# Reservoir Computing with Output Feedback

René Felix Reinhart

Vorgelegt zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
Technische Fakultät, Universität Bielefeld
Oktober 2011

Gedruckt auf alterungsbeständigem Papier ♾ ISO 9706

# Abstract – Reservoir Computing with Output Feedback

**René Felix Reinhart**

A dynamical system approach to forward and inverse modeling is proposed. Forward and inverse models are trained in associative recurrent neural networks that are based on non-linear random projections. Feedback of estimated outputs into such reservoir networks is a key ingredient in the context of bidirectional association but entails the problem of error amplification. Robust training of reservoir networks with output feedback is achieved by a novel one-shot learning and regularization method for input-driven recurrent neural networks. It is shown that output feedback enables the implementation of ambiguous inverse models by means of multi-stable dynamics. The proposed methodology is applied to movement generation of robotic manipulators in a feedforward-feedback control framework.

**Keywords:** forward and inverse models, bidirectional association, recurrent neural networks, output feedback dynamics, regularization, stability

# Contents

Chapter 1

# Introduction

Association is a key principle of neural processing [1, 2, 3]. An associative connection between two entities is naturally bidirectional, and, as a process, denotes the recall of one entity from the other. Many connectionist models of associative memory have been developed [4, 5, 6, 7]. Despite their ability to learn from examples and their neuroscientific motivation, early computational models of associative memory have rather limited capabilities: Hopfield-type networks [4] can associate only a discrete number of patterns with each other and fail to generalize to continuous mappings. Then, associative computation remains a neural model of content addressable memory. In the context of bidirectional association, another challenge arises that is rarely discussed but important to the subject: If the forward relation is many-to-one, the reverse relation is ambiguous, i.e. one-to-many. In terms of continuous mappings, the forward mapping is not invertible.

The case of ambiguous inverse models is a common problem in many application areas ranging from control engineering to computer vision. For instance, recognizing objects and their spatial orientation is an one-to-many mapping due to loss of information when three-dimensional scenes are projected onto a two-dimensional retina. Kersten therefore coined the term "inverse graphics" for object recognition [8][1]. Hinton also emphasized the need for both, internal forward and inverse models, in [10] where he proposed "to recognize shapes, first learn to generate images". In linguistics and functional neuroanatomy of language, it has been stated that speech recognition is the inverse process of speech production [11, 12]: The listener estimates configurations of the vocal tract from auditory input, which is an ambiguous inverse relation. The requirement of solving forward and inverse problems is even more prominent in motor learning. It has been hypothesized that paired forward and inverse models are crucial for skilled movement generation [13]. In particular, human reaching performance in the absence of visual feedback and also delayed proprioceptive feedback strongly indicate the presence of internal forward and inverse models in the brain [14]. Redundancy of the movement apparatus further gains ambiguity of the inverse model. The ambiguity of one-to-many relations states a main problem in these contexts.

In principle, the problem of ambiguity can be approached following one of two paradigms: Either, ambiguous data samples are discarded and learning is restricted to a single solution [9, 15, 16, 17]. Or, ambiguity is represented by a model such that multiple solutions to the inverse problem coexist [18, 19, 20, 21]. On the one hand, restricting the inverse model to a single solution alters the original one-to-many relation to a simpler one-to-one mapping which can be learned with classical feed-forward networks. On the other hand, this restriction renders the inverse model incomplete in the sense that not all possible solutions are captured by the model. The restricted modeling of a single solution only is often not appropriate. For instance,

[1] Even earlier, Poggio *et al.* denoted early vision to be "inverse optics" [9].

consider inverse graphics where modeling of a single solution rejects possibly correct solutions.

The multiplicity of the inverse relation adds flexibility to a system which is often more meaningful than apriori selection of a single response. However, representing one-to-many relations entails additional problems like the selection of a particular solution during system exploitation. The commitment to a particular solution out of many, i.e. ambiguity resolution, is often accomplished by additional circuitry [18, 13]. A more natural approach to ambiguity resolution is to utilize additional information that is integrated over time. For example, consider following the motion of an object for a while to disentangle if it is moving towards or away from you. An internal representation then "stands in" [14] for the missing information from the sensory input which has been hypothesized to be a minimal requirement for a system to be cognitive. A simple example for the resolution of ambiguity by an internal representation is the Necker cube (see Fig. 1.1). One perceives either one face to the front or the other depending on which information the perceiver adds internally to the same sensory stimulus. Essentially, the combination of forward and inverse models, memory and learning by means of bidirectional association constitutes a key ingredient of cognition.

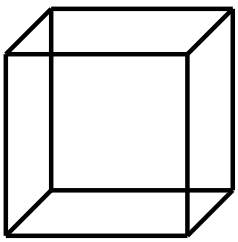


Fig. 1.1: The Necker cube admits ambiguous interpretation of its spatial configuration.

In this thesis, the associative principle is revived by an adaptive dynamical system approach that facilitates bidirectional and continuous association of ambiguous mappings. I combine and extend previous ideas to efficient training of connectionist models proposed in [22] and [23, 24] to achieve robust learning of bidirectional associative models. Inputs and outputs are treated as parts of a larger recurrent neural network with a hidden layer. Continuous association is modeled as an input-driven process, i.e. the system's representation is continuously biased by external inputs. Inputs or outputs are driven on demand in order to approximate the forward or inverse relation. The network dynamically inserts estimated values for the unspecified network parts in a recursive feedback loop. These feedback dynamics of estimated outputs can be multi-stable and thereby represent one-to-many relations. I formalize the concept of feeding back estimated outputs under the notion of output feedback dynamics. This concept unifies the ideas applied in a range of connectionist models, e.g. feed-forward networks with additional recurrent loops [25, 26, 20, 27, 28], fully recurrent neural networks [29] and particular flavors thereof [30, 22, 31, 23]. Ambiguity resolution by output feedback dynamics is based on the current system state that acts as short-term memory and integrates the internal representation over time.

An early and conceptually similar approach dates back to Barhen *et al.* [29]. They proposed an attractor-based scheme for learning inverse kinematics of redundant robots. In the outset of their method, multiple solutions of the inverse kinematics are represented by multi-stable attractor dynamics. Learning adapts all weights to imprint the training data into such attractor dynamics. The approach therefore suffers all the standard problems with complexity and efficiency of recurrent network training [32], and – while conceptually elegant – has not been demonstrated to be usable in practice for real robot applications.

Another related approach to represent multi-valued functions in a dynamical network has been proposed in [20]. Tomikawa and Nakayama utilize a combined representation of inputs and outputs in the hidden layer of a feed-forward network in order to resolve the ambiguity of multi-valued functions for learning. During system exploitation, estimated outputs are fed back iteratively into the network very similar to the model that is proposed in this thesis. However, additional constraints, in particular an integral condition, added to the learning objective renders training inefficient despite the feed-forward network structure. Moreover, weights of the additional constraints have to be balanced precisely [20], which makes learning sensitive to parameter configurations, and the model was only applied to toy examples.

I follow another thread of research that considers non-linear random projections networks. A basic variant of these networks was introduced under the notion of extreme learning machines [33]. Their spatio-temporal siblings comprise additionally random feedback connections between hidden processing units which provide a "rich reservoir" of dynamics [34]. These so called reservoir computing networks support efficient learning and obviate tuning of the internal representation. Bidirectional association in these reservoir networks is based on training a perceptron-like read-out layer which is recurrently connected to the reservoir. Training of the read-out connections imprints desired relations between inputs and outputs into the otherwise untrained network. In this thesis, associative reservoir computing networks with output feedback dynamics are trained to model forward and – possibly ambiguous – inverse relations. The success of the proposed approach requires the integration of learning and input-driven dynamical systems, which is particularly challenging with respect to stability of the output feedback dynamics. Feedback of erroneous outputs into the network can lead to error amplification which is a serious problem. Although output feedback dynamics and the associated stability problem occurs also in related connectionist models [25, 26, 20, 29, 27], stability of the output feedback dynamics is rarely discussed, e.g. in [35]. I formalize the concept of output feedback dynamics and the connected problem of output feedback stability in this thesis.

In the context of bidirectional association, another problem arises. Externally driven network parts change depending on which relation is currently queried, e.g. driving inputs while estimating outputs or vice versa. It is important to balance the contribution of inputs and outputs to the network such that the network state can equally well be driven by inputs or outputs. Otherwise, either the forward or inverse relation is modeled primarily and bidirectionality of the model is compromised.

I approach these crucial issues in the context of reservoir computing with output feedback in three steps. I first formalize the concept of output feedback dynamics and propose a tailored output feedback stability criterion. In a second step, I show that regularization of the learning and the internal representation mitigates the stability issue to a great extent. While regularization of the learning has been previously shown to support stability [36, 35], regularization is extended to the internal representation in this thesis. The main idea of this reservoir regularization is to implement desired network dynamics with small weights which reduces the gain of the system. For this purpose, I propose an efficient one shot learning method for input-driven recurrent neural networks that combines ideas from reservoir computing and associative memories. The reservoir regularization method increases the robustness of learning success by stabilizing output feedback dynamics. With the same methodology, also the problem of balancing contributions to the hidden layer is resolved in a single step. Stability of the output feedback dynamics and the balancing of contributions are necessary prerequisites for the application of reservoir networks with output feedback to bidirectional association.

With these concepts and methods at hand, I tackle the problem of ambiguity in a third step. I first show that ambiguity can be resolved by output feedback dynamics. I then adopt the developed one-shot learning methodology to shape multi-stable output feedback dynamics of the recurrent network model. Multi-stable output feedback dynamics represent one-to-many relations and disentangle ambiguous sensory stimuli by means of a short-term memory. I evaluate properties of the multi-stable attractor dynamics and demonstrate their application in different scenarios.

Finally, the gathered techniques are applied to movement generation, which is a salient example for forward and inverse modeling in the context of motor learning. In robotic experiments, I show that the proposed model for bidirectional association performs smooth movement generation in a feedforward-feedback control loop. The presented results emphasize the computational power of bidirectional association with dynamical systems. This thesis essentially contributes to the robust and efficient learning of bidirectional and ambiguous relations in dynamical connectionist models.

Chapter 2

# Associative models, inverse problems and ambiguity

In this chapter, mathematical prerequisites concerning association, continuous mappings and inverse problems are introduced. A brief review of associative models is given along the discussion and the methodology developed in this thesis is related to state-of-the-art approaches. At the end of this chapter, Tab. 2.1 gives a compact overview of the discussed associative methods.

## 2.1 Bidirectional association

Consider a set of data pairs $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1,\dots,N}$, where $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{y} \in \mathbb{R}^O$. *Association* denotes the process of recalling the corresponding output pattern $\mathbf{y}_n$ given a probably corrupted input pattern $\mathbf{x}_n + \nu$ [37, 7]. Association of patterns with themselves, i.e. $\mathbf{x}_n \equiv \mathbf{y}_n$, is called auto-association and hetero-association otherwise. More formally, association is a relation

$$\mathbf{f} : X \subset \mathbb{R}^D \to Y \subset \mathbb{R}^O$$

with a finite domain, $|X| < \infty$ (see Fig. 2.2 (left)). Training samples $(\mathbf{x}_n, \mathbf{y}_n)$ strictly define the associative relation $\mathbf{f}$ only for a finite set. In the recalling phase, inputs are generally drawn from the entire input space.

Standard Hebbian, correlation-based or pseudo-inverse learning can be employed to form linear transformations from the input space to the output space such that output patterns can be "retained" from an input key by a simple matrix multiplication [38, 7, 37]. These *associative memory* (first row in Tab. 2.1) models store the output data in their model parameters in a distributed fashion [38, 37, 7]: The input data and the fitted model are sufficient to recall $\mathbf{y}_n$ from $\mathbf{x}_n$ for $n = 1, \dots, N$. In case of correlation-based learning and orthogonal input patterns, all output patterns can be correctly recalled [38, 7]. For pseudo-inverse learning, linear independence of the input patterns is sufficient to achieve perfect recall [37].

The scheme of associative recall has been generalized to non-linear and dynamic associative memory models. The underlying principle of dynamical associative memories, such as the auto-associative Hopfield network [4] and its hetero-associative generalization [5], is the convergence of feedback dynamics to a fixed-point depending on the initial condition. Learning is supposed to shape the dynamics such that training patterns reside in the center of attractor basins. Iterating the dynamics from any initial condition in the attractor basin of a training pattern results in the recall of this pattern. I.e. dynamical associative memories robustly recall stored patterns from noisy keys which is often emphasized to be a desirable feature [37, 4, 5]. However, this paradigm drastically affects the kind of computation that can be accomplished: Association of discrete patterns enables robust recall in the presence of noise. But the gained robustness of

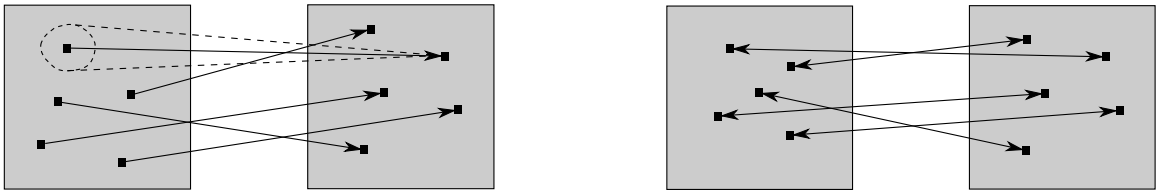


Fig. 2.2: Association: Mapping a finite domain to its codomain (left). Bidirectional association additionally maps the finite codomain back to its domain. Injective mapping shown.

recalling a finite set of patterns limits generalization of network responses to novel inputs. Through the glasses of function approximation, these associative models implement a piecewise constant, "quantized" [39] function with discontinuities at the borders between attractor basins (see Fig. 2.1). Discontinuities are typically due to saddle nodes of the dynamics, i.e. instable fixed-points, at the border of two attractor basins (circles in Fig. 2.1). The associative memory function $\mathbf{f}$ is queried at novel inputs and recalls the associated training patterns in the output space. Generalization by means of inter- or extrapolation is therefore restricted. Typical applications of discrete association are content addressable memories and pattern recognition. Also, sequence generation can be implemented by auto-association of patterns at successive time steps, i.e. $\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k))$ [37, 4, 5].

