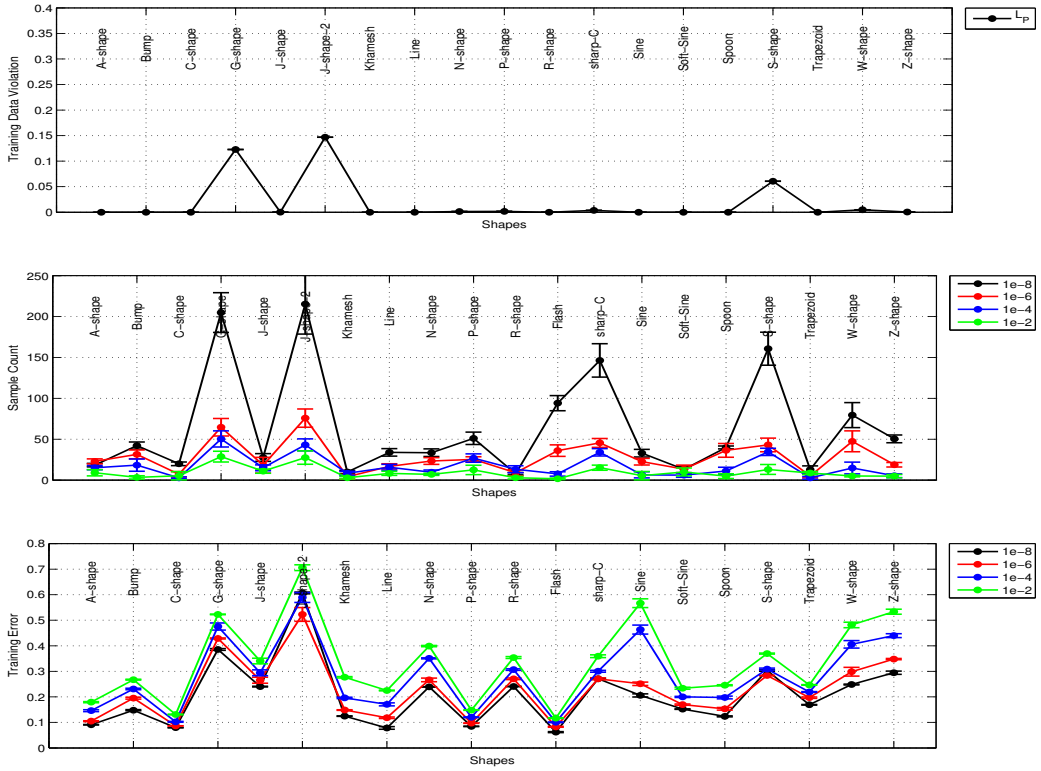

Fig. 7.7: Investigation on 20 shapes of the LASA data set. Evaluation of the violation measure M on each shape (first), the used number of samples until implementation of asymptotic stability (second), and the training errors (third).

the network dynamics will not accurately reproduce the demonstrations. It is also revealed that the training error is decreasing for lower regularization parameters α. These experiments show that the regularization parameter is still important to tune, besides that the stability of the resulting estimate is mainly influenced by the sampling scheme.

7.4.2 The Impact of Different Lyapunov Candidate

In previous sections, three different Lyapunov candidates L_q, L_P, and L_{ELM} are suggested for learning. This section, therefore, compares and discusses the resulting performance of the estimated dynamical systems when applying these candidates for learning on the LASA data set. In contrast to the previous experimental section, two different performance measures that are more reasonable to quantify the goodness of the estimate are chosen. The local accuracy of the estimate is

measured according to:

$$E_{\text{velo}} = \frac{1}{N_{\text{ds}}} \sum_{i=1}^{N_{\text{traj}}} \sum_{k=1}^{N^i} \left(r \left(1 - \frac{\mathbf{v}^i(k)^T \hat{\mathbf{v}}(\mathbf{x}^i(k))}{\|\mathbf{v}^i(k)\| \|\hat{\mathbf{v}}(\mathbf{x}^i(k))\| + \epsilon} \right)^2 + q \frac{(\mathbf{v}^i(k) - \hat{\mathbf{v}}(\mathbf{x}^i(k)))^T (\mathbf{v}^i(k) - \hat{\mathbf{v}}(\mathbf{x}^i(k)))}{\|\mathbf{v}^i(k)\| \|\mathbf{v}^i(k)\| + \epsilon} \right)^{\frac{1}{2}} , \tag{7.15}$$

which was suggested as a local measure of the velocity field in [193] and quantifies the discrepancy between the direction and magnitude of the estimated and the ground-truth velocity vectors for all training data points[5]. The accuracy of the reproductions is measured according to:

$$E_{\text{traj}} = \frac{1}{T \cdot N_{\text{ds}}} \sum_{i=1}^{N_{\text{traj}}} \sum_{k=1}^{N^i} \min_l \|\hat{\mathbf{x}}^i(k) - \mathbf{x}^i(l)\| , \tag{7.16}$$

where $\hat{\mathbf{x}}^i(\cdot)$ is the equidistantly sampled reproduction of the trajectory $\mathbf{x}^i(\cdot)$ and T denotes the mean length of the demonstrations.

The results for each shape are averaged over ten network initializations. Each network used for estimation of the dynamical system comprises $R = 100$ hidden-layer neurons, $\alpha = 10^{-8}$ as regularization parameter, and $N_C = 10^5$ pool samples for the sampling of the constraints. The networks for Lyapunov candidate learning also comprise $R = 100$ neurons in the hidden-layer and $\alpha = 10^{-8}$ as regularization parameter.

Results on Selected Shapes (A-shape and J2-shape). Two movements from the data set are used to analyze the performance of the methods in detail: A-shape and J-2-shape. This demonstrations are shown in Fig. 7.3 (left) and Fig. 7.8, respectively. The experimental results are provided in Tab. 7.2. The table contains the measure of violation M, the velocity error according to Eq. (7.15), and the trajectory error defined in Eq. (7.16). The first tabular states the results for the A-shape, the second tabular summarizes the experimental results for the J2-shape, and the overall results for the LASA data set are collected in Tab. 7.2 (last tabular). Fig. 7.8 visualizes the estimation of the J-2-shape and the respective Lyapunov candidates.

The numerical results in Tab. 7.2 show that the function used for implementation of asymptotic stability has a strong impact on the approximation ability of the networks. The A-shape (see first tabular) was accurately learned by all models. The reason is that the demonstrations do not violate the respective Lyapunov candidates to a high degree which is indicated by small value of M. The J-2-shape is one of the most difficult shapes in the LASA data set due to the

[5]Measure and values are ($r = 0.6$, $q = 0.4$) taken from [193] and $\epsilon = 10^{-6}$.

A-shape	**#Samples**	M	E_{velo}	E_{traj}
L_q	21.5 ± 3.4	0.008	$.106 \pm .0006$	$.0081 \pm .0003$
L_P	$\mathbf{18.8 \pm 2.1}$	**0.000**	$\mathbf{.096 \pm .0015}$	$\mathbf{.0053 \pm .0002}$
L_{ELM}	50.2 ± 10.6	0.000	$.103 \pm .0023$	$.0066 \pm .0004$

J-2-shape	**#Samples**	M	E_{velo}	E_{traj}
L_q	111.9 ± 23.5	0.198	$.093 \pm .0015$	$.0298 \pm .0032$
L_P	215.4 ± 36.8	0.147	$.143 \pm .0011$	$.0241 \pm .0004$
L_{ELM}	$\mathbf{77.9 \pm 7.8}$	**0.000**	$\mathbf{.069 \pm .0013}$	$\mathbf{.0079 \pm .0004}$

LASA all	**#Samples**	M	E_{velo}	E_{traj}
L_q	$\mathbf{61.2 \pm 2.4}$	0.0582	$.114 \pm .0003$	$.0218 \pm .0005$
L_P	63.2 ± 2.2	0.0180	$.110 \pm .0004$	$.0177 \pm .0003$
L_{ELM}	69.2 ± 4.8	**0.0001**	$\mathbf{.103 \pm .0006}$	$\mathbf{.0145 \pm .0004}$

Tab. 7.2: Learning results for A-shape, J-2-shape, and the entire data. The table contains the number of samples used until implementation of the continuous constraint, the violation of the training data through the Lyapunov Candidate measured by M, the velocity error E_{velo}, and the trajectory error E_{traj}.

high curvature inherent to the demonstrations of this movement shape. Therefore, the networks trained with the L_{ELM} candidates perform significantly better than the other methods, which is expected because of the high flexibility of the learned Lyapunov candidate. This is due to the high flexibility of the candidate function which smoothly suites the candidate towards the data. The third tabular in Tab. 7.2 shows the results for the whole data set. It shows that the differences in the velocity error are marginal in comparison to the values for the trajectory error. The networks trained with the quadratic Lyapunov candidate L_q perform the worst because the function cannot capture the structure in the data. The method using L_{ELM} has the lowest trajectory error values which is due to the high flexibility of the candidate function. Thus, the experiments support the hypothesis that the flexibility of the Lyapunov candidate becomes more important if the demonstrations are of high complexity.

In addition to the experiments, Fig. 7.8 shows the estimations of the J-2-shape and their respective Lyapunov candidates. The left column of the figure contains the estimation result obtained for an ELM without regards to stability. The data is accurately approximated but the target is not reached at the end of the reproduced trajectories. The plot also emphasizes that even reproductions starting in the vicinity of the demonstrations are prone to divergence. The second column illustrates the results for networks trained with respect to L_q. It is shown that

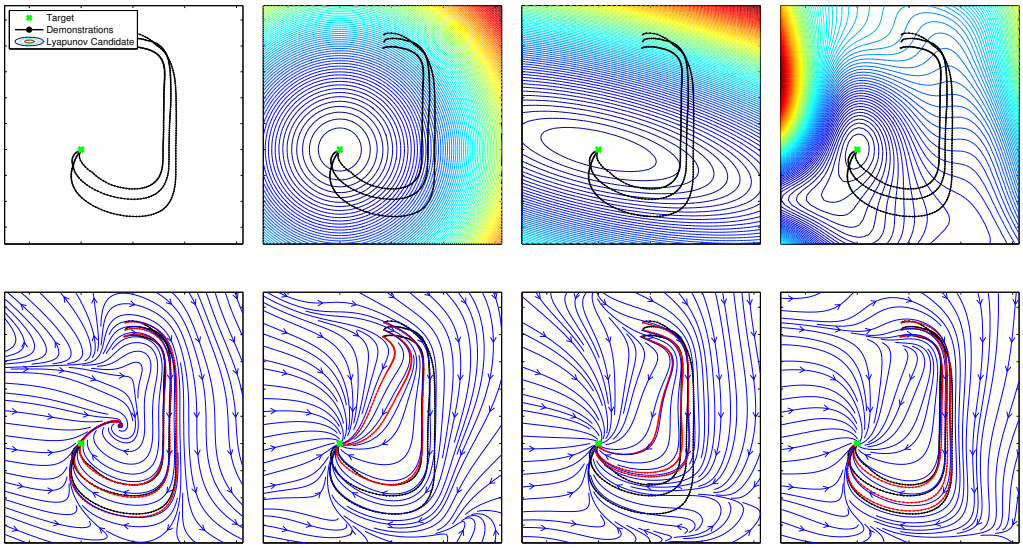

Fig. 7.8: Estimates of the J-2-shape and respective Lyapunov candidates. The J-2-shape approximated without explicit stabilization (first column), with L_q (second column), L_P (third column), and L_{ELM} as Lyapunov candidate (fourth column). The L_{ELM} Lyapunov candidate leads to good generalization results.

this Lyapunov candidate introduces a very strict form of stability, without respect to the demonstrations. The reproductions are directly converging towards the attractor. This is due to the high violation of the demonstrations by L_q close to the start of the demonstrations. Column three of Fig. 7.8 shows the results for L_P. This Lyapunov candidate is data-dependent but still too limited to capture the full structure of the J-2 demonstrations. The fourth column of the figure illustrates the performance of the networks trained by L_{ELM}. The Lyapunov candidate is strongly curved to follow the demonstrations closely (first row, fourth column). The estimate leads to very accurate reproductions with good generalization capability.

Results on the Entire Shape Set. Fig. 7.9 shows an investigation of several measures on each of the twenty shapes separately to get an overview of the impact of the Lyapunov candidate on the accuracy of the resulting estimate. The first plot of the figure shows the value for M defined in Eq. (7.7) for each shape of the data set. The quadratic and non-data dependent Lyapunov candidate function L_q is not able to capture the structure of some shapes such as the G, the J2, the sharp-C shape, etc. This violation can be relaxed by introduction of the matrix P for this candidate optimized by means of Eq. (7.10). Some of the shapes are not violated to a high degree anymore after optimization - e.g. the sharp-C shape. The L_{ELM}