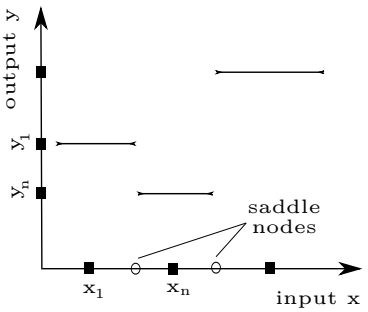



Fig. 2.1: Associative memories are piecewise constant, discontinuous functions.

Given a set of data pairs $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1,\ldots,N}$, there is apriori no meaningful difference in associating inputs from outputs or vice versa. Kosko therefore proposed to bidirectionally associate "inputs" with "outputs" [5]. His *bidirectional associative memory* (BAM, second row in Tab. 2.1) subsumes the Hopfield network as a special case and can be further generalized to multi-directional associative memories [40, 41]. The bidirectional association of finite sets is illustrated in Fig. 2.2 (right).

Most of the discussed associative memory models can be generalized to the case of multi-directional association. An early approach to multi-directional association, the Associatron [42] introduced by Nakano in 1972, comprises an internal representation in addition to input and outputs. Nakano formulated the concept of associative memories with multiple "neural areas" [42] to increase the memory capacity and resolve the problem of cross-talk between patterns. In this thesis, the coupling of multiple representations by an internal representation is also pursued. I focus on the case of input, internal and output "fields" only, which is sufficient for the discussion and does not impair the generality of the approach.

## 2.2 Forward and inverse models

In many cases, there is a continuous regularity between inputs and outputs which shall be extracted by learning of a parameterized model. Continuity means that a small change in input space corresponds to a small change in the output space, i.e. $\mathbf{f}(\mathbf{x}_i) \approx \mathbf{f}(\mathbf{x}_j)$ if $\mathbf{x}_i \approx \mathbf{x}_j$ (see Fig. 2.4). Approximating a *continuous mapping* $\mathbf{f} : X \to Y$ with $|X| = \infty$ by a parameterized model then enables to generalize the relation of inputs and outputs in the data to novel inputs that are not part of the training data.

Similar to the generalization of associative memories to bidirectional association, modeling the forward mapping $\mathbf{f} : \mathbf{x} \mapsto \mathbf{y}$ is not different from modeling the inverse mapping $\mathbf{g} : \mathbf{y} \mapsto \mathbf{x}$ under certain conditions. Formally, $\mathbf{g}$ is a right inverse of $\mathbf{f}$ if $(\mathbf{f} \circ \mathbf{g})(\mathbf{y}) = \mathbf{y} \quad \forall \mathbf{y} \in Y$. Note that a right inverse $\mathbf{g}$ may map outputs $\mathbf{y}$ to any $\mathbf{x}$ with $\mathbf{f}(\mathbf{x}) = \mathbf{y}$. That is, the right inverse

Fig. 2.4: Continuous and invertible mapping mapping (left). Bidirectional mapping with multi-valued and continuous inverse (right).

is not necessarily unique. The inverse is unique if $\mathbf{g}$ is also the left inverse of $\mathbf{f}$ such that $(\mathbf{g} \circ \mathbf{f})(\mathbf{x}) = \mathbf{x} \quad \forall \mathbf{x} \in X$ holds for the reverse composition. It is useful to distinguish different properties of the forward mapping in order to determine uniqueness or ambiguity of the inverse: If $\mathbf{f}$ is a homeomorphism, i.e. $\mathbf{f}$ is a continuous and bijective mapping, the forward mapping is *invertible*. Then, the forward and inverse mappings are both unique and continuous (sometimes referred to as one-to-one, see Fig. 2.4 (left)).

In many cases, however, the forward mapping is not injective and thus not invertible. This is always the case if there exist several causes that result in the same effect, e.g. several joint angles can result in the same end effector position. Then, the inverse relation is *ambiguous* (also referred to as one-to-many). Formally, there exist several functions $\mathbf{g}_b$ with $\mathbf{f}(\mathbf{g}_b(\mathbf{y})) = \mathbf{y}$ that are defined on a subset of the codomain of $\mathbf{f}$ and are called *solution branches*. Typically, there is a *principle branch* $\mathbf{g}$ that is defined on the entire codomain of $\mathbf{f}$ (see Fig. 2.3). Note that continuity may apply to each of the solution branches setting up multiple smooth solution manifolds (see Fig. 2.4 (right)). Solving this *inverse problem*, i.e. providing one of many suitable inputs to obtain desired outputs, is important in various application domains ranging from control engineering [15] to pattern recognition [9, 8].

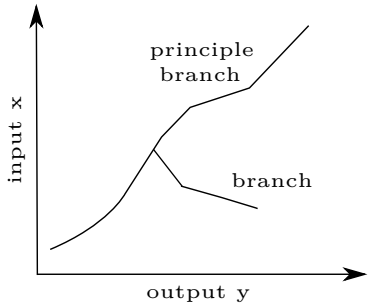



Fig. 2.3: Forward and inverse function with multiple solution branches.

From a set-theoretic viewpoint, the inverse of a non-injective function is a multi-valued function $\mathbf{f} : \mathbf{y} \mapsto \{\mathbf{x}_b\}_{b=1,\ldots,B}$. In Fig. 2.5 (left) the multi-valued relation between outputs and inputs that stems from a non-injective forward model is illustrated using sets. Typically, the set of solutions for all outputs $\mathbf{y}$ is non-convex unless the forward mapping is not constant for a range of inputs $\mathbf{x}$. In other words, the set of solutions $|\{\mathbf{x}_b\}| < \infty$ is discrete and thus forms a non-convex subset in the continuous input space $X$. Constant forward mappings have infinitely many inverse solutions. That is, the multi-valued inverse function has an infinite codomain ($|\{\mathbf{x}_b\}| = \infty$). This set of solution can in addition be non-convex, e.g. if two spatially separated regions of the input space map to the same output value. In general, there are complex relations between sets in both spaces, including overlaps and inclusions. A fork of branches, for instance, translates into "adjacent" sets, because the point of bifurcation belongs to both branches (or to the principal branch only depending on the definition). Fig. 2.5 (right) and Fig. 2.3 finally show that cases exist where both, the forward and the inverse relation can be ambiguous. Note that noisy data in practice compromise the uniqueness of the forward mapping: Probing outputs for the same input several times yields to various observations due to noise and thus the forward mapping is, strictly speaking, not one-to-one even though the function underlying the data generation process is an one-to-one mapping.

Although there is apriori no preferred direction of mapping "inputs" to "outputs" or vice versa, the process underlying the generation of inputs and outputs is often causal. For instance, consider the causal relationship between joint angles and the resulting end effector position

Fig. 2.5: Multi-valued inverse relation (left): Multiple values of the inverse are due to a non-injective forward mapping. Also both, the forward and inverse relation, can be multi-valued (right). Ambiguous relations are indicated by dashed arrows.

of a manipulator. Causality in physical systems induces directionality of the relation between inputs and outputs. Then, an approximation of $\mathbf{f}$ is typically referred to as *forward model* which maps causes to effects. Modeling $\mathbf{g}$ is denoted as *inverse modeling* accordingly. In this thesis, adaptive bidirectional mappings by means of combined forward and inverse models are the core application theme.

The terminology above also generalizes to the case of time-series data. Temporally ordered data samples, i.e. time-series, are indicated by sample indices $k$ in parentheses, $\mathbf{x}(k)$. In this thesis, smooth trajectories, i.e. $\mathbf{x}(k+1) \approx \mathbf{x}(k)$, and a continuous forward mapping are typically considered. For instance, consider data to be collected along the trajectories shown in Fig. 2.4. The mapping of sequences $\mathbf{x}(k)$ to $\mathbf{y}(k)$ is generally denoted by *sequence transduction*. Analogously, bidirectional sequence transduction is the combined forward and inverse mapping of time-series.

## 2.3 Inverse problems and ambiguity resolution

In case of a non-injective relation between inputs and outputs, there is no unique solution to the inverse problem. For instance, in the context of control engineering multiple system inputs can cause the same system state and thus there are multiple solution branches to the inverse problem. How can the problem of ambiguity be tackled? How can multiple solutions be represented, and how to select a solution, i.e. how to resolve ambiguities?

From a machine learning perspective, multiple solutions to an inverse problem constitute a serious problem for traditional feed-forward function approximators. Consider ambiguous data pairs $(\mathbf{x}_n, \mathbf{y}_n)$ and $(\mathbf{x}_m, \mathbf{y}_m)$ with $\mathbf{x}_n = \mathbf{x}_m$ and $\mathbf{y}_n \neq \mathbf{y}_m$ for $n \neq m$. Ambiguous data pairs with the same or similar input but distinct outputs result in averaged approximations: Any feed-forward model is a function $\mathbf{f} : \mathbf{x} \mapsto \mathbf{y}$ that maps inputs to unique outputs. Parameter estimation from ambiguous data pairs fails due to the non-convexity of many inverse problems, i.e. typically the least squares solution $\frac{1}{2}(\mathbf{y}_n + \mathbf{y}_m)$ does not necessarily reside in the non-convex solution set (compare Fig. 2.3 and the discussion in [15, 26]). Note that the problem is due to the fact that the input representation is the same for both outputs and thus does not allow to discriminate between solutions. Traditional feed-forward architectures can therefore not represent multiple solutions to a mapping without additional knowledge.

In the following, main learning approaches that tackle the inverse problem are discussed.

### 2.3.1 Restricted inverse modeling

The first approach restricts modeling to a single solution by assuming that the training data comprises samples of only a particular, e.g. the principle, solution branch of the inverse problem. This restriction converts the original ambiguous problem into a unique mapping task. Solving the inverse problem is shifted to the data collection process. It is, however, not trivial to decide if data stems from a single solution to the inverse problem or how to obtain data for a single solution only. This problem has been approached from different directions.

In [15], a distal teacher formulation for a composite setup comprising a serial combination of an inverse and forward model has been proposed. Training the inverse model is based on gradient descent with respect to the performance error mediated through the forward model. The performance error $\mathbf{f}(\mathbf{g}(\mathbf{y})) - \mathbf{y}$ is evaluated in target space, e.g. $\mathbf{y}$ is the desired end effector position of a robot arm. Thereby, a particular solution to the inverse problem is found and the non-convexity problem is circumvented (see [15] for a decent discussion). Selection of a particular solution depends on initial conditions, exploration of control commands and can be biased by introducing additional costs to the error function.

In [43, 44], the non-convexity problem is solved by utilizing a strong bias of the model, i.e. a topological arrangement of neurons in a self-organizing map, which results in selection of a single solution branch of the inverse problem. The original variant of *self-organizing maps* (SOM, third row in Tab. 2.1) is closely related to vector quantization and the local representation of inputs in the model restricts its generalization abilities. This drawback is cured by the "continuous generalization" of the self-organizing map [16]. The *parameterized self-organizing map* (PSOM, fourth row in Tab. 2.1) approximates a continuous manifold by utilizing basis functions that are *parameterized* by the underlying grid of local prototypes. A review of self-organizing maps and in particular their application to modeling inverse problems can be found in [39].

A goal-directed inverse modeling approach is taken in [17] which solves the non-convexity problem during control space exploration. "Goal babbling" [17] utilizes the structure of the goal-directed exploration process to weight collected data samples in order to resolve ambiguities of the inverse problem. Also this approach alleviates the non-convexity problem for the learner, which therefore can be any feed-forward function approximator.

Strong biases either of the model or by means of constrained learning [9, 26, 45] are typical approaches to cope with ill-posed inverse problems. However, a main drawback of restricting the inverse model to a single solution branch is that the inverse model is not complete. That is, the inverse model provides only a subset of the multi-valued inverse function. Moreover, the selected solution is either depending on initial conditions or is biased by incorporating prior knowledge.

### 2.3.2 Combining multiple experts

A second approach to deal with ambiguous inverse problems is to model each solution manifold separately, i.e. learning a set of expert models [18]. Solution selection during system exploitation then reduces to selection of an expert model that models a specific branch of the inverse.

This modular approach can be understood as generalization of the previous approach to select a specific solution to the inverse problem per model and hence inherits some of its difficulties: It is difficult to assign data samples to models such that each expert is trained with data from a single solution branch [18]. It is also apriori unclear how many expert models are needed to model the complete inverse problem, i.e. how many solution manifolds exist. And finally, an external gating or responsibility model has to be trained that selects an expert depending on the current control task [18, 13, 46, 47]. Though arbitrary criteria could be implemented here independently of the inverse models, it is an additional burden to design such a gating. Moreover, a discrete set of experts can not model a continuous set of solutions, which occurs always if the forward mapping is constant in a region of the input space.