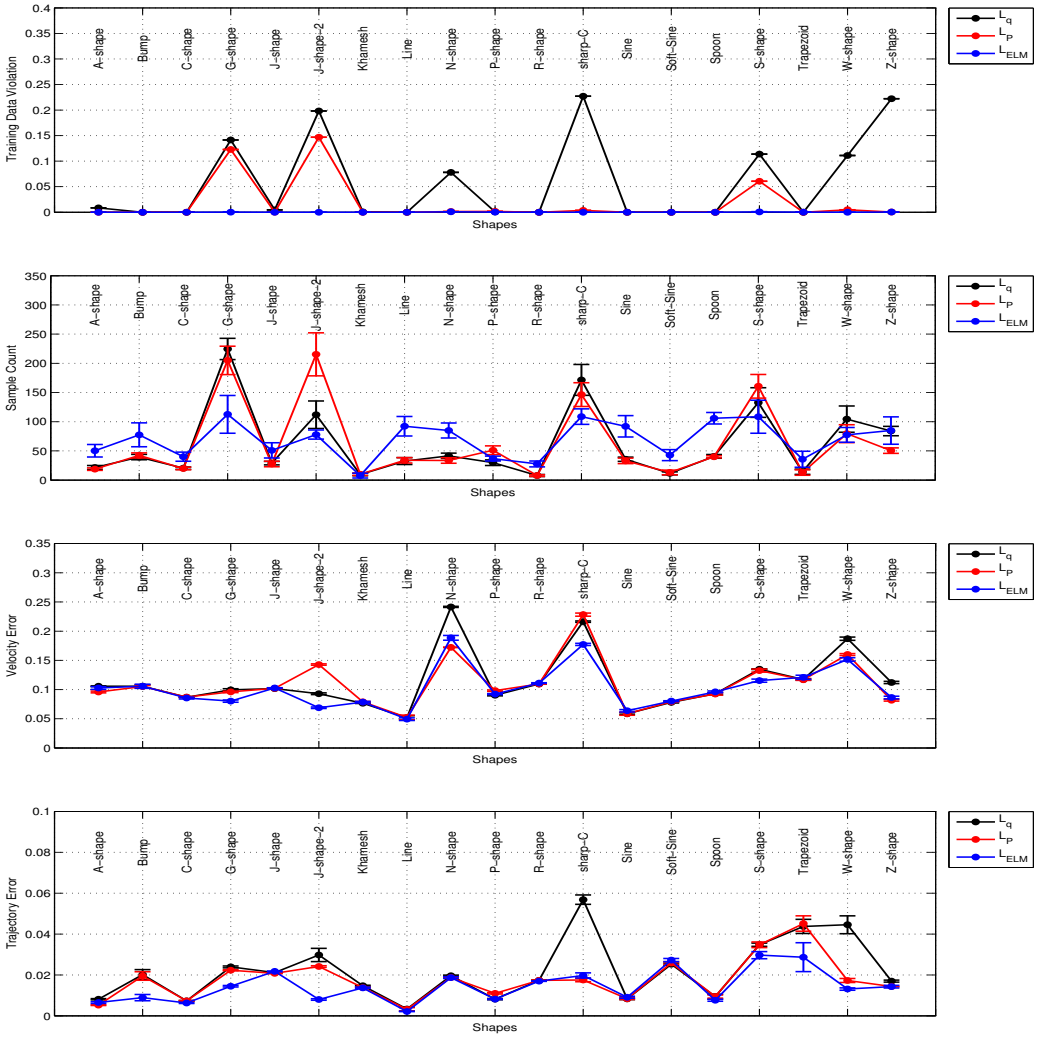

Fig. 7.9: Investigation on the LASA data set. Evaluation of the violation measure M on each shape (first), the used number of samples until implementation of asymptotic stability (second), and the training errors (third).

is strongly curved towards the data and thus prevents a violation of the shapes. These results support the idea that optimized Lyapunov candidates enhance the class of accurately learnable shapes. The second plot of Fig. 7.9 shows the number of samples drawn in the learning phase until implementation of asymptotic stability at $\mathbf{x}^*$. There is again a clear correlation between difficult shapes - the training data which are violated by the respective candidate function indicated by a high value of M - and the number of applied samples, i.e. the difficult shapes need more samples for implementation. Note, that the number of samples for some shapes is

very high for the data independent Lyapunov candidates. Fig. 7.9 (third) shows the velocity error obtained in this experiment. It is obvious that the training error is high for shapes which also produce a high value of violation M. This is also the case for Fig. 7.9 (fourth) which measures the trajectory error of the reproductions for different candidate functions. The networks equipped with the highly flexible L_{ELM} function perform the best among the different candidates.

Fig. 7.12 and Fig. 7.13 comprise the visualization for the results on the entire LASA data set. Fig. 7.12 contains the learned Lyapunov candidate functions L_{ELM} visualized as level sets and Fig. 7.13 shows the corresponding estimate of the dynamics with respect to the demonstrations. Note that all networks produce stable and accurate movements which converge to the given target point attractor due to the combination of optimized Lyapunov candidate and constrained learning. In conclusion, it is demonstrated that data-dependent and flexible Lyapunov candidates are of great significance for the accurate estimation of vector fields.

7.4.3 Kinesthetic Teaching of iCub

The presented Lyapunov approach is applied in a real world scenario involving the humanoid robot iCub [199]. Such robots are typically designed to solve service tasks in environments where a high flexibility is required. Robust adaptability by means of learning is thus a prerequisite for such systems. The experimental setting is illustrated in Fig. 7.10. A human tutor physically guides iCub's right arm in the

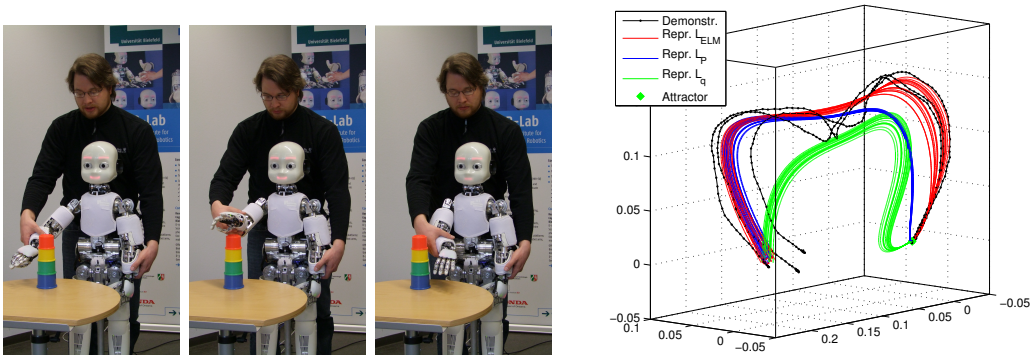

Fig. 7.10: Kinesthetic teaching of iCub (first three plots). The tutor moves iCub's right arm from the right to the left side of the small colored tower. Reproduction of the learned trajectories in meter according to the Lyapunov candidates L_q, L_P, and L_{ELM} (right).

sense of kinesthetic teaching using a recently established compliant force control on the robot [202]. The tutor can thereby actively move all joints of the arm to place the end-effector at the desired position. Beginning on the right side of the workspace, the tutor first moves the arm around the obstacle on the table, touches

its top, and then moves the arm towards the left side of the obstacle were the movement stops. This procedure is repeated three times. The recorded demonstrations comprise between $N_{\text{traj}} = 542$ and $N_{\text{traj}} = 644$ samples. The hidden-layer of the networks estimating the dynamical system consists of $R = 100$ neurons and the regression parameter is $\alpha = 10^{-5}$ in the experiment as well as for the network used for L_{ELM}. The networks' weights and biases are initialized randomly from an uniform distribution in the interval $[-10, 10]$ due to the low ranges of the movement. The results of the experiment are visualized in Fig. 7.10 (right). The figure shows the impact of the different Lyapunov candidates on the estimation of the dynamics. The estimation of the networks trained by the quadratic function L_q is not able to capture the complex shape of the dynamics. The networks trained with the L_P function provides a better performance. The networks using the L_{ELM} function yields an accurate estimate. Fig. 7.11 illustrates the robustness of the learned dynamics against spatial perturbations. Therefore, $N = 75$ starting points are randomly drawn from an uniform distribution in $\Omega = [-0.05, 0.1] \times [-0.5, 0.25] \times [-0.05, 0.2]$. Even the trajectories which start far away from the demonstrations form a trajectory which is influenced by the training data and converge to the target attractor. This observations underline the robustness and accuracy of the proposed learning method for movement generation of humanoid robots by means of programming by demonstration.

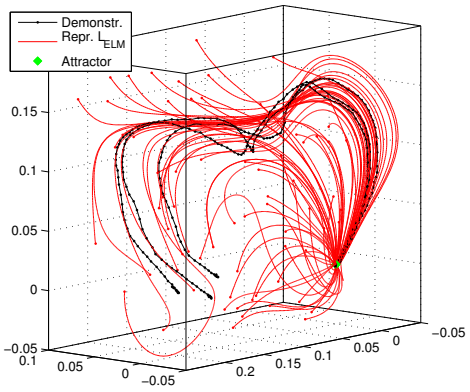

Fig. 7.11: Reproductions of iCub's movements according to L_{ELM} subject to random spatial perturbations.

7.5 Conclusive Remarks

This section introduces a novel learning scheme to approximate vector fields from few demonstrations called "neurally imprinted stable vector fields". The generalization of the vector fields becomes feasible due to the strong bias of the model induced by stability assumptions. Since the dynamical systems considered have only one asymptotically stable fixpoint attractor, the vector fields are shaped by application of Lyapunov's stability theory. The learning scheme principally incorporates linear inequality constraints in discrete points of the workspace derived from a so-called Lyapunov candidate efficiently by the successive quadratic programming technique introduced in Chap. 5. Despite the fact that the constraints are only satisfied in discrete points by construction, it is shown that these constraints can be generalized towards a continuous region. The proposed sampling

strategy permits to choose desired regions for the implementation of constraints because of the locality of the approach. It is shown experimentally that this flexibility can be advantageous due to the large variety of such constraints. They can appear in several bounded regions in possibly lower dimensional subregions of the input space, they can be defined only at certain discrete points or in continuous regions. Several experiments demonstrated that the new learning approach is sufficient to implement accurate estimates from only a few demonstrations and that this leads to stable dynamical systems. Further experiments demonstrated the role of regularization, the obtained errors, and the number of samples needed for implementation of the constraint. In addition, the accuracy of the estimates were enhanced by using a more flexible Lyapunov candidate function also learned by an extreme learning machine which is adapted towards the data. This new Lyapunov candidate relaxed the constraints such that more data sets can be learned with adequate accuracy. Finally, it is demonstrated that the learning scheme can cope with data obtained by kinesthetic teaching of the humanoid robot iCub and it is shown that it generates smooth and accurate reproductions of the learned demonstrations irrespective of strong perturbations.

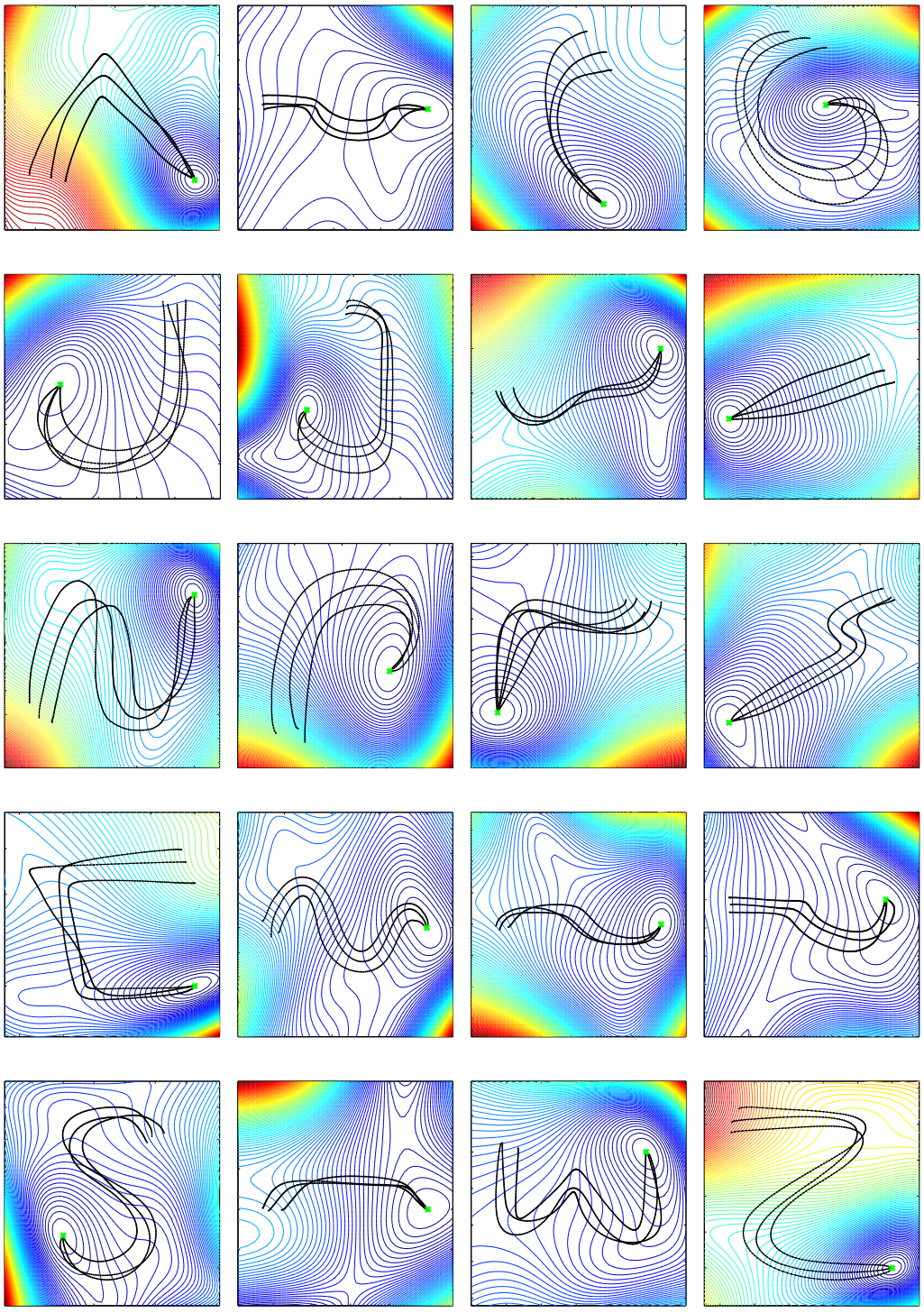

Fig. 7.12: Collection of learned Lyapunov candidates L_{ELM} for the LASA data set. Demonstrations are shown in black and Lyapunov candidates as equipotential lines.

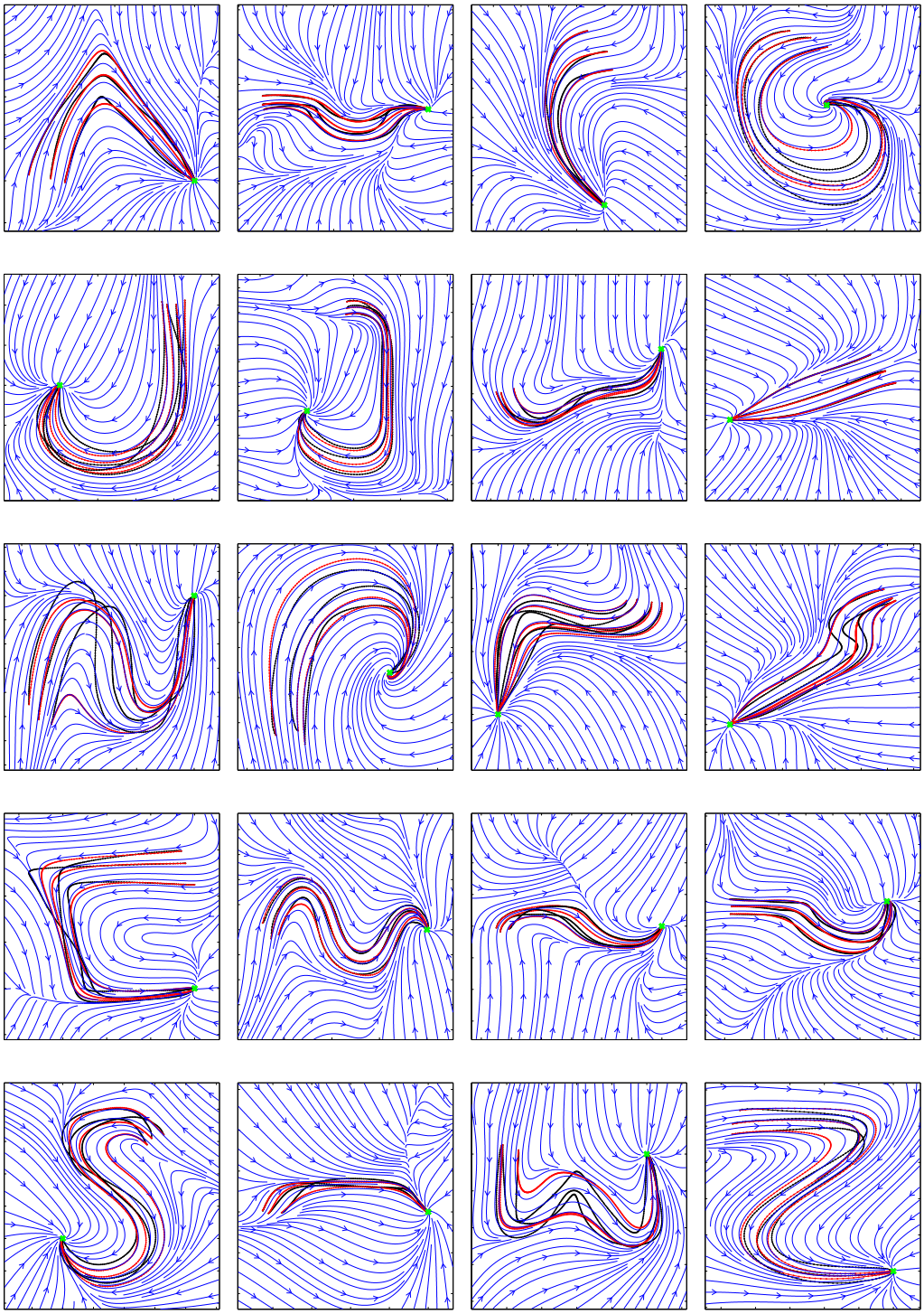

Fig. 7.13: Collection of the corresponding learned dynamical systems $\hat{\mathbf{v}}$ for the entire LASA data set. Reproductions are shown in red and dynamical flow is shown in blue.

Chapter 8

Reliable Modeling of the Ultrasonic Softening Effect by Integrating Prior Knowledge

Power-semiconductor modules are used to control and switch high electrical currents and voltages. Usually, ultrasonic wire bonding is used to connect the electrical terminals of such power modules. Recently, the application of copper instead of aluminum became indispensable because of the ever-growing market of powerful and efficient power modules. Copper has favorable mechanical and electrical properties but its physical behavior in the context of bonding is yet not well understood. A model of the processes involved is essential and desperately desired. This model needs to include the so-called ultrasonic softening effect. It is a key effect within the wire bonding process primarily enabling the robust interconnection between the wire and a substrate. However, the physical modeling of the ultrasonic softening effect is notoriously difficult because of its highly non-linear character and the absence of a proper measurement method. In a first step, this chapter emphasizes the importance of the modeling of the ultrasonic softening effect by showing its impact on the wire deformation experimentally. In a second step, a data-driven model of the ultrasonic softening effect, which is constructed from data using the proposed ELM technique, is presented. A typical caveat of data-driven modeling is the need for training data that cover the considered domain of process parameters in order to achieve accurate generalization of the trained model to new process configurations. In practice, however, the space of process parameters can only be sampled sparsely. It is shown that the novel ELM technique enables the integration of useful prior knowledge about the process into the data-driven modeling process, which results in accurate generalization despite the presence of sparse and noisy data.

The results in this chapter were obtained in collaborative manner in the BMBF funded project InCuB (intelligent copper bonding), which aims at the development of self-optimizing techniques to enable the reliable production of copper wire bonding connections. The common ground for the applied optimization techniques is the learned physical model of the ultrasonic softening effect for copper wire bonding. This chapter is almost completely reproducing the results of the paper by

Unger et al. [21], which summarizes parts of the work achieved in the InCuB project and recalls the main aspects of modeling the ultrasonic softening effect in a data-driven manner.

8.1 The Ultrasonic Softening Effect with Application to Copper Wire Bonding

Ultrasonic wire bonding is an established technology used since decades to connect the electrodes of electrical devices. Because of its flexibility, reliability and cost-efficiency it is widely used for connecting the individual electrodes of microelectronic chips as well as high power semiconductor modules like insulated-gate bipolar transistors (IGBT). Aluminum wire is preferably used in heavy wire applications because of its robust bonding behavior and low cost. Some copper wire bonds on a copper substrate are shown in Fig. 8.1[1]. In recent years, the growing market of powerful and efficient power modules requires a material with better mechanical and electrical properties than aluminium. Therefore, copper wire as bonding material is highly desired. The superior material properties of copper compared to aluminium include significantly higher electrical and thermal conductivity, mechanical stability as well as higher interconnection reliability of copper bonds. Therefore, smaller chips can be operated at higher temperature but with identical switching power leading to reduced costs and higher yield. For these reasons, a technology change from aluminium to copper is indispensable. Typical application fields of products equipped with copper wire bonds are, for instance, the strongly growing markets of renewable energy and electric vehicles [203].

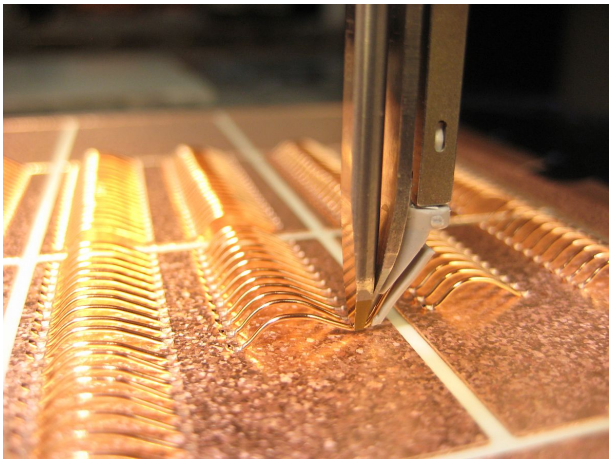

Fig. 8.1: The copper bonding process on a copper substrate. Copyright: Hesse Mechatronics GmbH.

Copper wire bonding is currently in the state of being established as an alternative interconnection method, mainly in thin wire applications, but recently also in heavy wire bonding of power electronics. Because of the different material properties, the bonding parameters in copper wire bonding differ significantly from

[1]Copyright on picture: Hesse Mechatronics GmbH, *www.hesse-mechatronics.de*

those of aluminium wire bonding. Ultrasonic power and the normal bonding forces are about 2 to 3 times higher. The copper wire bonding process also reacts more sensitive to parameter changes. This makes manufacturing of reliable copper bond connections challenging.

In order to increase the reliability of the copper bonds, an adaptation of the bonding parameters at runtime is desired. For this purpose, the german BMBF[2] funded the project InCuB[3], which develops self-optimizing and cognitive learning techniques to enable the reliable production of copper bonding connections under varying conditions. The bonding machine will be provided with the abilities to react to different external influences. Such influences can be the temperature, the material of the substrate, and process parameters (e.g. the normal force and its duration). Optimal compromises between several objectives need to be computed before operation. Therefore, multi-objective optimization techniques are employed [204] and a model of the entire bonding process is required.

Since the interaction of the these parameters is largely unknown and an accurate physical model is not yet available, learning is an reasonable means to construct a model of the copper bonding process. Thus, data and specific prior knowledge about the physics of copper bonding are provided to construct and analyze the performance of the model. The data-driven modeling extracts regularities of the bonding process from exemplary bonds produced with different process parameters. The trained model can then generalize these regularities to unseen process parameters. The only issue is the generalization from only few training examples. Accurate generalization of the data-driven model is again only achieved because specific prior knowledge about the bonding process is provided for learning. The data-driven development of this model through learning is done in the context of the self-optimization project[4] also part of the leading-edge cluster "it's OWL".

Several contributions investigating the bonding process with aluminum are already published. Some of these works deal with the quality control and are used in commercial products: a method called process integrated quality control (PiQC) is introduced in [205, 206, 207, 208]. The construction of reliable copper bonds is a persistently challenging task and not yet understood satisfactory [209]. One reason is that the prior knowledge gained from the aluminum bonding process [210] is not necessarily negotiable - a separate accurate model of copper bonding is desired. The modeling of the ultrasonic softening effect by means of finite element methods [211, 212] is, in principle, possible but the construction of a general model remains difficult and computationally expensive. Also, many of these models fit important parameters through data in a calibration step [213].

[2]Bundesministerium für Bildung und Foschung, Federal Republic of Germany.
[3]InCuB is a project within the "Intelligent Technical Systems" leading-edge cluster.
[4]Information about the self-optimization project can be found on *its-owl.de*

8.2 Copper Wire Deformation by Ultrasonic Softening

Ultrasonic wire bonding is a cold friction welding process. The wire is placed under the tip of a slim rod-like bonding tool (see Fig. 8.1). It is pressed onto the electrode

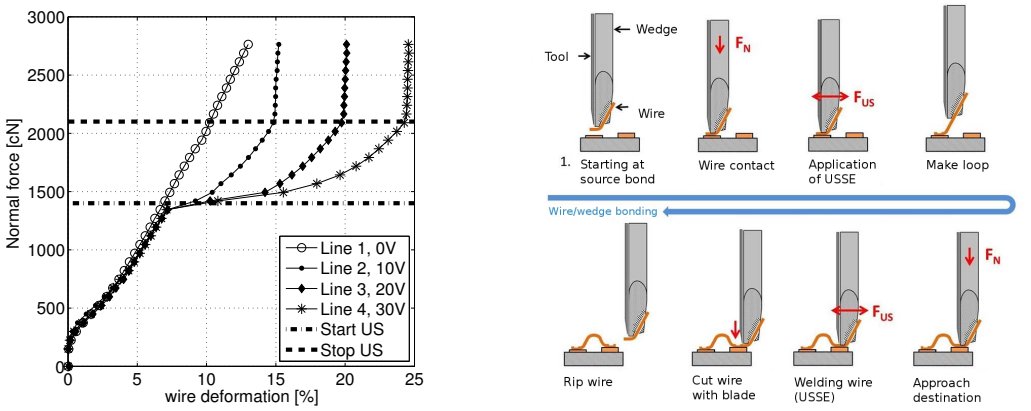

Fig. 8.2: Deformation experiment showing the ultrasonic softening effect (left). Schematic illustration of the copper wire bonding process by means of ultrasonic softening (right).

surface with a well-selected normal force causing an initial cold straining at the contact area. A so-called ultrasonic transducer generates mechanical vibrations in the ultrasonic range, e.g. 60 kHz, which are transferred by the bonding tool into the welding area. The deformation of the wire and the adhesion between wire and substrate steadily progress during this welding process. Finally a pure intermetallic compound between wire and electrode is formed at temperatures well below the melting point of the bonding partners. After the first bond the machine forms the so-called loop and then the second bond connection is established. After cutting the wire, the tool is lifted and the interconnection process is finished. The bonding process is schematically illustrated in Fig. 8.2 (right).

It is essential to consider all effects to build a model of the bonding process. These effects are: the static elasto-plastic deformation, the ultrasonic softening effect, and the proceeding adhesion between wire and substrate. This work particularly focuses on the ultrasonic softening effect. This effect was first presented in [214]. It describes the macroscopic softening of the material under applied ultrasound. The yield strength is lowered such that the material can be manipulated at lower mechanical stresses and forces. The ultrasonic softening has a significant influence on the bonding process, because it enables the deformation and growth of the bonding area with reasonable normal forces.

Deformation tests with the bonding machine are conducted to investigate the ultrasonic softening effect. The ultrasound is turned on and off during each experiment to analyze the effect of the high frequency vibrations. In each experiment, the normal force acting on the wire is increased linearly from 100 cN up to

2800 cN within 1200 ms. The bonding machine and in particular the bonding tool is not perfectly rigid, which makes it necessary to measure its elastic deformation. Therefore, the bonding tool is placed on a hard substrate, onto which it exerts the nominal force. The measured elastic deformation of the bonding machine is then subtracted from the total deformation in the regular bonding experiments. Ultrasonic softening and hardening effects are clearly visible in the observed wire deformations shown in Fig. 8.2 (left). Line 1, depicted in the figure, is a typical force-strain curve for a radially deformed wire without ultrasound. In this experiment, the static normal force only induces a deformation of approximately 13%. In experiments corresponding to line 2, 3, and 4, the ultrasound is applied between 1400 cN and 2100 cN with an altering amplitude from 10 V to 30 V. The wire softens considerably during the application of ultrasound. The amount of deformation increases with higher ultrasound voltages. After shutdown of the ultrasound, the curves show a hardening effect since the gradient of the force-deformation curves are increased compared to line 1 (see Fig. 8.2, left). As expected, the total deformation after applying ultrasound increases with the ultrasonic amplitude.

Ultrasonic softening is a highly non-linear effect acting on a microscopic scale. The InCuB research project requires an accurate model that can be computed efficiently. This thesis therefore proposes to use the ELM technique with additional prior knowledge about the physical effect, which leads to accurate models and are efficient to compute at the same time.

8.3 Data-Driven Modeling with Integration of Prior Knowledge

This section presents a data-driven approach to model the ultrasonic softening effect. First, the model's input-output structure is introduced. Then, the ELM modeling technique including the integration of prior knowledge about the bonding process into the learning is explained.

8.3.1 The Copper Wire Bonding Model

A schematic view of the ideal copper wire bonding process as considered in this setup, i.e. without external perturbations, is shown in Fig. 8.3. The input of the model is the point in time t of the process, the applied ultrasonic voltage $U_S(t)$ and the normal force $F_N(t)$ between bonding tool and substrate. The output of the model is the wire deformation $\hat{D}(t)$ at time t. The ultrasonic voltage and normal force are approximately constant during the bonding experiments considered in this experimental setup. Therefore, the model can be understood as a function of time parameterized by the bonding parameters, i.e. ultrasonic voltage and normal force. This leads to a two-dimensional encoding of the bonding process.