A variant of the expert model approach represents multiple solutions in a single model with additional inputs [21]. The additional inputs parameterize the learner's representation, alleviate the non-convexity problem during learning, and serve as selection mechanism during exploitation [48, 15]. This approach inherits the same drawbacks as the expert modeling approach and also requires supervised assignment of data pairs to particular input parameter configurations.

### 2.3.3 Recover invertibility by augmenting the inverse problem

Another approach tackles ambiguous inverse problems by adding augmented variables to the input data which resolve the ambiguity. The augmented variables act as additional constraints that select/discard particular solutions in the set of inverse solution branches. This approach is related to the previously discussed approach to parameterize the inverse model for the selection of sub-models. It is nevertheless useful to distinguish parameterizing a model in a rather binary fashion to select expert models from introducing augmented variables that resolve ambiguity by means of intrinsic properties of the inverse function. In the latter case, ambiguities are resolved by means of additional, typically domain-specific, constraints to the inverse model.

For instance, the model presented in a series of papers [49, 50, 51] implements feedback dynamics based on the *mean value of multiple computations* (MMC, fifth row in Tab. 2.1). In MMC networks, redundant sets of equations are provided by the designer which represent partial aspects of the modeled system. Ambiguity is resolved by augmenting the core relation between inputs and outputs with additional equations that, for example, express some constraints. During network exploitation, estimated variables are averaged over all redundant equations and then again fed back into the equations. These MMC dynamics relax to a particular solution even if the input does not fully constrain the solution, i.e. the approach copes with infinitely many solutions to the inverse problem.

In [16], a parameterized self-organized map was trained on augmented data which then allowed to resolve the ambiguous inverse kinematics of a simulated three link, four degrees of freedom robot arm. The augmented variables express specific constraints of the inverse model, e.g. coupling between joint angles or the elevation of a particular link, that can be weighted flexibly during model exploitation. In an expert model approach, this is not possible without knowledge about the particular properties of each expert. However, defining augmented variables requires even more complex domain knowledge of the inverse problem than assigning solution branches to expert models. Also, the selection of a particular solution during exploitation by means of selecting values for the augmented variables is left to additional mechanisms in this scheme.

### 2.3.4 Recover invertibility by incorporating temporal context

A fourth approach exploits temporal context to decide between ambiguous control schemes. For instance, originally instantaneous inverse problems are turned into a spatio-temporal mapping. But there are also inverse problems that naturally reside in the temporal domain. The resolution of ambiguities is then based on the history of inputs, i.e. memory. Memory in connectionist models is typically modeled by time delay lines [25] or transient-based short-term memories [52]. This scheme of ambiguity resolution is only meaningful in case of temporally correlated data and can be understood as temporal variant of the augmented variables approach. In deed, traditional neural time-series classification approaches consider learning as internal representation pursuit [25] and face the problem of typically ambiguous, non-discriminative input samples with $\mathbf{x}(k_1) = \mathbf{x}(k_2)$ but $\mathbf{y}(k_1) \neq \mathbf{y}(k_2)$ for some $k_1 \neq k_2$.

In this thesis, I also consider ambiguous time-series data pairs but which stem from a static relation between inputs and outputs. That means the data is temporally ordered but the underlying relation is instantaneous and requires no memory of the history: $\mathbf{y}(k) = \mathbf{f}(\mathbf{x}(k))$. For instance, let $\mathbf{x}(k)$ be the joint angle configuration of a manipulator and $\mathbf{y}(k)$ the end effector position. Then, using augmented variables that encode the temporal history has the drawback that different temporal dynamics of the inputs, i.e. speeds, disturb the spatio-temporal representation and thus can deteriorate the inverse model. Normalization of the data, e.g. by temporal warping of input sequences, or learning from rich data sets comprising several sequences with exemplary speeds becomes necessary and introduces additional costs. I show in Chapter 7 that an attractor-based approach to memory circumvents these problems.

### 2.3.5 Modeling inverse problems with probabilistic methods

Probabilistic methods have been applied to solve ambiguous inverse problems, e.g. [53]. Basically, the non-convexity problem is approached by modeling the joint distribution $P(\mathbf{x}, \mathbf{y})$ of inputs and outputs. Then, given a particular input $\mathbf{x}$, the conditional distribution $P(\mathbf{y}|\mathbf{x})$ provides a possibly multi-modal distribution of outputs. Also direct models of the conditional density $P(\mathbf{y}|\mathbf{x})$ can be learned which circumvents modeling the joint distribution, e.g. [54].

However, density estimation in high-dimensional spaces is notoriously hard and requires many training samples which are typically costly to collect. Moreover, the problem of explicitly selecting a particular output, i.e. a solution, from a multi-modal distribution remains. A maximum a-posteriori estimate is a good choice in general, but the value that maximizes the posterior could flip between two modes for small changes of the input. Also in the case of infinitely many solutions that cause a plateau of the posterior distribution, it remains an open question how to select a particular solution.

### 2.3.6 Dynamical system approach to inverse modeling

In this thesis, another approach is pursued that combines a joint representation of inputs and outputs with a dynamical ambiguity resolution scheme. The non-convexity of ambiguous inverse problems is tackled by utilizing a joint representation $\mathbf{h}(\mathbf{x}, \mathbf{y})$ of inputs and outputs in a connectionist model. I.e. an internal representation $\mathbf{h}(\mathbf{x}, \mathbf{y})$ is formed that is uniquely defined by $\mathbf{x}$ and $\mathbf{y}$ (compare Fig. 2.7 (left)). Learning a mapping from $\mathbf{h}(\mathbf{x}, \mathbf{y})$ to either $\mathbf{x}$ or $\mathbf{y}$ is then unique and does not suffer from the non-convexity problem because $\mathbf{h}(\mathbf{x}_n, \mathbf{y}_n) \neq \mathbf{h}(\mathbf{x}_m, \mathbf{y}_m)$ also if $\mathbf{x}_n = \mathbf{x}_m$ and only $\mathbf{y}_n \neq \mathbf{y}_m$ for $n \neq m$.

During network exploitation, typically only the input $\mathbf{x}$ is available and an appropriate output $\mathbf{y}$ shall be estimated by the model. Thus, the joint representation can not be calculated directly. The idea is to iteratively approach the joint representation $\mathbf{h}(\mathbf{x}, \mathbf{y})$ given $\mathbf{x}$ only by means of an input-driven, dynamical process using the current estimate of $\mathbf{y}(k)$ at iteration step $k$. Based on the current internal representation $\mathbf{h}(\mathbf{x}, \mathbf{y}(k))$, the output $\mathbf{y}(k+1)$ at the next time step is computed which is then recursively fed back into the model (compare Fig. 2.7 (right)). Learning in this framework reduces to assuring that $\mathbf{y}(k+1)$ is closer to the desired output $\mathbf{y}$ than $\mathbf{y}(k)$. Then, these *output feedback dynamics* give a slightly improved estimate $\mathbf{y}(k+1)$ and eventually converge to $\mathbf{y}$. That is, the output $\mathbf{y}$ is associatively recalled from input $\mathbf{x}$ by approaching an attractor of the possibly multi-stable output feedback dynamics. The driving input $\mathbf{x}$ continuously parameterizes the points of attraction of the output feedback dynamics. This enables the approximation of continuous mappings by means of parameterized attractor manifolds (see Fig. 2.6) which is fundamentally different from the piece-wise constant functions approximated by traditional associative memories.

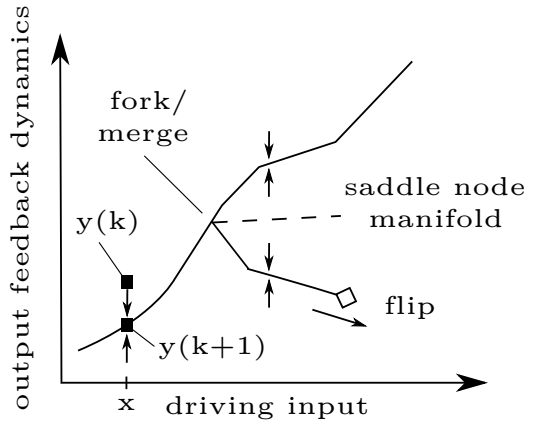



Fig. 2.6: Dynamical system approach to inverse modeling: Multiple attractor manifolds with typical bifurcations.

Output feedback dynamics can be multi-stable [20, 31, 55], i.e. have several points of attraction for each input $\mathbf{x}$. Solution branches of an ambiguous inverse problem can therefore be represented by multiple attractor manifolds (see Fig. 2.6). Fig. 2.6 further shows that solution branches are separated by saddle-nodes of the output feedback dynamics (provided that the output is a smooth function of the internal representation). Solution branches can smoothly merge/fork which spells out as a smooth bifurcation from uni- to bi-stable dynamics

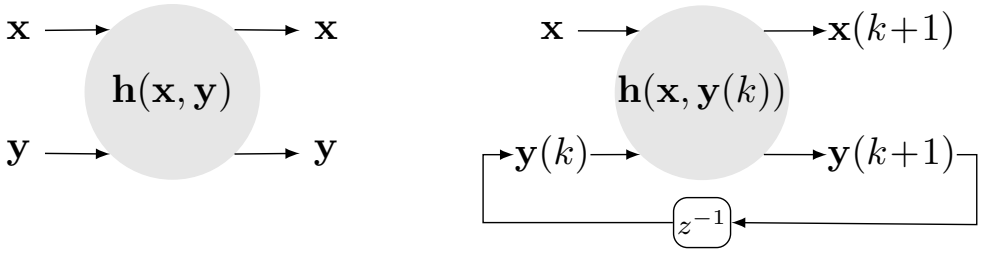



Fig. 2.7: A combined representation $\mathbf{h}(\mathbf{x}, \mathbf{y})$ of inputs and outputs resolves ambiguities (left). Dynamical recall of outputs (right): The representation is driven by external inputs $\mathbf{x}$ while iteration of output feedback dynamics recovers the output.

in the dynamical inverse model. Or, abrupt bifurcations can occur if the border of a solution branch is exceeded and this branch does not merge with another branch (see Fig. 2.6). In terms of set theory, a flip-node bifurcation occurs when leaving the domain of an inverse branch (compare Fig. 2.5). Note that bifurcations are driven by changes of the input $\mathbf{x}$ to the system. A main advantage of the dynamical approach to inverse modeling is that training does not require supervised information about the number of solution manifolds apparent in the data, nor an explicit assignment of data points to a certain solution. Training of multi-stable output feedback dynamics to represent ambiguous inverse models is demonstrated in Chapter 7.

In the dynamical approach to ambiguity representation and resolution, the case of infinitely many solutions translates to the concept of continuous attractor dynamics. A unique solution to the inverse problem corresponds to a globally stable fixed-point attractor (see Fig. 2.8 (left)), where the point of attraction is parameterized by the input to the inverse model. Two solutions to the inverse problem relate to bi-stable attractor dynamics (see Fig. 2.8 (middle)). The input to the inverse model may drive the output feedback dynamics through a saddle node bifurcation: A single fixed-point attractor splits into two fixed-points separated by a saddle node (from Fig. 2.8 (left) to (middle)). These two fixed-points then manifest a two-valued solution set of the inverse model for a particular input. Continuous attractor dynamics model the limit case of infinitely many solutions by means of a continuous attractor manifold for a single input (see Fig. 2.8 (right)). Continuous attractor dynamics can be understood as an evenly leveled valley in the system's associated energy function. Descending the energy landscape by iterating the dynamics yields a continuum of attractor states in the energy valley which are approached depending on the initial condition. In [56, 57], a model of continuous attractor dynamics is proposed to explain oculomotor control. I demonstrate continuous attractor manifolds of output feedback dynamics in Chapter 6 and Chapter 7.

In [29] and [58], a conceptually similar approach to represent the inverse kinematics of robotic manipulators by multi-stable terminal attractor dynamics has been proposed by Barhen, Gulati and Zak (sixth row in Tab. 2.1). However, it was not shown that their model is able to learn from ambiguous data or resolves ambiguities dynamically during operation. The network further does not model both, the forward and inverse model, and suffers from high computational load and bifurcations during learning. The implementation of output feedback dynamics to approximate multi-valued functions has been presented in [20]. However, the model also suffers from inefficient training and does not comprise a forward model.

The model proposed in this thesis learns the forward and inverse model efficiently and in a single network. The additional forward model is particularly useful in combination with feedback information from a plant: The network then implements a feedback control mechanism that estimates the current system state simultaneously to computing control commands. Moreover, integration of feedback from the plant renders control responsive to external perturbations. The proposed model resembles aspects of internal forward and inverse models that are also

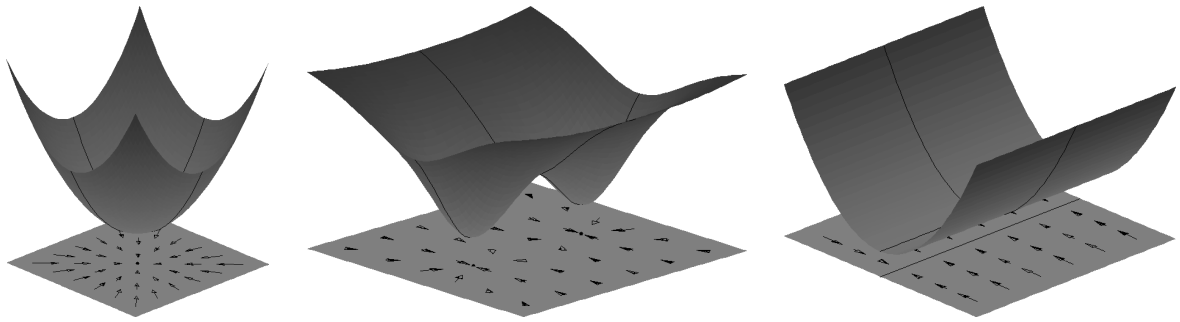


Fig. 2.8: Energy landscape of output feedback dynamics. Convex energy landscape with a single fixed-point attractor (left): The same attractor is approached for all initial conditions. Energy landscape with two minima correspond to bi-stable dynamics (middle): One of two attractors is chosen depending on the initial condition. A saddle node exists in between both attractor basins. Energy landscape with a level valley exhibits continuous attractor dynamics (right): Depending on the initial condition, one of the infinitely many solutions (black line in the plane) is selected.

hypothesized to be implemented in the brain for motor control [13].