The model needs to quantify the impact of different process parameter configurations which are not contained in the data set for learning. Besides this generalization of the training data to unseen process configurations, an efficient evaluation of the model is an additional requirement. Training can be accomplished offline and is decoupled from the evaluation phase.

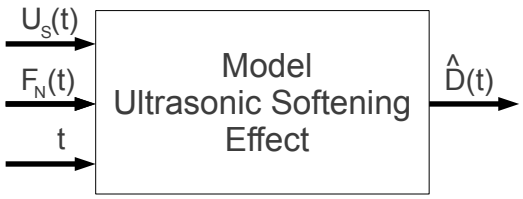

Fig. 8.3: Schematic view of the model of the ultrasonic softening effect.

8.3.2 Data-Driven Modeling with Extreme Learning Machines

To model the ultrasonic softening effect in a data-driven manner, again, the extreme learning machine (ELM) technique is applied. The input of the ELM is denoted by $\mathbf{x} = (t|U_S|F_N) \in \mathbb{R}^{I=3}$, the hidden-layer by $\mathbf{h} \in \mathbb{R}^R$, and the output is denoted by $\hat{D} \in \mathbb{R}^{O=1}$. For input $\mathbf{x}$, the output is computed by

$$\hat{D}(\mathbf{x}) = \sum_{j=1}^{R} W_j^{\mathbf{out}} f(\sum_{k=1}^{I} W_{jk}^{\mathbf{inp}} x_k + b_j) \ , \tag{8.1}$$

where $\hat{D}(\mathbf{x})$ encodes the wire deformation during the copper wire bonding process for input configuration $\mathbf{x}$. Let $\mathcal{D} = (X, D) = (\mathbf{x}^k, D^k)$ with $k = 1 \dots N_{\text{tr}}$ be the data set for training, where N_{tr} is the number of training samples, X is the collection of input configurations, and D is the matrix of target wire deformations. Inputs and outputs are normalized to the range [-1, 1] according to the distribution of the training data.

8.3.3 Learning with Prior Knowledge

Learning a well applicable model of the ultrasonic softening effect from few training samples is particularly challenging because only sparse information about the underlying mapping is given. It might be possible that larger parts of the parameter space controlling the bonding process remain uncovered with data. Therefore, there is considerable need for generalization towards regions subject to sparse sampling. Fortunately, prior knowledge about physical properties of the bonding process is available:

(**1**) The wire deformation is monotonically increasing in time.

(**2**) The wire deformation is monotonically increasing w.r.t. the ultrasonic voltage because higher voltages lead to a stronger wire deformation.

(**3**) The wire deformation is monotonically increasing w.r.t. the normal force of the tool. The normal force puts pressure on the wire and thus leads to a stronger wire deformation.

It is possible to rephrase this specific prior knowledge in mathematical terms using the model input variable $\mathbf{u} = (t|U_S|F_N) \in \mathbb{R}^3$ to formulate point-wise constraints:

$$\begin{aligned}
&(\mathbf{1})\ \partial_1 \hat{D}(\mathbf{u}) = \frac{\partial}{\partial t}\hat{D}(\mathbf{u}) > 0 : \forall t \in \Omega\ , \\
&(\mathbf{2})\ \partial_2 \hat{D}(\mathbf{u}) = \frac{\partial}{\partial U_S}\hat{D}(\mathbf{u}) > 0 : \forall U_S \in \Omega\ , \\
&(\mathbf{3})\ \partial_3 \hat{D}(\mathbf{u}) = \frac{\partial}{\partial F_N}\hat{D}(\mathbf{u}) > 0 : \forall F_N \in \Omega\ ,
\end{aligned} \tag{8.2}$$

where Ω is a predefined region in the model's input space.

The read-out weights are then trained by solving the proposed successively applied quadratic program optimizing $W^{\mathbf{out}}$ subject to a set of collected point constraints $U_i = \{\mathbf{u}_i^1, \ldots, \mathbf{u}_i^{N_\mathrm{u}}\}$ in order to implement the conditions (**1**), (**2**), and (**3**):

$$\begin{aligned}
W^{\mathbf{out}} = &\arg\min_W (\|W \cdot H(X) - D\|^2 + \alpha \|W\|^2) \\
&\text{subject to: } \partial_i \hat{D}(U_i) > 0 : i = 1, \ldots, 3\ ,
\end{aligned} \tag{8.3}$$

where the matrices $H(X)$ and T again collect the hidden states and targets for inputs X, respectively, and α is a regularization parameter. Solving the quadratic program guarantees satisfaction of the given constraints with respect to the discrete inputs U_i, which is already useful in many applications. The generalization of the discrete constraints U_i towards the continuous region Ω is again achieved by applying the sampling strategy proposed in Chap. 5.

8.4 Experimental Results

This section explains the experimental setup that was applied to acquire data from a bonding machine producing bond connections with copper wire. The generalization ability of the proposed ELM technique is analyzed systematically and compared to the classical ELM technique.

8.4.1 Experimental Setup

A Hesse Mechatronics Bondjet BJ939 bonding machine equipped with a standard wire bondhead is used for data acquisition. This bondhead is designed for bonding with copper wires in the range of 100 μm to 500 μm diameter. Tab. 8.1 lists the specifications of the bondhead, wire and substrate which are used for all experiments. In order to obtain training data for the data-driven modeling, a 5×5 -

Specification	Value
Bondhead type	RBK01 Back-cut
Transducer Type	60 kHz
Digital Generator Power Output	100 W
Wire Size	500 μm (Cu)
Substrate	DCB-thickness: 0.38 mm ceramic, 0.3 mm Cu

Tab. 8.1: Specification of the bondhead, wire and substrate.

grid covering the region of interest of the bonding parameters (voltage $U_S\,(t)$ and normal force $F_N\,(t)$) is created. The minimum and maximum ultrasonic voltage as well as the minimum and maximum normal forces are limited by the process. The ultrasonic voltage $U_S(t)$ is varied from 44 V to 52 V and the normal force $F_N\,(t)$ is varied from 3000 cN to 3800 cN both in 5 equally distant steps. For each grid point, ten individual bond connections are produced. The applied ultrasonic voltage $U_S(t)$ and normal force $F_N\,(t)$ are approximately constant throughout bonding. The resulting wire deformation trajectories $\hat{D}\,(t)$ for each process configuration are recorded, downsampled and averaged. The measured wire deformation is the height reduction during the time interval between touchdown of the tool on the wire and the end of the ultrasonic vibrations.

The experiments are conducted in random order to avoid time-dependent deviations in the data set due to environmental influences. To be able to differentiate between the first and second bond of each interconnection, the type of bond was logged as well. For both of these types, individual models are trained.

8.4.2 Learning Setup

ELMs with (CELM) and without (ELM) the application of the continuous constraints derived from prior knowledge about the bonding process are trained on the recorded data. To analyze the generalization ability of the data-driven models, a leave-one-out cross-validation is conducted. Training is repeated such that each of the 25 sampled process configurations is once left out from the training set and serves as test scenario. Additionally, the impact of the ELM parameters on the generalization performance is evaluated by changing the hidden-layer size $R \in [30, 50, 100]$ and the regularization parameter $\alpha \in [10^{-4}, 10^{-6}, 10^{-8}]$. The model performance is evaluated on the training and test set by computing the error between estimated and recorded wire deformation averaged over all time steps. Results are averaged over 10 independent ELM initializations.

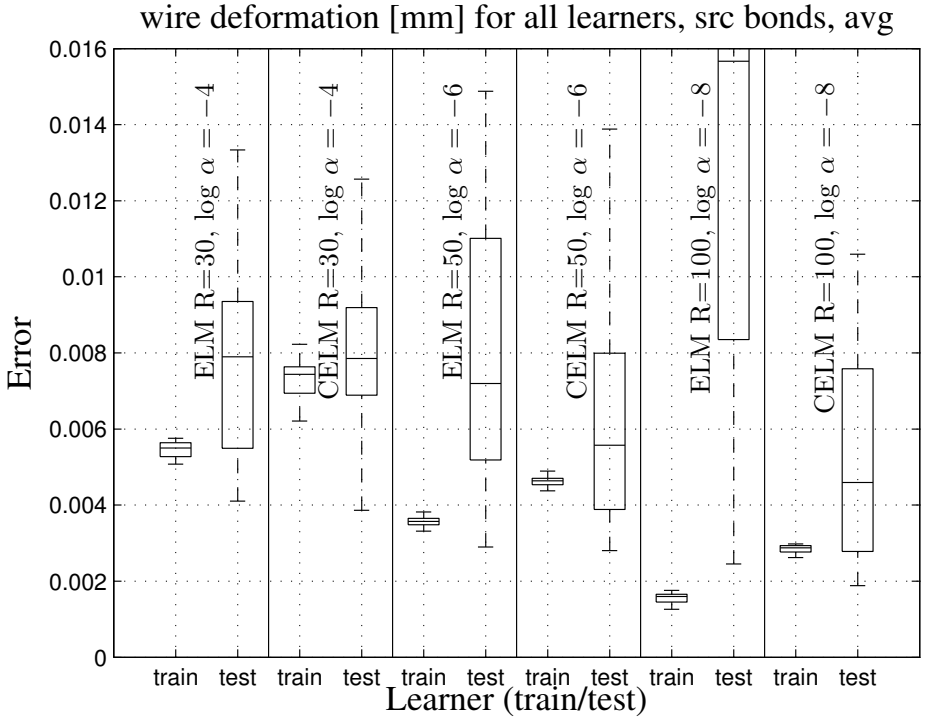

Fig. 8.4: Cross-validation errors for training and test sets of ELMs without (ELM) and with (CELM) constraints.

8.4.3 Results

The cross-validation results for ELM networks equipped with and without prior knowledge are shown in Fig. 8.4. The best performing network among the networks without prior knowledge has a hidden-layer with $R = 50$ neurons and a regularization parameter of $\alpha = 10^{-6}$. ELMs with $R = 100$ hidden neurons and a regularization parameter of $\alpha = 10^{-8}$ show a typical behavior when learning from few data: they achieve a low training error around $E_{\mathrm{tr}} = 0.002$ but have high corresponding test errors of about $E_{\mathrm{te}} = 0.016$. This big gap between training and test errors indicates strong overfitting. Applying the learning with constraints, the same networks have a slightly increased training error around $E_{\mathrm{tr}} = 0.003$, but the test error is significantly decreased towards $E_{\mathrm{te}} = 0.0045$. Note that the generalization capability is increased by application of prior knowledge irrespective of the network parameters in this scenario. This demonstrates that the application of suitable constraints alleviates the problem of overfitting when training data is sparse and that the integration of prior knowledge into the learning facilitates reliable and robust generalization.

Fig. 8.5 shows the results of the data-driven modeling with and without constraints in more detail. Each cell of the panel corresponds to the process param-

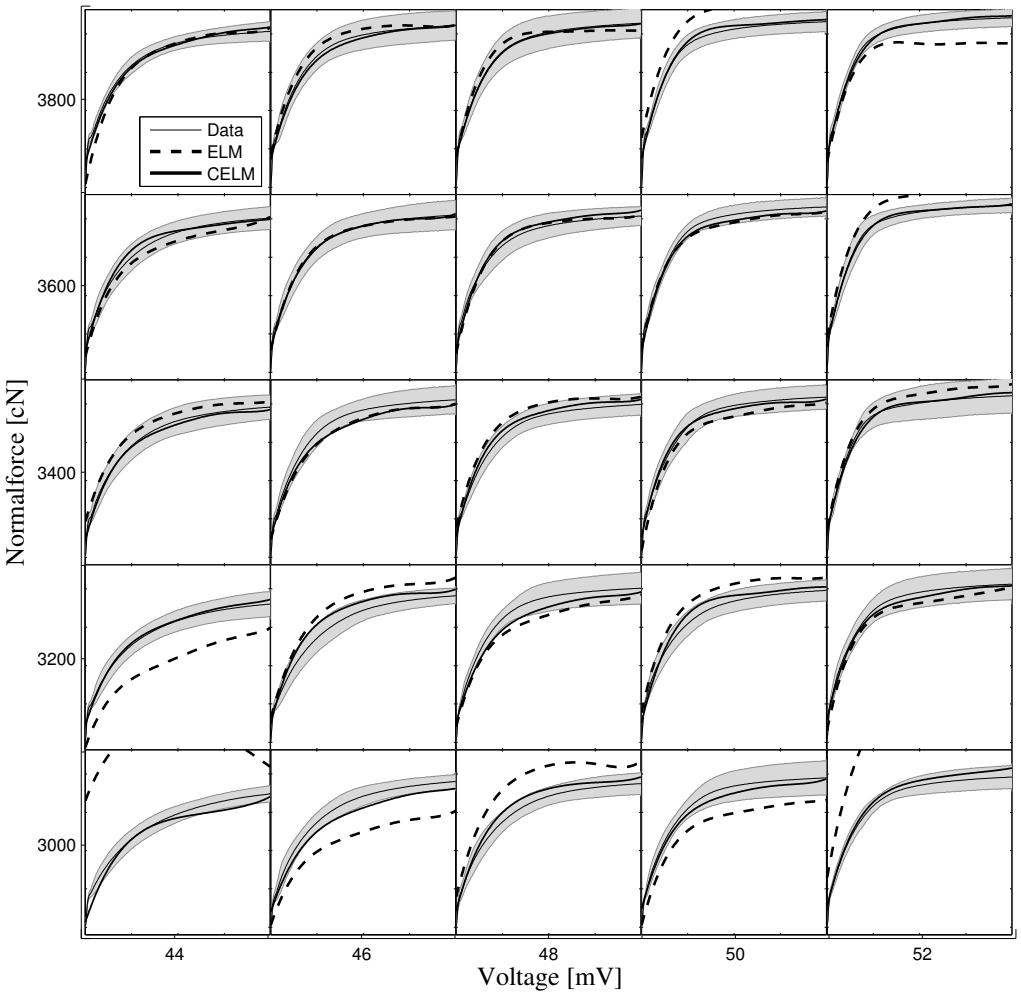

Fig. 8.5: The cross-validation results of the copper bonding experiment visualized for a ELM model without and with the application of constraints ($R = 100$ and $\alpha = 10^{-8}$). The recorded data is depicted in gray and the generalization of the respective parameter configuration is depicted by dashed (ELM) or solid lines (CELM). The constrained learning result in a good generalization performance within the standard deviation of the recorded wire deformations.

eters of the bonding machine indicated by the surrounding axes. In each cell, the recorded wire deformation over time for these bonding parameters is shown by a solid line with standard deviation computed from the ten repeated bonding experiments (gray areas). The generalization performance of models trained with and without prior knowledge are plotted by solid bold and dashed lines, respectively. Due to the high degree of non-linearity, a complex model is required in order to capture the structure of the data. However, a high model complexity, i.e. ELMs

with large hidden-layers and small regularization constants, are prone to overfitting and poor generalization when trained on sparse and noisy data. In particular, the parameter configurations at the corner and the edge of the matrix are difficult to generalize (see the corners of Fig. 8.5). This is reasonable since the models need to extrapolate the training examples to these process configurations. Thus, the data-driven modeling without prior knowledge needs a careful fine-tuning of its regularization parameter or network size. On the contrary, Fig. 8.5 demonstrates that the integration of prior knowledge about the underlying process into the learning by means of linear inequality constraints mitigates the issue of sparse data and results in a significantly increased generalization performance.

8.5 Conclusive Remarks

This chapter introduces the InCuB project which aims at the optimization of the copper wire bonding process. The ultimate goal of this research is the reliable production of copper wire bond connections under varying conditions for the production of reliable power modules. The basis for the optimization is a data-driven model because no analytic model is available. For the in-process adaptation of the bonding parameters, a model-based optimization together with self-optimizing techniques will be applied to achieve this goal in future work. For this purpose, a validated physical model of the process, which can be computed efficiently, is mandatory. Although ultrasonic wire bonding is widely used and the ultrasonic softening effect has been investigated in previous studies, there is still no model of the bonding process or the ultrasonic softening available which fulfills these requirements.

This chapter shows that the data-driven ELM model of the ultrasonic softening for copper wires for the first time which is accurate and efficient to compute. The issue of poor generalization from sparse data is addressed by integrating prior knowledge about the ultrasonic softening effect into the learning which then yields to accurate generalization and reduced overfitting. Supplementing data-driven modeling techniques with prior knowledge about the underlying process results in highly reliable models and is a promising methodology for a broader range of applications in the modeling domain of complex physical processes.

Chapter 9

Conclusion

The development and application of well-performing machine learning algorithms is an important next step towards the competitive industries of tomorrow. However, a good performance is not the only requirement for machine learning algorithms that are applied in such domains. It is at the same time strongly desired that they satisfy certain reliability requirements, while sustaining the universal approximation capability and advantages of data-driven modeling methods. This is particularly difficult because, in real applications, data are typically noisy, outliers occur, and there is always a risk of overfitting corrupted data. In addition, principally, data sampling is very expensive and thus only few samples can be provided in a conceivably high-dimensional input space.

The extreme learning machine (ELM) is a recently developed and potentially promising algorithm for the application in challenging engineering tasks. It mainly attracts researchers because of its structural simplicity and the efficient learning procedure. However, the black-box character of this technique appears as one of the major disadvantageous. This thesis therefore proposes a novel learning approach based on the concept of ELMs that focuses on the reliability of the learning.

The term "reliability" is referred to three actual requirements that are investigated in rigorous detail: first, the robustness of the learning results to different initializations of the input-output mapping which is especially important for random projections. Second, the consideration of input drifts which usually occur in long-term scenarios, and, third, the incorporation of prior knowledge about the task which tackles the undesired black-box character of ELMs.

At first, batch intrinsic plasticity (BIP) is introduced in Chap. 3, which is the "batch" version of intrinsic plasticity rule by Triesch. The method is capable of learning requested output distributions such as the exponential distribution and leads to the optimization of the information transmission through the hidden-layer neurons by entropy maximization. It is demonstrated that the generalization capability of the ELM networks used in several different robotic setups is enhanced. A successful application of BIP for ELMs is not surprising, since IP was already analyzed for other neural network techniques [11]. In conclusion, it is demonstrated that BIP reduces the dependence of the network's performance on their initialization in a computationally efficient manner.

In Chap. 4, IP learning is also applied for unsupervised compensation of input drifts. It is of great advantage that the input data does not need to be analyzed,

which offers a different and novel approach without application of continuous error feedback. It is demonstrated that the proposed modification of the IP rule and the usage of the natural gradient provides learning dynamics that are well suited for the online compensation of input drifts in a robotic scenario with the humanoid robot iCub. These chapters (Chap. 3 and Chap. 4) show that IP learning as unsupervised learning scheme is an interconnected means to handle different important aspects of reliability for ELM learning.

It is essential that the black-box character of random projection methods is tackled in tasks which crucially require a save application. The incorporation of prior knowledge about the specific tasks seems encouraging, because it cannot be expected that the learner automatically fulfills this prior knowledge by solely applying the data set for learning. In this thesis, prior knowledge is represented by continuous constraints in a previously defined workspace. The associated algorithms and experiments are discussed in Chap. 5. However, the actual incorporation is not trivial because the ELM paradigm possesses the universal approximation capability. It is nevertheless shown that large classes of prior knowledge can be formulated as continuous constraints on the read-out weights. Besides simple cases of prior knowledge such as partial monotonicity, convexity, or bounds on the output, also complex types are expressible such as state space stability of the to-be-approximated dynamics for the Puma560 robot and Lyapunov stability for movement generation for the humanoid robot iCub. Some "rules of thumb" are also derived from the experiments. First, the impact of prior knowledge is higher if the data set is noisy and/or sparse. This fact was used through-out entire thesis. Second, the strength of the Tikhonov regularization is directly linked to the constraint incorporation. Hence, the regularization parameter appears as an control variable to handle the curse of dimensionality. In summary, it is shown that this novel approach provides all the tools to combine the full representational power of the ELM with guarantees on the performance of the learned function, because the form of the ELM is particularly well suited for the incorporation of continuous constraints.

Finally, it is demonstrated that the proposed approach leads to excellent results in three different real world scenarios emphasizing its generality and relevance. The first example shows experimentally that the proposed approach is able to accelerate the control of the bionic handling assistant (BHA) significantly by learning the physical relation between pressure and expansion of a chamber. Learning is particularly hard since only few and very noisy data samples can be provided due to the slow convergence of the BHA towards the mechanical equilibrium for a certain pressure value. This issue was resolved by integration of physical prior knowledge about the robot into the ELM approach, which lead to a robust model for the control of the BHA. The proposed approach is also compared to other models implementing prior knowledge. However, these models only guarantee a single specific constraint in the entire state space and thus have less degrees of freedom resulting in a decreasing generalization ability.

Chap. 7 comprises experiments that show the proposed approach in the field of computational imitation learning and movement generation from few demonstrations in robotics. Stability in terms of Lyapunov's theory appears as a indispensable property for dynamical systems providing robots with the abilities to perform complex movements robustly. The proposed approach is thus applied to enforce stability by means of constraint sampling from a predefined Lyapunov candidate function. In fact, most of the state of the art approaches for estimation of dynamical systems only deal with simple and data-independent Lyapunov functions which renders the finding of a satisfactory solution to the trade-off between stability and accuracy difficult. The significance of the learning of highly flexible Lyapunov function candidates is thus highlighted and analyzed in a scenario where trajectories are learned and generalized from human handwriting motions. It is shown that the flexible learned Lyapunov candidate enables the proposed approach to learn robust movement generation from robotic data obtained by kinesthetic teaching of the humanoid robot iCub.

Chap. 8 shows how the proposed approach is applied to construct a precise model for the copper wire bonding process. This model is designated for the InCuB-project which develops a self-optimization procedure leading to robust copper wire bond connections. Learning in this scenario is only successful because prior knowledge about the specific tasks is available and leads to reliable results despite that the provided data are sparse and noisy. However, the time-correlated form of the training data of the applications in Chap. 7 and Chap. 8 prevents pretraining by BIP. Such tasks remain difficult for unsupervised tuning; a further investigation is left for future research.

The real world examples provide both, highly relevant applications and additional insight on the usefulness and generality of the proposed scheme. It demonstrates a typical case where data-driven learning is needed because no analytic models are available. However, learning is difficult because sampling can be done only in the real world setup and is thus expensive and potentially noisy. The ELM provides a promising approach for such application domains, because learning for this method is efficient while good generalization can be expected. Only the unreliable character of the original approach seems problematic. This issue is approached in three different aspects and it is showen that the reliability of ELMs is enhanced significantly. I expect that this treatment can open new application domains for data driven learning, where reliability is crucial and traditionally only analytic modeling has been possible.

Appendix A

Appendix

A.1 Related References by the Author

[13] - **Klaus Neumann and Jochen J. Steil. Batch Intrinsic Plasticity for Extreme Learning Machines. Proc. Int. Conf. on Artificial Neural Networks, vol. 6791, no. 1, pp. 339-346, 2011.**
This paper introduces batch intrinsic plasticity as a method to optimize extreme learning machines. The pretraining aims at desired output distributions of the hidden neurons and was inspired by intrinsic plasticity [45]. All authors contributed to the research design. The content of this paper is related to Chap. 3. I presented the paper at the ICANN 2011 in Helsinki, Finland

[14] - **Klaus Neumann and Jochen J. Steil. Optimizing Extreme Learning Machines via Ridge Regression and Batch Intrinsic Plasticity. Neurocomputing (Special Issue ELM 2012), vol. 102, pp. 23-30, 2013.**
This paper analyzes batch intrinsic plasticity in more detail. In particular together with ridge regression for robotic applications. All authors contributed to the research design. The content of the paper is used in Chap. 3. I presented the paper at the ELM 2011 in Hangzhou, China.

[12] - **Klaus Neumann, Christian Emmerich, and Jochen J. Steil. Regularization by Intrinsic Plasticity and its Synergies with Recurrence for Random Projection Methods. Journal of Intelligent Learning Systems and Applications vol. 4, no. 3, pp. 230-246, 2012.**
The Paper investigates the role of intrinsic plasticity as a feature regularization and recurrence as a technique to produce a non-linear mixture of sigmoid features for random projections. The paper was written during my PhD time but most of the actual content were developed in my master's thesis. Emmerich performed experiments regarding the role of the recurrent weights for static reservoir computing. All authors contributed to the research design. Parts of the paper are used in Chap. 2.

[15] - **Klaus Neumann and Jochen J. Steil. Intrinsic Plasticity via Natural Gradient Descent. Proc. Europ. Symp. on Artificial Neural Networks, pp. 555-560, 2012.**
This paper introduces the natural gradient descent for intrinsic plasticity which is

interpreted as stochastic gradient descent. All authors contributed to the research design. This paper is related to Chap. 4. I presented the paper at ESANN 2012 in Bruges, Belgium.

[16] - **Klaus Neumann, Claudius Strub, and Jochen J. Steil. Intrinsic Plasticity via Natural Gradient Descent with Application to Drift Compensation. Neurocomputing (Special Issue ESANN 2012), vol. 112, pp. 26-33, 2013.**
This paper extends the natural gradient descent for intrinsic plasticity in [15] by another modification of intrinsic plasticity and applies this novel learning rule to compensate input drifts. Main parts of this work considering the application to drift compensation and the working point transformation originate from the master thesis of Claudius Strub under my supervision. All authors contributed to the research design. This paper is the basis for Chap. 4.

[17] - **Klaus Neumann, Matthias Rolf, and Jochen J. Steil. Reliable Integration of Continuous Constraints into Extreme Learning Machines. Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, vol. 21, no. supp02, pp. 35-50, 2013.**
This paper proposes the ideas to use continuous constraints as prior knowledge and introduces an algorithm for implementation into extreme learning machines. It is shown that this algorithm also works in a real world scenario involving the bionic handling assistant. I developed the method to incorporate prior knowledge and Rolf provided the data set of the BHA. All authors contributed to the research design. Parts of the paper are used in Chap. 5 and Chap. 6. I presented the paper at the ELM 2013 in Singapore and received a best presentation award.

[18] - **Andre Lemme, Klaus Neumann, R. Felix Reinhart, and Jochen J. Steil. Neurally Imprinted Stable Vector Fields. Proc. Europ. Symp. on Artificial Neural Networks, pp. 327-332, 2013.**
This paper introduces a neural network strategy to model dynamical systems learned from data, e.g. in the field of imitation learning in robotics. The paper is a joint work of the European project "AMARSi" and the leading-edge cluster "it's owl". Andre Lemme and Felix Reinhart mainly developed the idea of movement generation through a vector field representation while I developed the algorithm based on Lyapunov's stability theory to enforce asymptotic stability by means of the constrained learning scheme introduced in [17]. All authors contributed to the research design. The paper is mainly related to Chap. 7. I presented the paper at the ESANN 2013 in Bruges, Belgium and received a best student paper award.

[19] - **Klaus Neumann, Andre Lemme, and Jochen J. Steil. Neural Learning of Stable Dynamical Systems based on Data-Driven Lyapunov Candidates. Int. Conf. on Intelligent Robots and Systems, pp. 1216-**

1222, 2013.
The paper proposes a method to learn Lyapunov candidate functions from data which I developed for the estimation of vector fields as in [18]. All authors contributed to the research design. The content is related to Chap. 7. I presented the paper at IROS 2013 in Tokyo, Japan.

[20] - **Andre Lemme and Klaus Neumann and R. Felix Reinhart and Jochen J. Steil. Neural Learning of Vector Fields for Encoding Stable Dynamical Systems. Neurocomputing, In Press.**
This paper investigates the results of [18] in more detail in a special issue. In particular, the method is tested in a humanoid robot scenario involving iCub. All authors contributed to the research design. The paper is published in the special issue of the ESANN 2013 conference. The content is strongly related to Chap. 7. The paper will appear soon.

[21] - **Andreas Unger and Walter Sextro and Simon Althoff and Klaus Neumann and R. Felix Reinhart and Michael Brökelmann and Daniel Bolowski and Karsten Guth. Investigation and modeling of the ultrasonic softening effect for the copper wire bonding process. Int. Conf. on Integrated Power Electronics Systems (CIPS 2014), In Press.**
This conference contribution investigates the ultrasonic softening effect for copper wire bonding and proposes an approach which models this effect with additonal use of prior domain knowledge about the task. While Reinhart developed the cross-validation framework, I provided the software library to enable the incorporation of prior knowlegde. The Incub project and partners[1] provided the data set for data-driven learning and investigations about the ultrasonic softening effect for copper wire bonding. All authors contributed to the research design. The content is related to Chap. 8. The paper will appear at the beginning of 2014 and Unger and I will present the paper at CIPS 2014 in Nürnberg, Germany.