The dynamical approach to model ambiguous inverse problems crucially depends on the ability to shape multi-stable attractor dynamics that reflect the multiplicity of solutions. An efficient learning mechanism that enables shaping of attractor dynamics on demand is introduced in Chapter 4. The method from Chapter 4 is extended to a regularization framework which facilitates the robustness of training output feedback dynamics as is shown in Chapter 5. Several desirable properties of dynamical systems for inverse modeling will be provided in Chapter 7.

## 2.4 Flexible selection of models by associative completion

The associative paradigm to model forward and inverse relations can be further generalized to the flexible selection of mappings between the input and output spaces. To facilitate the further discussion, we concatenate data pairs to single vectors, i.e. $\mathbf{u} = (\mathbf{x}, \mathbf{y})$. *Associative completion* means that a partially known pattern $\mathbf{u}$ is completed by an (auto-)associative process [59, 60, 39]. For example, consider $\mathbf{u} \in \mathbb{R}^3$ with the first and the last component known only. Associative completion returns the second component:

$$\begin{pmatrix} u_1 \\ \times \\ u_3 \end{pmatrix} \rightarrow \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$$

The core feature of associative completion is the flexible selection of the forward, inverse or a mixture of both models. That means any subset of components of $\mathbf{u}$ can be selected to query the model which then completes the complementary dimensions. Each constellation of inputs and outputs can be understood as a particular mapping, and there is apriori no preferred selection of inputs and outputs. However, incorporating the discussion above, it is important to distinguish the cases of unique and ambiguous mappings also in the context of associative completion. For instance, releasing an external input from the forward model can render the relation between selected inputs and outputs ambiguous. Note that associative completion differs from the approach to model each possible mapping between input and output space separately. Associative completion means that a single model is trained to represent the entire relation between input and output space. A particular model is then selected between parts of inputs and outputs during model exploitation.

Associative completion has been successfully implemented in non-dynamical associative models, for example by (parameterized) self-organizing maps using a flexible distance measure

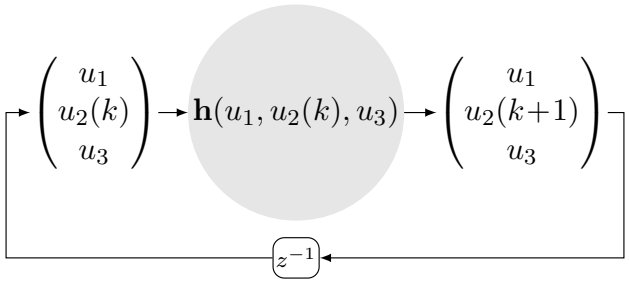



Fig. 2.9: Associative completion implemented by output feedback dynamics: The hidden representation is externally driven by components $u_1$ and $u_3$. The complementary components, in this case $u_2$, are recalled dynamically by iterating the output feedback loop.

[59, 16, 39]. The dynamical implementation of associative completion is much less considered in the literature, although the basic concepts can be already found in [37] and [42]. Opposed to the recall of a stored pattern in traditional associative memories where the memory is retrieved from a completely determined initial condition, associative completion is an input-driven process. In case of dynamical associative memories, the processing units are clamped to, or *driven by*, the known part of the pattern throughout the recall process. Fig. 2.9 illustrates the implementation of associative completion by output feedback dynamics.

Seung proposes an associative completion mechanism that is implemented by an auto-encoder network ([27], seventh row in Tab. 2.1). A similar configuration is commonly applied in stacked auto-encoder models, where the feature representation of the input is concatenated with supervised pattern information, e.g. class labels, in the topmost layer. Label information is then recalled by a feedback process which is driven by the current input representation. A structurally similar process, i.e. probabilistic inference, is applied in Deep Belief Networks [61].

The dynamical or recurrent implementation of associative completion in case of static input data has not been explicitly discussed in the literature other than in [27], even though Jordan-type recurrent neural networks [15] comprise the same feedback feedback loop constellation and principally enable associative completion. This might be due to the typical application of Jordan (and Elman) networks to time-series prediction than to the association of static patterns [15, 25].

This thesis focuses on dynamical associative completion implemented in *associative reservoir computing* (ARC, eighth row in Tab. 2.1) networks. The associative reservoir computing framework developed in Chapter 3 unifies a broad range of reservoir models and comprises output feedback dynamics with the modeling power outlined in Sec. 2.3.6. Output feedback dynamics moreover generalize the dynamical implementation of bidirectional mappings to associative completion (compare Fig. 2.7 and Fig. 2.9). I show that output feedback dynamics of associative reservoir networks can be trained efficiently to model forward and inverse mappings including multiple solutions, associative completion and sequence transduction (Chapter 6 and Chapter 7). Finally, output feedback dynamics of associative reservoir networks are exploited for movement generation in feed-forward as well as feedback control systems (Chapter 8).

Tab. 2.1: Taxonomy of associative models. The methods are related to each other with respect to their representation, dynamics, computational capabilities, and typical learning algorithms. Note that this is a rather broad overview and thus only some of the most prominent models for each conceptual niche are cited and described.

| Method | Representation | Dynamics | Computation | Learning | Citations |
|---|---|---|---|---|---|
| **Associative Memory** | • distributed | • none | • auto- and hetero-association | • correlation-based<br>• pseudo-inverse | [38], [7], [37] |
| **BAM** | • distributed<br>(• internal representation) | • attractor | • bidirectional association | • correlation-based | [5], [4], [42], [40], [41] |
| **SOM** | • local<br>• prototypes plus topology | • none/winner takes all<br>(• lateral inhibition) | • dimension reduction<br>• vector quantization<br>• discrete associative completion | • unsupervised vector quantization with topological constraint | [62], [63], [39] |
| **PSOM** | • hybrid between global and local<br>• prototypes plus topology and basis functions | • winner takes all plus continuous optimization | • forward and inverse mappings<br>• continuous associative completion | • unsupervised vector quantization with topological constraint or supervised initialization | [59], [60], [16] |
| **MMC** | • distributed<br>• augmented variables | • attractor | • ambiguous inverse mappings | • none | [49], [50], [51] |
| **Barhen** | • distributed<br>• internal representation | • terminal attractor | • ambiguous inverse mappings | • generalization of backpropagation | [29], [58] |
| **Autoencoder with Associative Completion** | • distributed<br>• internal representation | • (continuous) attractor<br>• variants with internal dynamics | • auto-association<br>• continuous associative completion | • backpropagation<br>• contrastive divergence<br>• error correction | [27], ([64], [65], [66]) |
| **Associative Reservoir Computing** | • distributed<br>• internal representation | • transients<br>• (multi-stable) attractor<br>• internal and output feedback dynamics | • sequence transduction/generation<br>• forward and (ambiguous) inverse mappings<br>• continuous associative completion | • regression<br>• online least squares<br>• intrinsic plasticity<br>• state prediction | [22], [23], [67], [24], [55] |

Chapter 3

# Reservoir Computing with output feedback

In this chapter, the basic idea of reservoir computing is introduced. Then, the associative reservoir computing model with output feedback is presented. Based on the associative reservoir computing framework, representative reservoir models are reviewed. The concepts of output feedback dynamics and stability are formalized.

## 3.1 The Reservoir Computing Paradigm

Learning in recurrent neural networks with hidden representations is typically implemented by gradient descent methods [68, 69]. Due to the recurrent network structure, optimizing the network parameters in closed form is not feasible in a direct way. Beside the enormous computational load of classical learning rules for recurrent networks, e.g. backpropagation through time, real time recurrent learning and Atiya-Parlos learning (see [68, 69] for an overview), bifurcation of the network dynamics during iterative parameter adaptation do not allow guaranteed convergence to a local minimum of the error function.

In the recent decade, a different approach to training input-driven recurrent neural networks networks has been developed which considers the recurrent network as a fixed feature generating device that transforms the input signal into a spatio-temporal representation in the network state. This leads to the idea of separating a "harvesting" phase to record the states of the input-driven but otherwise fixed recurrent dynamics, and the output learning, which can be simple, linear, and efficient once the states are harvested. This approach has become popular when using a randomly initialized recurrent network as "rich reservoir of dynamics" [34].

Reservoir computing was introduced by Jaeger as Echo State Network (ESN, [30]) and independently by Maass *et al.* under the notion of Liquid State Machine (LSM, [70]). Learning of these reservoir networks is restricted to the weights that project the network state to special read-out neurons. Steil showed in [71] that this functional decomposition of a recurrent network into a fixed reservoir and an adaptive read-out layer arises naturally from the constrained optimization approach first introduced by Atiya and Parlos in [69]. Also, analyzing the weight dynamics during learning in recurrent neural networks shows a limited adaptation of the internal connections in relation to the read-out connections [72]. Restricting learning to the read-out weights circumvents bifurcations of the reservoir dynamics and the resulting problem of a moving target function that maps internal neuron activations to the desired outputs. Then, learning boils down to a simple linear regression problem and can be implemented very efficiently in an offline or online manner [30, 71]. Despite this restriction, the reservoir approach is still powerful because the strength of computing with a dynamical system is preserved: Transient network

states encode the history of external inputs and can serve as short-term memory.

The reservoir paradigm of using a non-linear dynamical system for computation has been implemented using various reservoir "substrates". Typically, rate-coding or spiking neurons are utilized to build up a reservoir of dynamics [30, 70]. But also domain-specific reservoir models, e.g. comprising frequency sensitive neurons [73, 74, 75], can be used to build a reservoir. Recent research efforts try to exploit fast non-linear dynamics in photonic circuits for computation [76]. Fernando and Sojakka took the "liquid" approach literal and implemented a machine that can classify spoken digits using a bucket of water as reservoir substrate [77]. Even earlier than the formulation of echo state and liquid state approaches, the idea to use a fixed dynamical system instead of a fully trained recurrent neural network was proposed in [78] using iterative function systems as substrate of computation. These examples show that the general idea to read out the state of a dynamical system is very powerful. In the remainder of this thesis, however, only reservoirs comprising rate-coding neurons with sigmoidal activation functions are considered.

The reservoir computing paradigm to train only a perceptron-like read-out layer relies on the encoding of inputs in the reservoir by means of a "random, temporal and non-linear kernel" [79]. The mixture of both spatial and temporal components of the input is based upon three key ingredients: (i) the random projection into a high-dimensional state space (ii) with non-linearity introduced by typically sigmoidal activation functions and (iii) the recurrent connections in the reservoir implementing a short-term memory. On the one hand, the non-linear projection into a high-dimensional space is related to spatial kernel expansions which rely on the concept of a non-linear transformation of the original data into a high-dimensional space and the subsequent use of a simple, mostly linear, model. On the other hand, the recurrent connections implement a short-term memory by means of transient network states. Reservoir computing thereby exploits the "architectural bias" [80] of recurrent neural networks with small weights to finite memory machines even prior to learning. Due to this short-term memory, reservoir networks are typically utilized for temporal pattern processing such as time-series prediction, classification and generation [30]. In recent work, it is shown that recurrence in the reservoir also enhances the spatial encoding of static inputs in the reservoir [81]. Reservoir computing based on attractor states is therefore also suited for static pattern processing [82, 83, 84, 85].

## 3.2 Associative Reservoir Computing

In this thesis, a generalized version of reservoir networks is considered which extends the reservoir computing paradigm to a bidirectional, associative setup. Associative Reservoir Computing (ARC) is of particular interest in the context of sensory-motor tasks [22, 23]. A single network then learns the bidirectional relation between inputs and outputs: The mapping from sensory stimuli to motor commands, and the reverse mapping that can be used to predict or fill in missing sensory inputs. The network can either relate two (or more) time-series with each other, or be used in a static manner and learn to bidirectionally associate non-temporal signals. It further enables learning of ambiguous forward and inverse models [55], and implements an adaptive controller for movement generation [24, 55].

In the context of associative reservoir computing, feedback of the network output into the reservoir is a key ingredient. Although there are successful applications reported that use reservoir networks with output feedback [22, 23, 86, 71], the effects of output feedback have not been investigated systematically. In principle, I expect output feedback to add information of the output to the reservoir representation but also to cause additional transients which can be amplified in the worst case and result in a degraded performance. To grasp the problem of error amplification by output feedback, the concept of output feedback stability is introduced in this chapter. Alleviation of error amplification is a necessary prerequisite for robust association and this thesis mainly contributes to this question in Chapter 5 and Chapter 7.

Formally, the ARC architecture (see Fig. 3.1) comprises a recurrent network of non-linear

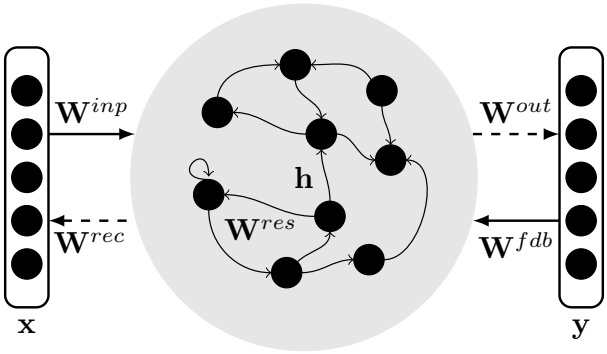



Fig. 3.1: Associative Reservoir Computing setup: Inputs and outputs are bidirectionally connected via the internal representation in the reservoir.

reservoir neurons $\mathbf{h} \in \mathbb{R}^R$ that interconnect inputs $\mathbf{x} \in \mathbb{R}^D$ and outputs $\mathbf{y} \in \mathbb{R}^O$. $\mathbf{W}^{net}$ captures all connection sub-matrices between neurons in the network and is defined by

$$\mathbf{W}^{net} = \begin{pmatrix} \mathbf{0} & \mathbf{W}^{rec} & \mathbf{0} \\ \mathbf{W}^{inp} & \mathbf{W}^{res} & \mathbf{W}^{fdb} \\ \mathbf{0} & \mathbf{W}^{out} & \mathbf{0} \end{pmatrix}. \tag{3.1}$$

Only connections $\mathbf{W}^{out} \in \mathbb{R}^{O \times R}$ and $\mathbf{W}^{rec} \in \mathbb{R}^{D \times R}$ projecting to the input and output neurons are trained by error correction (illustrated by dashed arrows in Fig. 3.1). All other weights, i.e. $\mathbf{W}^{inp} \in \mathbb{R}^{R \times D}$, $\mathbf{W}^{res} \in \mathbb{R}^{R \times R}$ and $\mathbf{W}^{fdb} \in \mathbb{R}^{R \times O}$, are initialized randomly with small weights and remain fixed. Functionally, $\mathbf{W}^{inp}$, $\mathbf{W}^{res}$ and $\mathbf{W}^{fdb}$ parameterize the excitation of the reservoir by inputs, outputs and the reservoir itself. The read-out weights $\mathbf{W}^{out}$ are trained to fit a desired input-to-output mapping from training examples, whereas $\mathbf{W}^{rec}$ is trained to *reconstruct* inputs from outputs.

Consider discrete reservoir dynamics

$$\mathbf{a}(k{+}1) = \mathbf{W}^{inp}\mathbf{x}(k) + \mathbf{W}^{res}\mathbf{h}(k) + \mathbf{W}^{fdb}\mathbf{y}(k) \tag{3.2}$$

$$\mathbf{h}(k) = \sigma(\mathbf{a}(k)) \tag{3.3}$$

with time steps $k \in \mathbb{N}$, where $\mathbf{h}(k)$ is obtained by applying non-linear activation functions $\sigma_i(\cdot)$ component-wise to the neural activations $a_i(k), i{=}1 \ldots N$. Typically, parameterized, sigmoidal activation functions like the logistic function

$$h_i = \sigma_i(a_i) = \frac{1}{1 + \exp\left(-s_i a_i - b_i\right)} \tag{3.4}$$

or the hyperbolic tangent

$$h_i = tanh_i(s_i a_i + b_i) \tag{3.5}$$

with slopes $s_i$ and biases $b_i$ are used. Input and output neurons have the identity as activation function, i.e. are linear neurons, and are updated according to

$$\mathbf{y}(k{+}1) = \mathbf{W}^{out}\mathbf{h}(k) \tag{3.6}$$

$$\mathbf{x}(k{+}1) = \mathbf{W}^{rec}\mathbf{h}(k). \tag{3.7}$$

The network dynamics can be compactly written by collecting input, reservoir, and output neurons in a vector $\mathbf{z}(k) = (\mathbf{x}(k)^T, \mathbf{h}(k)^T, \mathbf{y}(k)^T)^T$. Then, the dynamics of the entire system are described by

$$\mathbf{z}(k+1) = \sigma(\mathbf{W}^{net}\mathbf{z}(k)), \tag{3.8}$$

where $\sigma$ are then linear activation functions for input and output neurons.

### 3.2.1 Associative completion with output feedback dynamics

Association of inputs and outputs is accomplished by iterating output feedback dynamics. The forward model, which maps inputs to outputs, is queried by driving the network with external inputs $\mathbf{x}(k)$ while estimated outputs $\hat{\mathbf{y}}(k)$ are fed back into the hidden state in a recursive loop (compare Fig. 3.2). Equation (3.2) of the network dynamics changes to

$$\mathbf{a}(k+1) = \mathbf{W}^{inp}\mathbf{x}(k) + \mathbf{W}^{fdb}\hat{\mathbf{y}}(k) + \mathbf{W}^{res}\mathbf{h}(k). \tag{3.9}$$

Although the network's input neurons are clamped to the external signal, an estimated value $\hat{\mathbf{x}}(k+1) = \mathbf{W}^{rec}\mathbf{h}(\mathbf{x}(k), \hat{\mathbf{y}}(k)) = \mathbf{W}^{rec}\sigma(\mathbf{W}^{inp}\mathbf{x}(k) + \mathbf{W}^{fdb}\hat{\mathbf{y}}(k) + \mathbf{W}^{res}\mathbf{h}(k))$ for these clamped neurons is also available. The forward model coins also the notion of output feedback dynamics because estimated outputs are fed back into the model.

For the backward model, which is externally driven by $\mathbf{y}(k)$, (3.2) alters to

$$\mathbf{a}(k+1) = \mathbf{W}^{inp}\hat{\mathbf{x}}(k) + \mathbf{W}^{fdb}\mathbf{y}(k) + \mathbf{W}^{res}\mathbf{h}(k). \tag{3.10}$$

In this case, the estimated inputs are fed back recursively into the network, which is also referred to under the general notion of output feedback dynamics since then inputs become the functional outputs of the model.

Note that the forward path from inputs $\mathbf{x}$ to outputs $\mathbf{y}$ typically models an inverse problem in the sense of Chapter 2. This is often more convenient because generally applications require to solve inverse problems. With respect to this convention, the inverse model is "from left to right" (input-to-output) and the forward model "from right to left" (output-to-input).

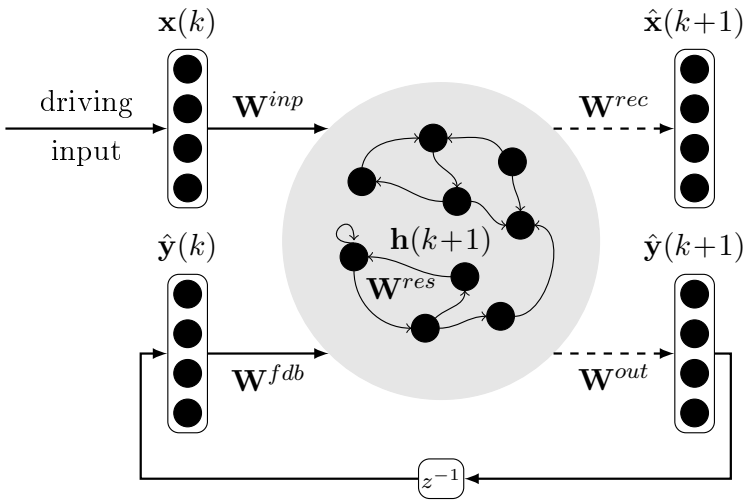



Fig. 3.2: Output feedback dynamics of an associative reservoir network: Estimated outputs $\hat{\mathbf{y}}$ from the last iteration step are fed back into the network whereas externally driven inputs are clamped to desired values $\mathbf{x}$. Note that estimated outputs for externally driven inputs (here $\hat{\mathbf{x}}$ for $\mathbf{x}$) can be estimated simultaneously though they do not enter the output feedback loop.

The partial combination of both models is also possible which implements a mixed constraint satisfaction in input and output space. In a generic form, which also includes the cases of pure forward or backward mappings, the output feedback dynamics can be written as

$$\mathbf{a}(k+1) = \mathbf{W}^{inp}\hat{\mathbf{x}}^*(k) + \mathbf{W}^{fdb}\hat{\mathbf{y}}^*(k) + \mathbf{W}^{res}\mathbf{h}(k) \tag{3.11}$$

with $\hat{\mathbf{x}}^*(k) = (\hat{x}_1^*(k), \ldots, \hat{x}_D^*(k))^T$, where $\hat{x}_i^*(k) = x_i(k)$ for all constrained input components $i$, and $\hat{x}_i^*(k) = \hat{x}_i(k)$ for all unconstrained input components. The same notational convention applies to $\hat{\mathbf{y}}^*$ and allows to flexibly mix constrained and unconstrained components in input and output space, i.e. *associative completion*.

### 3.2.2 Transient- and attractor-based computation

Originally, computation in reservoir networks is based on transient network dynamics and utilized for time-series processing where transient network states serve as short-term memory. Transients are temporally elusive but reproducible traces in the network state trajectory (see [87] for a discussion). In case of static, temporally non-contiguous data, transient dynamics by means of a short-term memory are not meaningful because each sample is independent of the previous one. Then, *attractor-based computation*, i.e. let the dynamics settle to a fixed-point attractor before the estimated output is interpreted, is favorable [82, 81].

Note that there are two distinguished "sources" of dynamics considered in this thesis: (i) Internal dynamics of the reservoir layer set up by $\mathbf{W}^{res}$, and (ii) dynamics due to the output feedback loop. Considering the reservoir dynamics, the mapping $(\mathbf{x}, \mathbf{y}) \mapsto \bar{\mathbf{h}}$ from externally driven inputs to attractor states $\bar{\mathbf{h}}$ is unique if the reservoir subsystem is globally asymptotically stable. Unique attractor states of the reservoir layer are essential for learning of a functional relationship between inputs and outputs. Conditions for convergence of the reservoir layer are discussed more deeply in Sec. 3.2.3. Properties of the output feedback dynamics are the main topic of the following chapters. Depending on the task, globally stable, oscillatory or even multi-stable output feedback dynamics are targeted. Estimated outputs obtained from converged output feedback dynamics are denoted by $\hat{\mathbf{y}}(\mathbf{x})$. The same notation is utilized for estimated inputs $\hat{\mathbf{x}}(\mathbf{y})$.

In this section, technical details of monitoring the convergence of the network state are described. A short discussion of transient- and attractor-based computation in the context of different applications follows in Sec. 3.2.4.

#### Monitoring the convergence of reservoir networks

Application of the attractor-based computation scheme requires to monitor the convergence of the network state. The following algorithm monitors convergence of the state based on the fixed-point condition

$$\mathbf{h}(k+1) = \mathbf{h}(k). \tag{3.12}$$

Algorithm 3.1 iterates the network dynamics (3.2), (3.3), (3.6) with clamped input pattern $\mathbf{x}$ until the network state change $\Delta h$ falls below a small constant $\delta$. The same scheme can be applied if the network is driven by outputs $\mathbf{y}$ or a combination of inputs and outputs. Algorithm 3.1 approximately terminates if the network state reaches a fixed-point, or the reservoir state does not converge in $k_{max}$ iteration steps.

Algorithm 3.1 relies on the threshold $\delta$ to determine convergence and is thus approximate in nature: If the network dynamics fulfill the fixed-point condition (3.12), convergence is identified correctly. But also state changes below the threshold $\delta$, which are not necessarily in the vicinity of a fixed-point, can be mistaken to be a fixed-point. However, Algorithm 3.1 has proven to be adequate if the admitted number of time steps $k_{max}$ is sufficiently large and $\delta$ small.

**Algorithm 3.1** Convergence algorithm

**Require:** get external input $\mathbf{x}$
**Require:** set $k=0$, $\Delta h=\infty$, $\delta=10^{-6}$ and $k_{max}=1000$
1: **while** $\Delta h > \delta$ and $k < k_{max}$ **do**
2: inject external input $\mathbf{x}$ into network
3: execute network iteration (3.2)–(3.7)
4: compute state change $\Delta h = \|\mathbf{h}(k) - \mathbf{h}(k-1)\|^2$
5: $k=k+1$
6: **end while**
7: return $k$ (if $k = k_{max}$, the network did not converge)

#### Observation of network convergence

The convergence criterion monitors whether an observable connected to the change of state in the network drops below a certain value. Measuring the state change directly as proposed in Algorithm 3.1 is rather inefficient. It is therefore useful to extend the framework to monitor the convergence of a network by considering different observables of the network state change. For example, measuring $\Delta h$ by $\|\mathbf{h}(k) - \mathbf{h}(k-1)\|^2$ rather than $\|\mathbf{h}(k) - \mathbf{h}(k-1)\|$ is an option which saves the computation of the square root. An adopted Hopfield energy of the network was utilized in [83] to monitor and verify convergence in experimental settings. These measures implement plainly the general idea of a change in state space of the network and are directly related to the common fixed-point condition (3.12). However, their computation is inefficient, either because the last network state has to be stored or the network weights are involved in their computation which results in quadratic complexity in the number of neurons.