A.2 Proof of the Proposition in Eq. (5.14)

The separation of the output into a linear combination of the hidden states can be used to determine the upper bound of the remainder for a single neuron for the Taylor approximation. Partial derivatives of the hidden states are given in order to calculate the remainder terms:

$$D^m h_j(\mathbf{u}) = f^{(M)}(a_j \sum_k W^{\mathbf{inp}}_{jk} u_k + b_j) \cdot a_j^M W^{\mathbf{inp}}_{jm_1} \dots W^{\mathbf{inp}}_{jm_M} \tag{A.1}$$

[1] Members of the "Lehrstuhl für Mechatronik und Dynamik der Fakultät für Maschinenbau", the Hesse Mechatronics GmbH, and the Infineon Technologies AG.

Since a Taylor approximation of the constraint is performed, the remainder term is written in Lagrange form. The upper bound therefore becomes:

$$rem(\mathbf{u},\mathbf{u}^0) = \frac{1}{3!}\left[(\mathbf{u}-\mathbf{u}^0)^T\nabla\right]^3 \sum_i \gamma_i D^{\mathbf{m}}\, y_i(\mathbf{u}) - c|_{\mathbf{u}\in\mathcal{S}} \tag{A.2}$$

$$= \frac{1}{3!}\left[(\mathbf{u}-\mathbf{u}^0)^T\nabla\right]^3 \sum_i \gamma_i D^{\mathbf{m}} \sum_j W_{ij}^{\text{out}}\, h_j(\mathbf{u})|_{\mathbf{u}\in\mathcal{S}} \tag{A.3}$$

$$= \frac{1}{3!}\sum_i \gamma_i \sum_j W_{ij}^{\text{out}}\left[(\mathbf{u}-\mathbf{u}^0)^T\nabla\right]^3 D^{\mathbf{m}} h_j(\mathbf{u})|_{\mathbf{u}\in\mathcal{S}} \tag{A.4}$$

$$= \frac{1}{3!}\sum_i \gamma_i \sum_j W_{ij}^{\text{out}}\left[(u_j-u_j^0)^T\frac{\partial}{\partial u_j}\right]^3 D^{\mathbf{m}} h_j(u_j)|_{\mathbf{u}\in\mathcal{S}} \tag{A.5}$$

$$\leq \frac{1}{6}\sum_i \gamma_i \sum_j W_{ij}^{\text{out}} M_j |\underline{u}_j - \overline{u}_j|^3 = rem : \forall \mathbf{u},\mathbf{u}^0 \in \mathcal{S}\ , \tag{A.6}$$

where $u_j = W_j^{\mathbf{inp}}\mathbf{u}$ is the projection of the input onto the weight vector of the jth hidden neuron by means of the scalar product. Eq. (A.6) holds subject to the following maximizations and minimizations for the one-dimensional case:

$$M_j = \max_{u_j\in[\underline{u}_j,\overline{u}_j]}\left[f^{(M+3)}(a_j u_j + b_j)\cdot a_j^{M+3} W_{jm_1}^{\mathbf{inp}}\ldots W_{jm_M}^{\mathbf{inp}}\right], \tag{A.7}$$

$$\underline{u}_j = \min_{\mathbf{u}\in\mathcal{S}} W_j^{\mathbf{inp}}\mathbf{u}\ ,\ \overline{u}_j = \max_{\mathbf{u}\in\mathcal{S}} W_j^{\mathbf{inp}}\mathbf{u}\ . \tag{A.8}$$

The step from Eq. (A.4) to Eq. (A.5) is only possible because the Fermi-functions are one-dimensional with respect to the respective weight vector. Furthermore, since the effectively one-dimensional Fermi-function f is used as activation and derivatives of this function are always analytically known polynomials in f, a calculation of the neuron-specific maximum M_j in the step from Eq. (A.7) to Eq. (A.8) is possible with very small numerical error $\approx 10^{-16}$. Note that the implementation of this algorithm can be significantly simplified by using specific geometrical shapes for the region $\mathcal{S}$, i.e. convex polytopes, which are more practicable for calculation reasons. In the example described in Sect. 5.5.3, a rectangular region in the two-dimensional input space is used. The regions for the BHA implementation where cubic (see Chap. 6). Then both the projection (see Eq. (5.12)) of the region onto the weight vector of a neuron becomes very simple (only the edge-point of the rectangle needs to be projected onto the vector in order to get $\underline{u}_j$ and $\overline{u}_j$) and the calculation of the global extrema in Eq. (5.15) is simpler - standard-functions can be used.

References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] Y. Miche. *Developing Fast Machine Learning Techniques with Applications to Steganalysis Problems*. PhD thesis, Aalto University, Helsinki, Finland, 2010.

[3] M. Kukar, I. Kononenko, and S. Ljubljana. Reliable classifications with machine learning. In *Proc. European Conference on Machine Learning*, pages 219–231, 2002.

[4] M. van Heeswijk, Y. Miche, E. Oja, and A. Lendasse. GPU-Accelerated and Parallelized ELM Ensembles for Large-scale Regression. *Neurocomputing*, 74:2430–2437, 2011.

[5] S. Haykin. *Neural Networks: a Comprehensive Foundation*, volume 1. Englewood Cliffs, NJ, Prentice-Hall, 1999.

[6] K.-I. Funahashi. On the Approximate Realization of Continuous Mappings by Neural Networks. *Neural Networks*, 2(3):183–192, 1989.

[7] H. Allende, R. Nanculef, and R. Salas. Robust Bootstrapping Neural Networks. In *Advances in Artificial Intelligence*, volume 2972, pages 813–822. 2004.

[8] Y. Miche, B. Schrauwen, and A. Lendasse. Machine Learning Techniques Based on Random Projections. In *Proc. Europ. Symp. on Artificial Neural Networks*, pages 295–302, 2010.

[9] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme Learning Machine: a New Learning Scheme of Feedforward Neural Networks. In *IEEE Proc. Int. Joint Conf. on Neural Networks*, pages 985–990, 2004.

[10] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme Learning Machine: Theory and Applications. *Neurocomputing*, 70(1–3):489–501, 2006.

[11] J. J. Steil. Online Reservoir Adaptation by Intrinsic Plasticity for Backpropagation Decorrelation and Echo State Learning. *Neural Networks*, 20(3):353–364, 2007.

[12] K. Neumann, C. Emmerich, and J. J. Steil. Regularization by Intrinsic Plasticity and its Synergies with Recurrence for Random Projection Methods. *Journal of Intelligent Learning Systems and Applications*, 4(3):230–246, 2012.

[13] K. Neumann and J. J. Steil. Batch Intrinsic Plasticity for Extreme Learning Machines. In *Proc. Int. Conf. on Artificial Neural Networks*, pages 339–346, 2011.

[14] K. Neumann and J. J. Steil. Optimizing Extreme Learning Machines via Ridge Regression and Batch Intrinsic Plasticity. *Neurocomputing*, 102:23–30, 2013.

[15] K. Neumann and J. J. Steil. Intrinsic Plasticity via Natural Gradient Descent. In *Proc. Europ. Symp. on Artificial Neural Networks*, pages 555–560, 2012.

[16] K. Neumann, C. Strub, and J. J. Steil. Intrinsic Plasticity via Natural Gradient Descent with Application to Drift Compensation. *Neurocomputing*, 112:26–33, 2013.

[17] K. Neumann, M. Rolf, and J. J. Steil. Reliable Integration of Continuous Constraints into Extreme Learning Machines. *Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 21(supp02):35–50, 2013.

[18] A. Lemme, K. Neumann, R. F. Reinhart, and J. J. Steil. Neurally Imprinted Stable Vector Fields. In *Proc. Europ. Symp. on Artificial Neural Networks*, pages 327–332, 2013.

[19] K. Neumann, A. Lemme, and J. J. Steil. Neural Learning of Stable Dynamical Systems based on Data-Driven Lyapunov Candidates. In *IEEE Proc. Int. Conf. on Intelligent Robots and Systems*, pages 1216–1222, 2013.

[20] A. Lemme, K. Neumann, R. F. Reinhart, and J. J. Steil. Neural Learning of Vector Fields for Encoding Stable Dynamical Systems. *Neurocomputing*. In Press.

[21] A. Unger, W. Sextro, S. Althoff, T. Meyer, K. Neumann, F. R. Reinhart, M. Brökelmann, D. Bolowski, and K. Guth. Modeling the Ultrasonic Softening Effect for Robust Copper Wire Bonding. In *Proc. Int. Conf. on Integrated Power Electronics Systems*, 2014. In Press.

[22] L.P. Wang and C.R. Wan. Comments on the Extreme Learning Machine. *IEEE Trans. on Neural Networks*, 19(8):1494–1495, 2008.

[23] G.-B. Huang. Reply to Comments on the Extreme Learning Machine. *IEEE Trans. on Neural Networks*, 19(8):1495–1496, 2008.

[24] G.-B. Huang, D. Wang, and Y. Lan. Extreme Learning Machine: a Survey. *Int. Journal of Machine Learning and Cybernetics*, 2(2):107–122, 2011.

[25] R. Penrose. A Generalized Inverse for Matrices. In *Mathematical Proc. of the Cambridge Philosophical Society*, pages 406–413, 1955.

[26] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan. A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks. *IEEE Trans. on Neural Networks*, 17(6):1411–1423, 2006.

[27] Q. Zhu, A. Qin, P. Suganthan, and G.-B. Huang. Evolutionary Extreme Learning Machine. *Pattern Recognition*, 38(10):1759–1763, 2005.

[28] G. Feng, G.-B. Huang, Q. Lin, and R. Gay. Error Minimized Extreme Learning Mmachine with Growth of Hidden Nodes and Incremental Learning. *IEEE Trans. on Neural Networks*, 20:1352–1357, 2009.

[29] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[30] F. Han, H.-F. Yao, and Q.-H. Ling. An Improved Extreme Learning Machine Based on Particle Swarm Optimization. In *Proc. Int. Conf. on Intelligent Computing*, volume 6840, pages 699–704, 2012.

[31] F. Girosi, M. Jones, and T. Poggio. Regularization Theory and Neural Networks Architectures. *Neural Computation*, 7:219–269, 1995.

[32] C. M. Bishop. Training with Noise is Equivalent to Tikhonov Regularization. *Neural Computation*, 7(1):108–116, 1995.

[33] W. Deng, Q. Zheng, and L. Chen. Regularized Extreme Learning Machine. In *Proc. IEEE Symp. on Computational Intelligence and Data Mining*, pages 389–395, 2009.

[34] A. N. Tikhonov and V. Y. Arsenin. Solution of Incorrectly Formulated Problems and the Regularization Method. *Soviet Math. Doklady*, 4:1035–1038, 1963.

[35] A. N. Tikhonov. Solutions of Ill-Posed Problems. *Mathematics of Computation*, 32(144):1320–1322, 1978.

[36] B. Schrauwen, M. Wardermann, D. Verstraeten, J. J. Steil, and D. Stroobandt. Improving Reservoirs using Intrinsic Plasticity. *Neurocomputing*, 71(7):1159–1171, 2008.

[37] G.-B. Huang, L. Chen, and C.-K. Siew. Universal Approximation using Incremental Constructive Feedforward Networks with Random Hidden Nodes. *IEEE Trans. on Neural Networks*, 17(4):879–892, 2006.

[38] G.-B. Huang and L. Chen. Enhanced Random Search Based Incremental Extreme Learning Machine. *Neurocomputing*, 71:3460–3468, 2008.

[39] Y. Miche, A. Sorjamaa, and A. Lendasse. OP-ELM: Theory, Experiments and a Toolbox. In *Proc. Int. Conf. on Artificial Neural Networks*, pages 145–154, 2008.

[40] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse. OP-ELM: Optimally Pruned Extreme Learning Machine. *IEEE Trans. on Neural Networks*, 21(1):158–162, 2010.

[41] H.-J. Rong, Y.-S. Ong, A.-H. Tan, and Z. Zhu. A Fast Pruned-Extreme Learning Machine for Classification Problems. *Neurocomputing*, 72(1–3):359–366, 2008.

[42] Y. Miche, M. van Heeswijk, P. Bas, O. Simula, and A. Lendasse. TROP-ELM: A Double-Regularized ELM using LARS and Tikhonov Regularization. *Neurocomputing*, 74(16):2413–2421, 2011.

[43] C. Emmerich, F. R. Reinhart, and J. J. Steil. Recurrence Enhances the Spatial Encoding of Static Inputs in Reservoir Networks. In *Proc. Int. Conf. on Artificial Neural Networks*, pages 148–153, 2010.

[44] F. R. Reinhart. *Reservoir Computing with Output Feedback*. PhD thesis, Faculty of Technology, Bielefeld University, 2011.

[45] J. Triesch. A Gradient Rule for the Plasticity of a Neurons Intrinsic Excitability. In *Proc. Int. Conf. on Artificial Neural Networks*, pages 65–70, 2005.

[46] J. J. Steil. Several Ways to Solve the MSO Problem. In *Proc. Europ. Symp. on Artificial Neural Networks*, pages 489–494, 2007.

[47] K. A. Krycha and U. Wagner. Applications of Artificial Neural Networks in Management Science: a Survey. *Journal of Retailing and Consumer Services*, 6(4):185–203, 1999.

[48] P. J. G. Lisboa. A Review of Evidence of Health Benefit from Artificial Neural Networks in Medical Intervention. *Neural Networks*, 15(1):11–39, 2002.

[49] G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Math. Control Signals Systems*, 2:303–314, 1989.

[50] K. Hornik, M. Stinchcombe, and H. White. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, 2(5):359–366, 1989.

[51] K. Hornik. Approximation Capabilities of Multilayer Feedforward Networks. *Neural Networks*, 4(2):251–257, 1991.

[52] G. E. Hinton D. E. Rumelhart and R. J. Williams. Learning Representations by Back-Propagating Errors. *Nature*, 323:533–536, 1986.

[53] D. Fradkin and D. Madigan. Experiments with Random Projections for Machine Learning. In *Proc. Int. Conf. on Knowledge Discovery and Data Mining*, pages 517–522, 2003.

[54] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz Mappings into a Hilbert Space. In *Proc. Conf in Modern Analysis and Probability*, volume 26, pages 189–206, 1984.

[55] S. Dasgupta. Experiments with Random Projections. In *Proc. Conf. on Uncertainty in Artificial Intelligence*, pages 143–151, 2000.

[56] E. Bingham and H. Mannila. Random Projection in Dimensionality Reduction: Applications to Image and Text Data. *Knowledge Discovery and Data Mining*, pages 245–250, 2001.

[57] S. Kaski. Dimensionality Reduction by Random Mapping: Fast Similarity Computation for Clustering. In *Proc. IEEE Int. Joint Conf. on Neural Networks*, volume 1, pages 413–418, 1998.