A more efficient way of measuring the rate of change of the network state is to use the sum of its components, i.e. approximating $\Delta h$ by

$$\|\sum_{i=1}^{R} h_i(k) - \sum_{i=1}^{R} h_i(k-1)\|^2.$$

This observable is ambiguous, i.e. there can be different states $\mathbf{h}(k)$ and $\mathbf{h}(k-1)$ while the component-wise sum of both states is equal. This case, however, is very rare. But, and this is important, the simplified observable of the state change is very efficient to compute and requires only the storage of a scalar from iteration $k$ to $k+1$. In practice, the author could not observe any significant differences between using one of the above mentioned criteria and therefore applied the efficient implementation mostly in this thesis.

### 3.2.3 Network initialization and the Echo State Property

Supervised learning in the reservoir approach is restricted to weights $\mathbf{W}^{rec}$ and $\mathbf{W}^{out}$ that project the internal representation $\mathbf{h}$ to inputs and outputs. The remaining weight matrices $\mathbf{W}^{inp}$, $\mathbf{W}^{res}$ and $\mathbf{W}^{fdb}$ are randomly initialized and remain fixed. The initialization of these weights is therefore important for performance and stability [30, 88, 89]. Typically, the weights are drawn from an uniform distribution in $[-a^\star, a^\star]$, where $\star$ denotes the respective sub-matrix $\mathbf{W}^\star$ of the network. Often, sparse connectivities $0 \leq \rho^\star \leq 1$ are preferred, where $\rho^\star$ denotes the density of connections in the respective sub-matrix of the network.

More sophisticated initialization procedures for reservoir networks have been proposed that consider connectivity patterns according to topological constraints (see [90] for a discussion), or condensed reservoir creation schemes based on permutation matrices [91] and other reservoir creation rules [92]. However, the gain in performance is rather restricted if initialization parameters are not adopted to the task at hand and therefore these specialized initializations

of the weight matrices are not further considered in this thesis. I stick to uniformly distributed network weights that are sometimes sparse but not according to a specific structure.

The reservoir matrix $\mathbf{W}^{res}$ plays a special role in the network configuration: The recurrent reservoir connections introduce dynamics in the reservoir layer. These reservoir dynamics functionally serve as short-term memory [30] or, in case of attractor-based computation, nonlinearly mix the input representation and produce complex features [89, 81]. In either case, it is important that these reservoir dynamics have some asymptotic properties, i.e. wash out initial conditions, such that the reservoir state is uniquely determined by a finite history of inputs. This requirement for computation that reservoirs have *echo states* has been formalized by Jaeger under the notion of the Echo State Property (ESP, [30]). The ESP implies global asymptotic stability of the reservoir state in case of no output feedback connections ($\mathbf{W}^{fdb} = \mathbf{0}$) [30]. Hence, reservoirs that have the ESP always converge to an unique attractor state for constant inputs independent of initial conditions. This is important for attractor-based computation with reservoir networks because then the input to reservoir state mapping does not depend on previous reservoir states.

Unfortunately, there is no necessary and sufficient condition for the ESP known so far and mostly stricter sufficient or only necessary conditions are typically applied to obtain reservoirs with the ESP. These criteria are based on the maximal singular value or the maximal absolute eigenvalue of $\mathbf{W}^{res}$, are rather too conservative or actually do not guarantee the ESP, and apply strictly only for constant input. In practice, the sufficient criterion based on the maximal singular value turned out to yield "poor" reservoir dynamics [30]. Therefore, the scaling of the spectral radius $\lambda_{max}(\mathbf{W}^{res})$, i.e. the largest absolute eigenvalue of $\mathbf{W}^{res}$, to the border of instability is commonly applied in reservoir computing in order to obtain "rich" dynamics, even though a tighter bound for the ESP that is based on matrix operator norms has been proposed in [93].

In summery, the weights $\mathbf{W}^{inp}, \mathbf{W}^{res}$ and $\mathbf{W}^{fdb}$ are first randomly initialized according to ranges $a^{inp}, a^{res}, a^{fdb}$ and densities $\rho^{inp}, \rho^{res}, \rho^{fdb}$. Then, the reservoir matrix $\mathbf{W}^{res}$ is scaled, i.e. multiplied by a factor, such that the spectral radius $\lambda_{max}(\mathbf{W}^{res})$ is close to unity.

### 3.2.4 Supervised read-out learning

In this section, supervised learning schemes for the read-out layers are introduced. In principle, any supervised learning rule could be applied and there are various implementations of reservoir networks with rather exotic read-out learning mechanisms. For the sake of simplicity, only the learning rules utilized in this thesis are discussed comprising a standard offline learning scheme based on regularized linear regression and an efficient online learning rule.

#### Offline learning by linear regression

Typically, reservoir learning focuses on the optimization of the read-out weights $\mathbf{W}^{out}$ that project the reservoir state to the read-out neurons in order to infer a desired input to output mapping from a set of training examples. Consider a sequence of input-output pairs $(\mathbf{x}(k)^T, \mathbf{y}(k)^T)^T$, where $k = 1, \ldots, K$. The goal is to minimize the mean square error

$$E(\mathbf{W}^{out}) = \sum_{k=1}^{K} \sum_{i=1}^{O} \left( y_i(k) - \mathbf{w}_i^{out} \mathbf{h}(k-1) \right)^2 \tag{3.13}$$

by adapting only the read-out weights $\mathbf{W}^{out}$, where in the following $\mathbf{w}_i^\star$ denotes the $i$-th row of the matrix $\mathbf{W}^\star$. The error $E$ can be understood as one step ahead prediction error because the network output at time $k$ is a projection of the network state at time $k-1$.

The error $E$ can be minimized by linear regression. Therefore, the inputs $\mathbf{x}(k)$ are injected into the network and the reservoir states $\mathbf{h}(k)$ as well as the desired output series $\mathbf{y}(k)$ are

recorded in a reservoir state matrix

$$\mathbf{H} = \begin{pmatrix} \mathbf{h}(0)^T \\ \vdots \\ \mathbf{h}(K-1)^T \end{pmatrix} \in \mathbb{R}^{K \times R} \text{ and target matrix } \mathbf{Y} = \begin{pmatrix} \mathbf{y}(1)^T \\ \vdots \\ \mathbf{y}(K)^T \end{pmatrix} \in \mathbb{R}^{K \times O},$$

respectively. The procedure of collecting the reservoir states $\mathbf{H}$ is called *harvesting states*. In case of output feedback connections $\mathbf{W}^{fdb} \neq \mathbf{0}$, it is common practice to *teacher-force* the network with desired outputs $\mathbf{y}(k)$ during the state harvest. That is, inputs and outputs of the network are clamped to the desired values contained in the training data. The optimal read-out weights are determined by the least squares solution

$$(\mathbf{W}_{opt}^{out})^T = \left(\mathbf{H}^T\mathbf{H} + \alpha^{out}\mathbb{1}\right)^{-1}\mathbf{H}^T\mathbf{Y}, \tag{3.14}$$

where the Tikhonov factor $\alpha^{out} \geq 0$ is used to regularize the read-out weights $\mathbf{W}^{out}$ [36].

Similarly, the weights from reservoir to input neurons $\mathbf{W}^{rec}$ are subject to supervised learning. The mean square error (3.13) is then

$$E(\mathbf{W}^{rec}) = \sum_{k=1}^{K}\sum_{i=1}^{D}\left(x_i(k) - \mathbf{w}_i^{rec}\mathbf{h}(k-1)\right)^2 \tag{3.15}$$

which is minimized by the regularized linear regression solution

$$(\mathbf{W}_{opt}^{rec})^T = \left(\mathbf{H}^T\mathbf{H} + \alpha^{rec}\mathbb{1}\right)^{-1}\mathbf{H}^T\mathbf{X}. \tag{3.16}$$

Both, the adaptation of $\mathbf{W}^{out}$ and $\mathbf{W}^{rec}$, can be accomplished with a unique sweep through the training set.

In case of attractor-based computation, the attractor states of the reservoir are collected: Each input-output sample $(\mathbf{x}_k, \mathbf{y}_k)$ is applied to the network until convergence of the reservoir state to an attractor $\bar{\mathbf{h}}_k$. The attractor states $\bar{\mathbf{h}}_k$ are then used in (3.14) and (3.16) instead of the transient network states $\mathbf{h}(k)$, and the target matrices are set up by inputs and outputs for $k = 1, \ldots, K$.

### Online Backpropagation-Decorrelation learning

Based on the backpropagation-decorrelation (BPDC) learning rule introduced in [71], the read-out weights $\mathbf{W}^{out}$ and $\mathbf{W}^{rec}$ can be trained efficiently and online. In the proposed network configuration, the BPDC learning rule simplifies to

$$\Delta w_{ij}(k) = \frac{\eta}{\|\mathbf{h}(k-1)\|^2}\, h_j(k-1)\,(d_i^*(k) - d_i(k)), \tag{3.17}$$

where $\mathbf{d}(k) = (\mathbf{x}(k)^T, \mathbf{y}(k)^T)^T$ collects all read-out neurons and $d_i^*(k)$ is the desired target value of read-out neuron $i$ at time step $k$. The online learning scheme is illustrated in Fig. 3.3

Conceptually, BPDC constitutes an efficient error correction rule with time-dependent learning rate $\bar{\eta}(k) = \eta\|\mathbf{h}(k-1)\|^{-2}$, input predicate $h_j(k) = \sigma_j(x_j(k))$ and error signal $d_i^*(k) - d_i(k)$. The

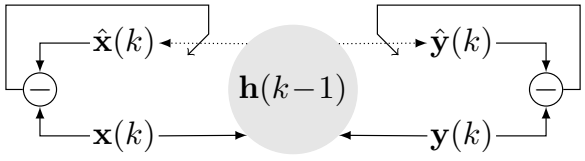



Fig. 3.3: Online learning of associative reservoir networks.

adaptive learning rate $\bar{\eta}(k)$ can be interpreted as a time and context sensitive self-regularization: It scales the gradient step width depending on the overall network activity. Hence, learning is accelerated for sparse network states.

Analogous to offline read-out learning, online learning can also be applied in case of attractor-based computation using the attractor states $\bar{\mathbf{h}}_k$ instead of the transient states $\mathbf{h}(k{-}1)$ in (3.17).

### Attractor-based learning versus learning from trajectories

In the setting laid out in Fig. 3.1 and equations (3.2)–(3.17), there are no direct connections from input to output such that the current output is a projection of the reservoir state at the previous time step. Actually, the input at time step $k$ is incorporated into the hidden state in time-step $k{+}1$ and contributes to the output earliest at time-step $k{+}2$.

In the limit of convergence, the distinction of time steps becomes irrelevant, because input, output and hidden states become temporally congruent. Then, there is no one-step ahead prediction, because at an attractor state it holds that $\mathbf{h}(k{+}1) = \mathbf{h}(k) = \bar{\mathbf{h}}(k)$ and thus $\mathbf{y}(k{+}1) = \mathbf{W}^{out}\mathbf{h}(k) = \mathbf{W}^{out}\mathbf{h}(k{+}1) = \mathbf{W}^{out}\bar{\mathbf{h}}(k)$. Speaking in terms of dynamical associative memories (compare Chapter 2), learning then imprints attractor conditions into the output feedback dynamics.

In case of static mappings and temporally non-contiguous data, learning consequently first iterates the network until convergence by clamping the inputs and then adjust the weights to impose the respective attractor to the network. In case of trajectory data and a static relation between inputs and outputs, this attractor-based learning, however, can be less efficient, because iteration until convergence has to be performed in every step. Then, using trajectory-based learning, where in every time step a new data pair $(\mathbf{x}(k), \mathbf{y}(k))$ is presented to the model, can be more convenient. This learning from correlated data works along reasonably smooth trajectories like robot movements which are naturally limited in their speeds and accelerations. In case of slowly changing input trajectories, actually, learning from trajectories is approximately equivalent to attractor-based learning: In the limit of zero velocity of inputs and outputs, i.e. $||\dot{\mathbf{x}}|| \to 0$ and $||\dot{\mathbf{y}}|| \to 0$, the network is teacher-forced with a static pattern and iteration of the dynamics let the network converge to the associated attractor state $\bar{h}$. The temporal scheme, however, allows a more flexible exploitation and better interpretation in terms of correlated data processing if the training data are smooth trajectories.

Regarding training versus exploitation phase, all combinations of attractor- and transient-based computation can be applied according to application needs:

1. Training and exploitation using transients: This is the typical case in time-series processing with reservoir networks.

2. Training and exploitation attractor-based: This is the typical case in static pattern processing with temporally non-contiguous data.

3. Transient-based training and attractor-based exploitation: Learning from smooth trajectory data can be efficiently accomplished by the transient scheme because then explicit convergence of the network is not required. If there is an instantaneous relation between inputs and outputs, attractor-based computation achieves the best performance during exploitation in particular when the network has output feedback connections [83].

4. Attractor-based training and transient-based exploitation: This is useful in case of a static mapping between inputs and outputs that is learned from possibly temporally non-contiguous data. Transient-based exploitation can be utilized for smooth trajectory generation in this case [55].