[58] C. Hegde, M. B. Wakin, and R. G. Baraniuk. Random projections for manifold learning. In *Proc. Neural Information Processing Systems*, 2007.

[59] X. Z. Fern and C. E. Brodley. Random Projection for High Dimensional Data Clustering: a Cluster Ensemble Approach. In *Proc. Int. Conf. on Machine Learning*, pages 186–193, 2003.

[60] Y. H. Pao and Y. Takefuji. Functional-Link Net Computing, Theory, System Architecture, and Functionalities. *IEEE Computation*, 25(5):76–79, 1992.

[61] Y. H. Pao. *Adaptive Pattern Recognition and Neural Networks*, volume 1. Addison-Wesley, 1989.

[62] B. Igelnik and Y. H. Pao. Stochastic Choice of Basis Functions in Adaptive Function Approximation and the Functional-Link Net. *IEEE Trans. on Neural Networks*, 6(6):1320–1329, 1995.

[63] Y. H. Pao, G. H. Park, and D. J. Sobajic. Learning and Generalization Characteristics of the Random Vector Functional-Link Net. *Neurocomputing*, 6(2):163–180, 1994.

[64] B. Igelnik, Y. H. Pao, S. R. LeClair, and C. Y. Shen. The ensemble approach to neural-network learning and generalization. *IEEE Trans. on Neural Networks*, 10(1):19–30, 1999.

[65] G. H. Park and Y. H. Pao. Unconstrained Word-Based Approach for Off-Line Script Recognition using Density-Based Random-Vector Functional-Link Nets. *Neurocomputing*, 31(1–4):45–65, 2000.

[66] D. Husmeier and J. G. Taylor. Modelling Conditional Probabilities with Committees of RVFL Networks. In *Proc. Int. Conf. on Artificial Neural Networks*, pages 1053–1058, 1997.

[67] S. Dehuri and S.-B. Cho. A Comprehensive Survey on Functional-Link Neural Networks and an Adaptive PSO-BP Learning for CFLNN. *Neural Computing and Applications*, 19(2):187–205, 2010.

[68] D. S. Broomhead and D. Lowe. Multivariable Functional Interpolation and Adaptive Networks. *Complex Systems*, 2:321–355, 1988.

[69] D. Lowe. Adaptive Radial Basis Function Nonlinearities, and the Problem of Generalisation. In *Proc. Int. Conf. on Artificial Neural Networks*, pages 171–175, 1989.

[70] T. Poggio and F. Girosi. A Theory of Networks for Approximation and Learning. *A.I. Memo, Massachusetts Institute of Technology*, 1140, 1989.

[71] J. Moody and C. J. Darken. Fast Learning in Networks of Locally-Tuned Processing Units. *Neural Computation*, 1(2):281–294, 1989.

[72] D. Wettschereck and T. Dietterich. Improving the Performance of Radial Basis Function Networks by Learning Center Locations. In *Proc. Neural Information Processing Systems*, pages 1133–1140, 1992.

[73] K.-I. Funahashi and Y. Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801–806, 1993.

[74] P. J. Werbos. Backpropagation through Time: what it does and how to do it. *Proc. of the IEEE*, 78(10):1550–1560, 1990.

[75] R. J. Williams and D. Zipser. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, 1:270–280, 1989.

[76] K. Doya. Bifurcations in the Learning of Recurrent Neural Networks. *Proc. IEEE Int. Symp. on Circuits and Systems*, 6:2777–2780, 1992.

[77] M. Lukoševičius and H. Jaeger. Reservoir Computing Approaches to Recurrent Neural Network Training. *Computer Science Review*, 3(3):127–149, 2009.

[78] Y. Bengio, P. Simard, and P. Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Trans. on Neural Networks*, 5(2):157–166, 1994.

[79] B. Schrauwen, D. Verstraeten, and J. Van Campenhout. An Overview of Reservoir Computing: Theory, Applications and Implementations. In *Proc. Europ. Symp. on Artificial Neural Networks*, pages 471–482, 2007.

[80] M. Hermans and B. Schrauwen. Recurrent Kernel Machines: Computing with Infinite Echo State Networks. *Neural Computation*, 24(6):104–133, 2011.

[81] D. Verstraeten, B. Schrauwen, and D. Stroobandt. Reservoir-Based Techniques for Speech Rrecognition. In *Proc. World Conf. on Computational Intelligence*, pages 1050–1053, 2006.

[82] M. D. Skowronski and J. G. Harris. Automatic Speech Recognition using a Predictive Echo State Network Classifier. *Neural Networks*, 20(3):414–23, 2007.

[83] K. Neumann, M. Rolf, J. J. Steil, and M. Gienger. Learning Inverse Kinematics for Pose-Constraint Bi-Manual Movements. In *Proc. Int. Conf. on Adaptive Behaviour (Simulation of Adaptive Behaviour)*, pages 478–488, 2010.

[84] E. A. Antonelo, B. Schrauwen, and D. Stroobandt. Event Detection and Localization for Small Mobile Robots using Reservoir Computing. *Neural Networks*, 21(6):862–71, 2008.

[85] M. Rolf, J. J. Steil, and M. Gienger. Efficient Exploration and Learning of Whole Body Kinematics. In *Proc. IEEE Int. Conf. Development and Learning*, pages 1–7, 2009.

[86] R. F. Reinhart and J. J. Steil. Reaching Movement Generation with a Recurrent Neural Network Based on Learning Inverse Kinematics for the Humanoid Robot iCub. In *Proc. Humanoids*, pages 323–330, 2009.

[87] P. Buteneers, B. Schrauwen, D. Verstraeten, and D. Stroobandt. Real-time Epileptic Seizure Detection on Intra-Cranial Rat Data using Reservoir Computing. In *Proc. Int. Conf. on Advances in Neuro-Information Processing*, pages 56–63, 2009.

[88] B. Noris, M. Nobile, L. Piccinini, M. Berti, M. Molteni, E. Berti, F. Keller, D. Campolo, and A. Billard. Gait Analysis of Autistic Children with Echo State Networks. Workshop on Echo State Networks and Liquid State Machines, 2006.

[89] A. F. Krause, B. Bläsing, V. Dürr, and T. Schack. Direct Control of an Active Tactile Sensor Using Echo State Networks. In *Human Centered Robot Systems*, volume 6, pages 11–21. 2009.

[90] T. Natschläger W. Maass and Henry Markram. Real-Time Computing without Stable States: a new Framework for Neural Computation Based on Perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[91] H. Jaeger. The Echo State Approach to Analysing and Training Recurrent Neural Networks. *Technical Report GMD, German National Research Center for Information Technology*, 148, 2001.

[92] U. D. Schiller and J. J. Steil. On the Weight Dynamcis of Recurrent Learning. In *Proc. Europ. Symp. on Artificial Neural Networks*, pages 73–78, 2003.

[93] U. D. Schiller and J. J. Steil. Analyzing the Weight Dynamics of Recurrent Learning Algorithms. *Neurocomputing*, 63:5–23, 2005.

[94] A. F. Atiya and A. G. Parlos. New Results on Recurrent Network Training: Unifying the Algorithms and Accelerating Convergence. *IEEE Trans. on Neural Networks*, 11(3):697–709, 2000.

[95] J. J. Steil. Backpropagation-Decorrelation: Recurrent Learning with O(N) Complexity. In *Proc. Int. Joint Conf. on Neural Networks*, pages 843–848, 2004.

[96] J. J. Steil. Stability of Backpropagation-Decorrelation Efficient O(N) Recurrent Learning. In *Proc. Europ. Symp. on Artificial Neural Networks*, pages 43–48, 2005.

[97] J.J. Steil. Online Stability of Backpropagation-Decorrelation Recurrent Learning. *Neurocomputing*, 69(7–9):642–650, 2006.

[98] F. Biegler-Koenig and F. Baermann. A Learning Algorithm for Multilayered Neural Networks Based on Linear Least Squares Problems. *Neural Networks*, 6(1):127–131, 1993.

[99] J. Y. F. Yam, T. W. S. Chow, and C. T. Leung. A New Method in Determining the Initial Weights of Feedforward Neural Networks. *Neurocomputing*, 16(1):23–32, 1997.

[100] E. Castillo, O. Fontenla-Romero, B. Guijarro-Berdiñas, and A. Alonso-Betanzos. A Global Optimum Approach for One-Layer Neural Networks. *Neural Computation*, 14(6):1429–1449, 2002.

[101] O. Fontenla-Romero, D. Erdogmus, J. C. Principe, A. Alonso-Betanzos, and E. Castillo. Linear Least-Squares Based Methods for Neural Networks Learning. In *Proc. Int. Joint Conf. on Artificial Neural Networks and Neural Information Processing*, pages 84–91, 2003.

[102] E. Castillo. A Very Fast Learning Method for Neural Networks Based on Sensitivity Analysis. *Journal of Machine Learning Research*, 7:1159–1182, 2006.

[103] B. Widrow, A. Greenblatt, Y. Kim, and D. Park. The No-Prop Algorithm: a New Learning Algorithm for Multilayer Neural Networks. *Neural Networks*, 37:182–188, 2013.

[104] G. Li, P. Niu, X. Duan, and X. Zhang. Fast Learning Network: a Novel Artificial Neural Network with Fast Learning Speed. *Neural Computing and Applications*, pages 1–13, 2013.

[105] J. S. Prakash R. Rajesh. Extreme Learning Machines - A Review and State-of-the-art. *Int. Journal of Wisdom based Computing*, 1(1):35–49, 2011.

[106] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang. Extreme learning machine for regression ans multi-class classification. *IEEE Transactions on Systems, Man, and Cybernetics: Part B*, 42(2):513–529.

[107] J. Triesch. Synergies between Intrinsic and Synaptic Plasticity in Individual Model Neurons. In *Proc. Neural Information Processing Systems*, pages 1417–1424, 2004.

[108] J. Triesch. Synergies between Intrinsic and Synaptic Plasticity Mechanisms. *Neural computation*, 19(4):885–909, 2007.

[109] N. J. Butko and J. Triesch. Learning Sensory Representations with Intrinsic Plasticity. *Neurocomputing*, 70(7):1130–1138, 2007.

[110] P. I. Corke. *Robotics, Vision and Control: Fundamental Algorithms in Matlab*. Springer, 2011.

[111] A. Frank and A. Asuncion. UCI Machine Learning Repository, 2010.

[112] G.-B. Huang and L. Chen. Convex Incremental Extreme Learning Machine. *Neurocomputing*, 70(16–18):3056–3062, 2007.

[113] C. Strub. Permanent Intrinsic Plasticity of Dendridic and Neuronal Excitability. Master's thesis, CoR-Lab, Bielefeld University, 2012.

[114] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with Drift Detection. In *Proc. Brazilian Symposium on Artificial Intelligence*, pages 286–295, 2004.

[115] R. Klinkenberg and T. Joachims. Detecting Concept Drift with Support Vector Machines. In *Proc. Int. Conf. on Machine Learning*, pages 487–494, 2000.

[116] A. Dries and U. Rückert. Adaptive Concept Drift Detection. *Stat. Anal. Data Mining*, 2(5–6):311–327, 2009.

[117] M. Holmberg and T. Artursson. Drift Compensation, Standards and Calibration Methods. *Handbook of Machine Olfaction: Electronic Nose Technology*, pages 325–346, 2002.

[118] S.-I. Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10:251–276, 1998.

[119] S.-I. Amari. Differential-Geometrical Methods in Statistics. *Lecture Notes in Statistics, New-York: Springer-Verlag*, 28, 1985.

[120] M. K. Murray and J. W. Rice. Differential Geometry and Statistics. *Chapman and Hall/CRC*, 1993.

[121] S.-I. Amari and S. C. Douglas. Why Natural Gradient? *Acoustics, Speech, and Signal Processing*, 2:1213–1216, 1998.

[122] C. C. Douglas, J. Hu, J. Ray, D. T. Thorne, and R. S. Tuminaro. Natural-Gradient Adaptation. *Unsupervised Adaptive Filtering: Blind Source Separation*, 1:13–61, 2000.

[123] M. Rattray, D. Saad, and S.-I. Amari. Natural Gradient Descent for On-Line Learning. *Physical Review Letters*, 81:5461–5464, 1998.

[124] M. Rattray and D. Saad. Analysis of Natural Gradient Descent for Multilayer Neural Networks. *Physical Review E*, 59:4523–4532, 1999.

[125] T. Heskes. On Natural Learning and Pruning in Multilayered Perceptrons. *Neural Computation*, 12(4):881–901, 2000.

[126] A. Gonzalez and J. R. Dorronsoro. Natural learning in NLDA networks. *Neural Networks*, 20(5):610–620, 2007.

[127] S. Kullback and A. Leibler. On Information and Sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 1951.

[128] L. Kostal and P. Lansky. Classification of Stationary Neuronal Activity According to its Information Rate. *Network*, 17(2):193–210, 2006.

[129] C. R. Rao. Information and the Accuracy Attainable in the Estimation of Statistical Parameters. *Bull. Calcutta Math. Soc.*, 37:81–91, 1945.

[130] A. Freire, A. Lemme, J. J. Steil, and G. Barreto. Learning Visuo-Motor Coordination for Pointing without Depth Calculation. *Proc. Europ. Symp. on Artificial Neural Networks*, pages 91–96, 2012.

[131] A. Lemme, A. Freire, G. Barreto, and J. J. Steil. Kinesthetic Teaching of Visuomotor Coordination for Pointing by the Humanoid Robot iCub. *Neurocomputing*, pages 179–188, 2013.

[132] Bernhard Lang. Monotonic Multi-layer Perceptron Networks as Universal Approximators. In *Proc. Int. Conf. on Artificial Neural Networks*, pages 750–750, 2005.

[133] H. Daniels and M. Velikova. Monotone and Partially Monotone Neural Networks. *IEEE Trans. on Neural Networks*, 21(6):906–917, 2010.

[134] E. Hartman. Training Feedforward Neural Networks with Gain Constraints. *Neural Computation*, 12(4):811–829, 2000.

[135] D. D. Lee and H. S. Seung. Algorithms for Non-Negative Matrix Factorization. In *Proc. Neural Information Processing Systems*, pages 556–562, 2000.

[136] M. V. Velikova. *Monotone Models for Prediction in Data Mining.* PhD thesis, Tilburg University, Tilburg, Netherlands, 2006.

[137] B.-G. Hu, Y. Wang, S. H. Yang, and H. B. Qu. How to add Transparency to Artificial Neural Networks. *Pattern Recognition and Artificial Intelligence*, 20(1):72–84, 2007.

[138] D. Psichogios and L. H. Ungar. A Hybrid Neural Network-First Priciples Approach to Process Modeling. *American Institute of Chemical Engineers Journal*, 38(10):1499–1511, 1992.

[139] M. L. Thompson and M. A. Kramer. Modeling Chemical Processes Using Prior Knowledge and Neural Networks. *American Institute of Chemical Engineers Journal*, 40(8):1328–1340, 1994.

[140] P. Niyogi, F. Girosi, and T. Poggio. Incorporating Prior Information in Machine Learning by Creating Virtual Examples. In *Proc. of the IEEE*, volume 86, pages 2196–2209, 1998.

[141] C. J. C. Burges and B. Schölkopf. Improving the Accuracy and Speed of Support Vector Machines. In *Proc. Neural Information Processing Systems*, pages 375–381, 1997.

[142] B. Schölkopf, P. Simard, A. Smola, and V. Vapnik. Prior Knowledge in Support Vector Kernels. *Proc. Neural Information Processing Systems*, pages 640–646, 1998.

[143] T. Poggio and F. Girosi. Networks for Approximation and Learning. In *Proc. of the IEEE*, volume 78, pages 1481–1497, 1990.

[144] T. Yu. *Incorporating Prior Domain Knowledge into Inductive Machine Learning: its Implementation in Contemporary Capital Markets*. PhD thesis, University of Technology, Sydney, Australia, 2007.

[145] T. Yu, S. Simoff, and T. Jan. VQSVM: a Case Study for Incorporating Prior Domain Knowledge into Inductive Machine Learning. *Neurocomputing*, 73(13–15):2614–2623, 2010.

[146] O. Bousquet, S. Boucheron, and G. Lugost. Introduction to Statistical Learning Theory. *Advanced Lectures on Machine Learning, Lecture Notes in Artificial Intelligence*, 3176:169–207, 2004.

[147] Y. Tanaka, M. Fukushima, and T. Ibaraki. A Comparative Study of Several Semi-Infinite Nonlinear Programming Algorithms. *European Journal of Operational Research*, 1:92–100, 1988.

[148] W. H. Joerding and J. L. Meador. Encoding a Priori Information in Feedforward Networks. *Neural Networks*, 4(6):847–856, 1991.

[149] J. Sill and Y. S. Abu-Mostafa. Monotonicity Hints. In *Proc. Neural Information Processing Systems*, pages 643–640, 1997.

[150] Y. S. Abu-Mostafa. Hints. *Neural Computation*, 7:639–671, 1995.

[151] A. Selonen, J. Lampinen, and L. Ikonen. Using External Knowledge in Neural Network Models. In *Proc. SPIE on Intelligent Robots and Computer Vision: Algorithms, Techniques, Active Vision, and Materials Handling*, pages 239–249, 1996.

[152] F. Wang and Q. J. Zhang. Incorporating Functional Knowledge into Neural Networks. In *Proc. Int. Conf. on Neural Networks*, pages 266–269, 1997.

[153] C. Dugas, Y. Bengio, F. Belisle, C. Nadeau, and R. Garcia. Incorporating Functional Knowledge in Neural Networks. *Journal of Machine Learning Research*, 10:1239–1262, 2009.

[154] L. Haichuan, S. Hongye, X. Lei, G. Yong, and R. Gang. Multiple-Prior-Knowledge Neural Network for Industrial Processes. In *Proc. IEEE Int. Conf. on Automation and Logistics*, pages 385–390, 2010.

[155] R. Andrews and S. Geva. On the Effects of Initialising a Neural Network with Prior Knowledge. In *Proc. Neural Information Processing Systems*, pages 251–256, 1999.

[156] F. Lauer and G. Bloch. Incorporating Prior Knowledge in Support Vector Regression. *Machine Learning*, 70:89–118, 2008.

[157] O. L. Mangasarian and E.W. Wild. Nonlinear knowledge in kernel approximation. *IEEE Trans. on Neural Networks*, 18:300–306, 2007.

[158] Y.-J. Qu and B.-G. Hu. Generalized Constraint Neural Network Regression Model Subject to Linear Priors. *IEEE Trans. on Neural Networks*, 22(12):2447–2459, 2011.

[159] Z. Sun, Z. Zhang, H. Wang, and M. Jiang. Cutting Plane Method for Continuously Constrained Kernel-Based Regression. *IEEE Trans. on Neural Networks*, 21(2):238–247, 2010.

[160] F. Han, T.-M. Lok, and M. Lyu. A New Learning Algorithm for Function Approximation Incorporating A Priori Information into Extreme Learning Machine. In *Proc. Int. Symp. on Neural Networks*, volume 3971, pages 631–636, 2006.

[161] X. Liu, P. Li, and C. Gao. Symmetric Extreme Learning Machine. *Neural Computing and Applications*, 22(3–4):551–558, 2013.

[162] T. F. Coleman and Y. Li. A Reflective Newton Method for Minimizing a Quadratic Function subject to Bounds on some of the Variables. *Journal on Optimization (SIAM)*, 6(4):1040–1058, 1996.

[163] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization*. Academic Press, 1981.

[164] G. Tevatia and S. Schaal. Inverse Kinematics for Humanoid Robots. In *Proc. Int. Conf. on Robotics and Automation*, pages 294–299, 2000.

[165] E. Oyama, A. Agah, K. F. MacDorman, T. Maeda, and S. Tachi. A Modular Neural Network Architecture for Inverse Kinematics Model Learning. *Neurocomputing*, 38:797–805, 2001.

[166] B. Bocsi, D. Nguyen-Tuong, L. Csato, B. Schölkopf, and J. Peters. Learning Inverse Kinematics with Structured Prediction. In *Proc. Int. Conf. on Intelligent Robots and Systems*, pages 698–703, 2011.

[167] J. Barhen, S. Gulati, and M. Zak. Neutral Learning of Constrained Nonlinear Transformations. *Computer*, 22(6):67–76, 1989.

[168] Y. Tomikawa and K. Nakayama. Approximating many Valued Mappings using a Recurrent Neural Network. In *Proc. Int. Joint Conf. on Neural Networks*, volume 2, pages 1494–1497, 1998.

[169] R. F. Reinhart and J. J. Steil. Neural Learning and Dynamical Selection of Redundant Solutions for Inverse Kinematic Control. In *Proc. Humanoids*, pages 564–569, 2011.

[170] M. W. Hannan and I. D. Walker. Kinematics and the Implementation of an Elephant's Trunk Manipulator and other Continuum Style Robots. *Journal of Robotic Systems*, 20(2):45–63, 2003.

[171] C. Laschi, B. Mazzolai, V. Mattoli, M. Cianchetti, and P. Dario. Design of a Biomimetic Robotic Octopus Arm. *Bioinspiration and Biomimetics*, 4(1):6–15, 2009.

[172] Deutscher Zukunftspreis (German Future-Award), 2010.

[173] K. J. Korane. Robot Imitates Nature. *Machine Design*, 82(18), 2010.

[174] A. Grzesiak, R. Becker, and A. Verl. The Bionic Handling Assistant - A Success Story of Additive Manufacturing. *Assembly Automation*, 31(4):329–333, 2011.

[175] H. Kay and L. H. Ungar. Estimating Monotonic Functions and their Bounds. *American Institute of Chemical Engineering Journal*, 46(12):2426–2434, 2000.

[176] M. Rolf, J. J. Steil, and M. Gienger. Bootstrapping Inverse Kinematics with Goal Babbling. In *Proc. IEEE Int. Conf. on Development and Learning*, pages 147–154, 2010.

[177] M. Rolf, J. J. Steil, and M. Gienger. Goal Babbling Permits Direct Learning of Inverse Kinematics. *IEEE Trans. Autonomous Mental Development*, 2(3):216–229, 2010.

[178] M. Rolf and J. J. Steil. Constant Curvature Continuum Kinematics as Fast Approximate Model for the Bionic Handling Assistant. In *IEEE Int. Conf. on Intelligent Robots and Systems*, pages 3440–3446, 2012.

[179] M. Rolf and J. J. Steil. Learning to Reach with Festo's Bionic Handling Assistant. http://www.youtube.com/watch?v=aGFiMviI0VY.

[180] M. Gharib, F. Pereira, D. Dabiri, and D. Modarress. Quantitative Flow Visualization. *Annals of the New York Academy of Sciences*, 972(1):1–9, 2002.

[181] A. Verri and T. Poggio. Motion Field and Optical Flow: Qualitative Properties. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11(5):490–498, 1989.

[182] F. A. Mussa-Ivaldi and S. F. Giszter. Vector Field Approximation: a Computational Paradigm for Motor Control and Learning. *Biological Cybernetics*, 67(6):491–500, 1992.

[183] A. Pistillo, S. Calinon, and D. G. Caldwell. Bilateral Physical Interaction with a Robot Manipulator through a Weighted Combination of Flow Fields. In *Proc. Int. Conf. on Intelligent Robotics and Systems*, pages 3047–3052, 2011.

[184] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. *Robot programming by demonstration*, volume 1. Springer, 2008.

[185] M. Mühlig, M. Gienger, and J. J. Steil. Interactive Imitation Learning of Object Movement Skills. *Autonomous Robots*, 32(2):97–114, 2012.

[186] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242, 1999.

[187] F. L. Moro, N. G. Tsagarakis, and D. G. Caldwell. On the Kinematic Motion Primitives (kMPs) - Theory and Application. *Frontiers in Neurorobotics*, 6(10), 2012.

[188] A. Ude, A. Gams, T. Asfour, and J. Morimoto. Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives. *Transactions on Robotics*, 26(5):800–815, 2010.

[189] S. M. Khansari-Zadeh and A. Billard. BM: an iterative algorithm to learn Stable Non-Linear Dynamical Systems with Gaussian Mixture Models. In *Proc. Int. Conf. on Robotics and Automation*, pages 2381–2388, 2010.

[190] A. Ijspeert, J. Nakanishi, and S. Schaal. Learning Attractor Landscapes for Learning Motor Primitives. In *Proc. Neural Information Processing Systems*, pages 1523–1530, 2003.

[191] E. Gribovskaya, S. M. Khansari-Zadeh, and A. Billard. Learning Non-Linear Multivariate Dynamics of Motion in Robotic Manipulators. *Int. Journal of Robotics Research*, 30(1):80–117, 2011.

[192] K. Dautenhahn and C. L. Nehaniv. The Agent-Based Perspective on Imitation. *Imitation in Animals and Artifacts*, pages 1–40, 2002.

[193] S. M. Khansari-Zadeh and A. Billard. Learning Stable Non-Linear Dynamical Systems with Gaussian Mixture Models. *Trans. on Robotics*, 27(5):943–957, 2011.

[194] Z. Artstein. Stabilization with Relaxed Controls. *Nonlinear Analysis TMA*, 7(11):1163–1173, 1983.

[195] E. D. Sontag. A Universal Construction of Artstein's Theorem on Nonlinear Stabilization. *Systems and Control Letters*, 13(2):117–123, 1989.

[196] P. Kokotović and M. Arcak. Constructive Nonlinear Control: a Historical Perspective. *Automatica*, 37(5):637–662, 2001.

[197] S. M. Khansari-Zadeh. *A Dynamical System-based Approach to Modeling Stable Robot Control Policies via Imitation Learning.* PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 2012.

[198] F. Reinhart, A. Lemme, and J. Steil. Representation and Generalization of Bi-manual Skills from Kinesthetic Teaching. In *Proc. Humanoids*, pages 560–567, 2012.

[199] N. G. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A. Ijspeert, and M. C. Carrozza. iCub: the Design and Realization of an Open Humanoid Platform for Cognitive and Neuroscience Research. *Advanced Robotics*, 21(10):1151–1175, 2007.

[200] S. M. Khansari-Zadeh. `http://www.amarsi-project.eu/open-source`, 2012.

[201] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms.* Wiley-Interscience, 2006.

[202] G. Sandini, G. Metta, and D. Vernon. The iCub Cognitive Humanoid Robot: an Open-System Research Platform for Enactive Cognition. *50 years of Artificial Intelligence*, pages 358–369, 2007.

[203] D. Siepe, R. Bayerer, and R. Roth. The Future of Wire Bonding is? Wire Bonding! In *Proc. Int. Conf. on Integrated Power Electronics Systems*, 2010.

[204] J. Gausemeier, F. J. Rammig, W. Schäfer, and W. Sextro. *Dependability of Self-Optimizing Mechatronic Systems.* Springer-Verlag, Berlin, Heidelberg, Germany, 2013.

[205] S. Hagenkötter, M. Brökelmann, and H.-J. Hesse. PiQC - a Process Integrated Quality Control for Nondestructive Evaluation of Ultrasonic Wire Bonds. In *Proc. IEEE Ultrasonics Symposium*, pages 402–405, 2008.

[206] H. Stürmann. Process Integrated Quality Control (PiQC). *EPP EUROPE Electronics Production and Test*, 11, 2007. Hesse & Knipps GmbH, Paderborn, Germany.

[207] Herbert Stürmann. Wire Bonding Process Control Targeting 100% Yield. *Advanced Packaging*, 17(1):32–35, 2008. Hesse & Knipps GmbH, Paderborn, Germany.

[208] K. Schrimper. Process Integrated Quality Control (PiQC) for Fully Automatic Wire Bonders. *EPP EUROPE Electronics Production and Test*, (11), 2008. Hesse & Knipps GmbH, Paderborn, Germany.

[209] R. Rodwell and D. A. Worrall. Quality Control in Ultrasonic Wire Bonding. *Microelectronics International*, 2(3):67–72, 1985.

[210] A. Siddiq and E. Ghassemieh. Thermomechanical Analyses of Ultrasonic Welding Process using Thermal and Acoustic Softening Effects. *Mechanics of Materials*, 40(12):982–1000, 2008.

[211] C. Zhang and L. Li. A Coupled Thermal-Mechanical Analysis of Ultrasonic Bonding Mechanism. *Metallurgical and Materials Transactions*, 40(2):196–207, 2009.

[212] C. Zhang. *A Thermomechanical Analysis of an Ultrasonic Bonding Mechanism.* PhD thesis, Utah State University, USA, 2011.

[213] C. Doumanidis and Y. Gao. Mechanical Modeling of Ultrasonic Welding. *Welding Research Journal*, 83:140–146, 2004.

[214] B. Langenecker, C. W. Fountain, and S. R. Colberg. Effects of ultrasound on deformation characteristics of structural metals. *Defense Technical Information Center*, 1965.