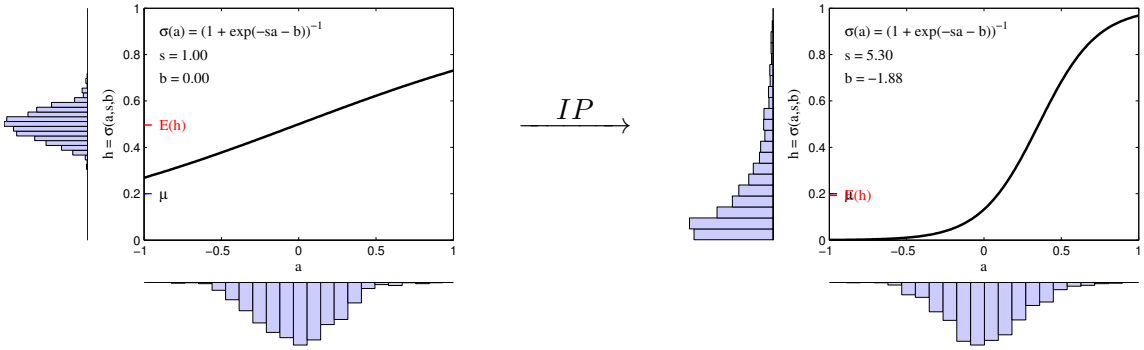



Fig. 3.4: IP learning maximizes a neuron's information transmission properties by adapting the slope $s$ and bias $b$ of the activation function: Initial activation function with input and output distributions (left). Optimized activation function after IP training with approximately exponential output distribution (right).

### 3.2.5 Unsupervised reservoir optimization by intrinsic plasticity

The potential of reservoir computing obviously depends on the properties and the quality of the input encoding in the reservoir. To address this issue, Steil proposed to use a biologically inspired learning rule for unsupervised reservoir optimization that is based on the principle of neural intrinsic plasticity. Intrinsic plasticity (IP) was first introduced in [94] in the context of feed-forward networks. IP changes the neurons' gains $s_i$ and biases $b_i$ of the logistic activation functions (3.4) in order to optimize information transmission of the inner reservoir neurons [86]. That is, neurons should be active only sparsely according to an approximately exponential distribution (see Fig. 3.4). Recent work also shows that IP acts as a regularization on the reservoir representation and therefore improves the generalization performance [89, 95].

More formally, denote by $F_a(a)$ and $F_h(h)$ the input and output distributions of the logistic neuron with $h = \sigma(a, s, b)$ (compare (3.4)). $F_a(a)$ is determined by the statistical properties of the net input the neuron receives in the operating network. Let $F_{\exp}(h) = 1/\mu \exp(-h/\mu)$ be the desired exponential output distribution of a neuron. The average activity rate $\mu$ of the exponential distribution appears later as global parameter for the learning process. Then, the Kullback-Leibler divergence $D$ between $F_h$ and $F_{\exp}$ can be written as

$$D(F_h, F_{\exp}) = H(h) + \frac{1}{\mu} E(h) + \log(\mu)$$

where $H(h)$ is the entropy of the neurons output distribution and $E(h)$ the expected value [96].

Further calculations yield the following online gradient rule to adapt the parameters $s_i$ and $b_i$ with learning rate $\eta_{IP}$ for all reservoir neurons $i = 1, \ldots, R$:

$$\Delta b_i(k) = \eta_{IP} \left( 1 - \left( 2 + \frac{1}{\mu} \right) h_i(k) + \frac{1}{\mu} h_i(k)^2 \right), \tag{3.18}$$

$$\Delta s_i(k) = \eta_{IP} \frac{1}{s_i(k)} + a_i(k) \Delta b_i(k). \tag{3.19}$$

IP is an unsupervised learning technique in the sense that there is no error signal required. Note, however, that IP is fed with information about the output data in reservoir networks with output feedback if teacher-forced training is applied.

A similar IP rule can be derived for *tanh* (3.5) activation functions, where the target distribution can be Gaussian [97] or, with one eye on sparsity, Laplacian [91]. These gradient IP rules are local in time and space and therefore efficient to compute. The idea of IP has been generalized to an offline learning scheme recently in [95] under the notion of Batch Intrinsic Plasticity (BIP). BIP is particularly suited for and efficient with Extreme Learning Machines (ELMs, [33]).

Online IP can be applied in parallel with BPDC learning [86, 98, 24], or in combination with offline read-out learning [99]. The combination of IP and the BPDC learning rule has proven to work well together [86, 98, 23, 24]. For each data sample, the network dynamics (3.2), (3.3) are iterated first before the read-out weights $\mathbf{W}^{rec}$ and $\mathbf{W}^{out}$ are adapted according to (3.17). Finally, the IP rules (3.18), (3.19) are applied at each time step $k$. In case of offline read-out training, the reservoir is typically pretrained by IP before states are harvested for the read-out regression.

Note that IP affects the reservoir dynamics by scaling the reservoir weights $\mathbf{W}^{res}$ implicitly through adaptation of the slopes $\mathbf{s}$. In addition, adaptation of biases $\mathbf{b}$ shifts the point of operation of each activation function. Effects of IP on reservoir stability have been investigated in [86] and [100]. It turns out that, although IP increases the absolute eigenvalues, the reservoir dynamics do usually not "blow up", i.e. the neurons are not driven into saturation. This might be due to the homeostasis constraints formulated by IP itself.

### 3.2.6 General Reservoir Computing Learning Scheme

In this section, the typical design cycle according to the reservoir computing scheme is summerized. Algorithm 3.2 gives a generic recipe and applies to a wide range of reservoir networks including also the non-dynamic Extreme Learning Machine.

First of all, data samples and the task at hand guide the design cycle. Beside the number of input neurons $D$ and output neurons $O$, the task and data properties determine which kind of training and exploitation scheme to apply. For instance, transient-based reservoir computation for time-series prediction tasks, or attractor-based computation for static pattern processing.

Next, the designer has to commit to some general model selection steps. In particular, the size of the reservoir $R$ is an important model parameter. But also the kind of activation functions, and in particular, which sub-matrices of $\mathbf{W}^{net}$ (see (3.1)) to initialize, are crucial design decisions and dramatically affect the network's capabilities. For instance, setting output feedback connections $\mathbf{W}^{fdb} = \mathbf{0}$ restricts the model to unidirectional applications: Driving outputs externally can then not excite the reservoir since $\mathbf{W}^{fdb} = \mathbf{0}$ (compare Fig. 3.1).

Then, the network's parameters have to be initialized. This step is discussed in more detail in Sec. 3.2.3. As a rule of thumb, slopes can be set to unity, i.e. $\mathbf{s} = \mathbf{1}$, and biases $\mathbf{b}$ are initialized uniformly in $[-1, 1]$. A better, input-specific adaptation of biases and slopes can be achieved by applying intrinsic plasticity (see Sec. 3.2.5) in a pretraining phase or during online learning in parallel with read-out weight adaptation. IP learning is well-suited to adopt the model complexity to the task at hand [89] which then mitigates the need to chose an appropriate network size and initialization ranges. Nevertheless, the relative scaling of input, reservoir and probably output feedback weights is not changed by IP and remains a rather important factor for successful training [88]. Scaling of the spectral radius $\lambda_{max}(\mathbf{W}^{res})$ can be understood as another pretraining step (Sec. 3.2.3). The spectral radius is typically scaled close to unity, e.g. $\lambda_{max}(\mathbf{W}^{res}) = 0.95$, if hyperbolic tangent activation functions (3.5) with $\mathbf{s} = \mathbf{1}$ and $\mathbf{b} = \mathbf{0}$ are used. For other configurations of the activation functions, e.g. $\mathbf{b} \neq \mathbf{0}$, the linear approximation of the system is impaired and the value of the spectral radius has to be adopted.

Then, learning of the read-out weights $\mathbf{W}^{out}$ and probably $\mathbf{W}^{rec}$ is conducted. Here, several combinations of learning rules (online or offline) and harvesting modes (teacher-forced or unforced) are possible. Note, however, that the unforced training mode, i.e. feeding only inputs $\mathbf{x}(k)$ into the network during state harvesting, is meaningful only if the network has no output feedback connections, or learning proceeds online: Online learning brings the estimated outputs closer to the target values such the reservoir state is driven into the dynamical regime that is actually apparent if the network is teacher-forced. Unforced online training has been conducted successfully in [101, 23] and in the context with a reward-based, explorative learning rule in [102, 103]. In case of offline training, the learning would not "see" the teacher-forced

**Algorithm 3.2** Reservoir Computing Learning Scheme

**Require:** set of data samples and a task definition

1: chose network size $R$, activation functions $\sigma$ and network connectivity pattern $\mathbf{W}^{net}$
2: initialize network parameters randomly including slopes $\mathbf{s}$ and biases $\mathbf{b}$
3: scale spectral radius $\lambda_{max}(\mathbf{W}^{res})$ (optional)
4: unsupervised pretraining, e.g by intrinsic plasticity (optional)
5: read-out learning (online (3.17) or offline (3.14)–(3.16))
6: exploit network

network dynamics and the network dynamics will be fundamentally different during output feedback-driven network exploitation.

In principle, offline learning proceeds in two steps: First, the reservoir states and the respective targets are harvested. Then, the read-out weights are computed, e.g. by (3.14) and – depending on the model – also by (3.14). Online learning is typically conducted for several sweeps, i.e. epochs, through the data set. Thereby, the harvesting and adaptation phase proceed "online" and in parallel: For each data example, the network state is collected and the read-out weights are adopted. In general, the learning may introduce additional model selection parameters like regularization constants and learning rates.

One has to take care for another important issue to achieve good results when training dynamical reservoir networks ($\mathbf{W}^{res} \neq \mathbf{0}$). Due to the reservoir dynamics, one has to *wash out* initial transients by teacher-forcing the network with part of the training pattern for several time steps. Otherwise, "false" transients will be harvested and used for learning which degrades the performance.

Finally, the trained network is ready for exploitation. Again, several options, e.g. the mode of operation, can be selected. Which mode is suitable is mainly defined by the task at hand (see discussion in Sec. 3.2.4).

## 3.3 Taxonomy of random projection methods

In this section, reservoir and non-dynamic random projection approaches are reviewed and related to each other. Tab. 3.1 puts the discussed random projection methods into a taxonomy depending on key features including network structure, properties of the projection, kind of dynamics, and their typical application. Also, structurally similar approaches that utilize learning schemes like backpropagation to tune their internal representation are briefly discussed. It is pointed out that the ARC model unifies the presented random projection networks.

### 3.3.1 Random projections

Early approaches utilizing random projections are linear and non-dynamic. Basically, a random matrix $\mathbf{W}^{inp}$ is used to linearly project samples $\mathbf{x}_k$ from the high-dimensional data space to a low-dimensional representation, i.e. $\mathbf{h}_k = \mathbf{W}^{inp}\mathbf{x}_k$. Theoretical results support the application of random projections in particular in the context of dimension reduction: Random projections preserve pairwise distances of samples in the data and projection space accurately [104, 105, 106]. Moreover, it was shown that elongated distributions of inputs are more compact after random projection into a lower dimensional space [106].

For these reasons, random projections are mainly applied as efficient and unsupervised dimension reduction technique [107, 106, 108, 109]. Clustering techniques that rely on pairwise distances of the data points can then work easily in the projected space [110]. Also, using the projected data in combination with supervised learning for classification, random projections achieve competitive performance compared to dimension reduction by principle component

analysis [111]. While random projections require no learning, PCA requires the calculation of outer products and the matrix inversion of these correlations which is not feasible for very high-dimensional data. Therefore, random down-projections are considered as a well-balanced trade-off between performance and efficiency in particular for computations on complex data bases [112, 113]. A summary of the basic properties and selected references concerning linear random projections are given in the first row of Tab. 3.1.

### 3.3.2 Extreme Learning Machine (ELM)

A historically rather recent extension of random projections to non-linear function approximation was proposed in [33] under the notion of Extreme Learning Machines (ELMs). ELMs are single-layered feed-forward, non-dynamic networks, where supervised learning is restricted to the perceptron-like read-out layer. The projection from inputs to the hidden layer is randomly initialized and remains fixed. Non-linearity is introduced by sigmoidal activation functions in the hidden layer. Restricting learning to the read-out layer alleviates the need for backpropagation of errors like typically applied in Multi-Layer Perceptrons (MLP) and therefore accelerates learning significantly. Theoretic results show that ELMs have nevertheless universal function approximation capabilities [114].

The ELM approach differs from typical MLPs and linear random projections in the idea to use a non-linear and high-dimensional projection of the input data in the hidden layer, i.e. $R \gg D$. This *random kernel* idea underlies also the following family of dynamic random projection methods. To determine a suitable kernel "size", i.e. number of hidden neurons, several mechanisms have been proposed for automated model selection (see [95] for a brief review). See the second row in Tab. 3.1 for the summarized properties of the ELM.

### 3.3.3 Echo State Network (ESN)

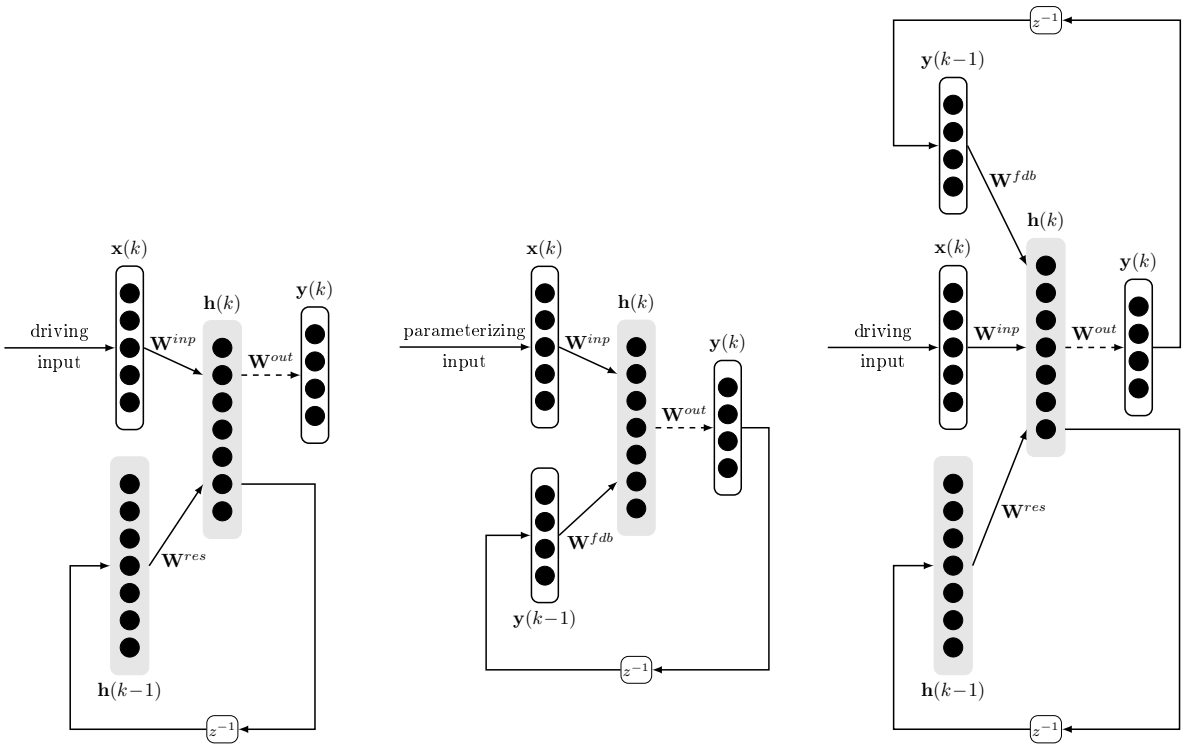
Under the notion of reservoir computing, a family of dynamical and non-linear random projection methods is unified. Though the reservoir paradigm is fairly general and comes in different flavors as discussed in the beginning of the chapter, here only the main prototypical variants with rate-coding neurons and with a special focus on associative reservoir computing are discussed.

In a first step, extending the ELM by random connections between the hidden neurons adds a temporal component to the otherwise spatial encoding of inputs in the hidden layer. The recurrent layer then acts as a "spatio-temporal kernel" which can serve as transient-based short-term memory [30] or as enriched, recurrent mixture of the hidden representation [89, 81]. The combination of a reservoir network with the offline training by regression is often referred to as Echo State Network (ESN), although also online learning can be applied to the same network architecture [30, 83].

Echo State Networks may or may not have output feedback connections $\mathbf{W}^{fdb}$. Although the original ESN formulation already comprises connections $\mathbf{W}^{fdb}$ from the output into the reservoir [30], the definition of the Echo State Property and most experiments exclude output feedback connections explicitly. This is due to the fact that, although the feedback connections $\mathbf{W}^{fdb}$ are also randomly initialized and not subject to adaptation in ESNs, learning of the read-out layer has significant impact on the overall system because estimated outputs are then fed back into the reservoir via $\mathbf{W}^{fdb} \neq \mathbf{0}$ [115]. The role of output feedback and its theoretical implications are discussed in detail in Sec. 3.4.

To this end, it is important to distinguish ESNs with and without output feedback connections $\mathbf{W}^{fdb}$ (compare third and fourth row in Tab. 3.1):

**ESNs without output feedback ($\mathbf{W}^{fdb} = \mathbf{0}$)** are feed-forward function approximators with short-term memory and are therefore typically applied to time-series transduction, prediction




(a) Elman-type network. (b) Jordan-type network. (c) ESN with output feedback.

Fig. 3.5: Echo State Networks with asynchronous update but without output feedback are Elman-type networks [25] (a). Jordan-type networks [48] implement output feedback dynamics and are typically utilized for parameterized pattern generation (b). The combination of Elman- and Jordan-type networks is structurally equivalent to Echo State Networks with output feedback (c).

and classification [30, 116, 34, 52, 98, 86]. Note that ESNs without output feedback connections and recurrent reservoir connections $\mathbf{W}^{res} = \mathbf{0}$ degenerate to Extreme Learning Machines.

The original ESN formulation with layer-wise, asynchronous update of neural activities [30] and without feedback connections from the outputs into the reservoir is structurally equivalent to Elman-type networks [25, 117] with random input and random feedback connections (see Fig. 3.5 (a)). Asynchronous updates are usually applied to multi-layer networks that operate on static data, e.g. forward propagation of activities in MLPs. Throughout this thesis, the synchronous update (3.2) is used in all reservoir implementations and therefore the reservoir networks discussed here can be understood as a full recurrent neural network with special output neurons but without a layered structure which is similar to the ideas in [29, 58]. Technically, applying synchronous network updates in ESNs yields to an increased memory capacity by one time step because then the reservoir is not yet excited by the input at time step $k$ and the previous input at $k-1$ is available for the read-out. In case of attractor-based computation, however, both update rules are equivalent with respect to the reservoir state and read-out learning because then the reservoir is several times updated with the input pattern (see discussion in Sec. 3.2.4).

**ESNs with output feedback ($\mathbf{W}^{fdb} \neq \mathbf{0}$)** are typically trained for autonomous pattern generation [30, 118, 22, 86] by implementing a recursive prediction loop at the output neurons: The "predicted" output $\hat{\mathbf{y}}(k+1) = \mathbf{W}^{out}\mathbf{h}(k)$ is recursively fed back into the reservoir via the output feedback connections $\mathbf{W}^{fdb}$. If the read-out weights $\mathbf{W}^{out}$ are trained properly,

the output feedback dynamics exhibit cyclic attractor dynamics, i.e. generating an oscillatory activity pattern autonomously. Inputs can additionally parameterize the generated pattern [30, 119, 120] (see Fig. 3.5 (b) and (c)).

Implementation of parameterized autonomous pattern generation by recursive prediction is conceptually related to sequence generation in associative memories [5] where the pattern at time step $k$ is associated with the pattern at time step $k+1$. Sequence generation proceeds by recursively feeding associated patterns of the next time step back into the network. The same recursive prediction loop is implemented by Jordan-type recurrent neural networks [48, 15] (see Fig. 3.5 (b)). A more recent approach of autonomous pattern generation with Jordan-type networks is the recurrent neural network with parametric bias (RNNPB) [28, 121]. In Jordan-type RNNs and RNNs with parametric bias, however, learning is applied to all network weights which involves temporal generalization of backpropagation methods. In RNNPB, the parametric bias representation of the stored patterns is additionally self-organized during learning. In reservoir networks, the network parameterization is typically an one-of-k, hand-coded representation of the pattern [30, 119, 120], and restricted adaptivity of the network alleviates the need for backpropagation of errors through the network and time. Besides the efficient learning, generalization by means of transition between generated patterns and their mixture is still possible in ESN with output feedback [120, 119]. Structurally, ESNs with output feedback can be understood as combination of Elman- and Jordan-type networks with random weights from input, hidden and output neurons to the hidden layer (compare Fig. 3.5 (c)).

### 3.3.4 Associative Reservoir Computing (ARC)

The Associative Reservoir Computing (ARC) model was introduced in combination with offline learning in [22] and with online learning in [23]. In the outset of associative methods, there is no dedicated input or output: with respect to the network architecture and the learning algorithm, inputs $\mathbf{x}$ and outputs $\mathbf{y}$ are functionally equivalent (compare Fig. 3.1). It is nevertheless convenient to name $\mathbf{x}$ as input and $\mathbf{y}$ as output, because generally there is a forward and inverse relation in the data (compare to discussion in Chapter 2). The forward or inverse model is queried by either driving the network by inputs, i.e. clamping the input neurons to the desired value, and running the feedback dynamics at the output nodes, or by driving the networks by outputs while the inputs run in a feedback loop (see (3.9)–(3.10)). Due to the functional equivalence of inputs and outputs including the respective feedback dynamics, the network has the same capabilities per direction as ESNs with output feedback. Moreover, associative completion can be accomplished by driving only parts of inputs and outputs (see (3.11)).

ARC generalizes the random projection methods discussed above to a unified network model. The subsumed non-linear random projection methods can be obtained by restricting the network connection matrix $\mathbf{W}^{net}$ of the associative reservoir model (see third column in Tab. 3.1 displaying $\mathbf{W}^{net}$ for each model):

1. Linear random projections with $R \ll D$ for dimension reduction have

$$\mathbf{W}^{net} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{W}^{inp} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & (\mathbf{W}^{out}) & \mathbf{0} \end{pmatrix}$$

   and linear activation functions $\sigma(a_i) = a_i$. Optionally, a trained read-out connections $\mathbf{W}^{out}$ can be attached to the dimension reduction for function approximation purposes. Note that two linear mappings in series are only meaningful in terms of efficient learning, i.e. in cases where $D$ is very large.

2. Extreme Learning Machines (ELMs) are associative reservoir networks with

$$\mathbf{W}^{net} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{W}^{inp} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}^{out} & \mathbf{0} \end{pmatrix}$$

and asynchronous update rules

$$\mathbf{y}(k) = \mathbf{W}^{out}\sigma(\mathbf{W}^{inp}\mathbf{x}(k))$$

propagating activity layer-wise through the network. In particular, non-linear activation functions $\sigma(\cdot)$ and a high-dimensional network state $R \gg D$ distinguish ELMs from the linear random projection approach.

3. Echo State Networks (ESNs) have in addition a recurrent reservoir and optionally output feedback connections as discussed in Sec. 3.3.3. Hence, the network connectivity is given by

$$\mathbf{W}^{net} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{W}^{inp} & \mathbf{W}^{res} & (\mathbf{W}^{fdb}) \\ \mathbf{0} & \mathbf{W}^{out} & \mathbf{0} \end{pmatrix}.$$

4. Associative Reservoir Computing (ARC) networks in addition comprise trained connections $\mathbf{W}^{rec}$ from the reservoir to the input neurons:

$$\mathbf{W}^{net} = \begin{pmatrix} \mathbf{0} & \mathbf{W}^{rec} & \mathbf{0} \\ \mathbf{W}^{inp} & \mathbf{W}^{res} & \mathbf{W}^{fdb} \\ \mathbf{0} & \mathbf{W}^{out} & \mathbf{0} \end{pmatrix}$$

The additional read-out weights $\mathbf{W}^{rec}$ enable to estimate, i.e. "reconstruct", inputs when driving the network outputs with an external signal. For this bidirectional association, output feedback connections $\mathbf{W}^{fdb} \neq \mathbf{0}$ are obligatory. Otherwise, the network state can not be driven by feeding outputs into the network.

5. In the context of static data association, a tailored, "light-weight" variant of ARC is particularly efficient. This Associative Extreme Learning Machine (AELM) goes without a dynamic reservoir, i.e.

$$\mathbf{W}^{net} = \begin{pmatrix} \mathbf{0} & \mathbf{W}^{rec} & \mathbf{0} \\ \mathbf{W}^{inp} & \mathbf{0} & \mathbf{W}^{fdb} \\ \mathbf{0} & \mathbf{W}^{out} & \mathbf{0} \end{pmatrix}.$$

Due to the non-recurrent hidden state, learning and recall of associations is not complicated by internal transients. An asynchronous update rule of the network state, i.e.

$$\mathbf{a}(k) = \mathbf{W}^{inp}\mathbf{x}(k) + \mathbf{W}^{fdb}\mathbf{y}(k) \tag{3.20}$$

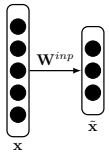
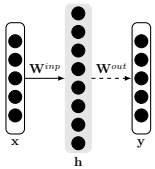
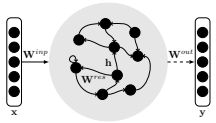
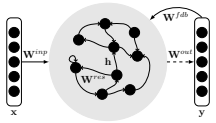
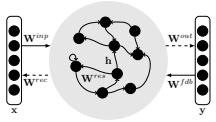
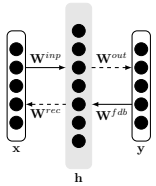
further tailors the model to efficient application on static data (compare with synchronous update rule (3.2)).

The presence of output feedback, however, introduces dynamics to the originally pure feed-forward ELM approach during network exploitation: The network is driven, for example, by external inputs $\mathbf{x}$, while the outputs are iteratively fed back into the network until convergence. I.e. output feedback dynamics implement a dynamical associative completion mechanism in AELMs. The output feedback-driven mode utilizing asynchronous update rules is structurally equivalent to Jordan-type networks (see Fig. 3.5 (b)).

Although, the AELM has no transient dynamics of the "reservoir" layer itself, multi-stable output feedback dynamics can act as attractor-based short-term memory (see Chapter 7). Moreover, transient-based computation utilizing the output feedback dynamics can also be exploited for movement generation (see Chapter 8). The AELM was first proposed by the author in [55] for learning of ambiguous bidirectional mappings.

Sometimes short-cut connections between inputs and outputs are trained which constitute a linear sub-model [30, 122, 23]. In backpropagation-decorrelation networks [71], also recurrent connection between output nodes are considered. In [23], a full recurrent network with a fully occupied block-matrix $\mathbf{W}^{net}$ implements an associative reservoir network. Also, additional bias units for each trained read-out layer can be added [122]. These additional elements are covered by the ARC model presented here, however, I restrict considerations to the core model in the following. Specific network models will be chosen from the presented repertoire of reservoir methods in order to investigate particular aspects of associative reservoir computing. For instance, analyzing the typical ESN with and without output feedback will give basic insights into the role of output feedback dynamics.

Tab. 3.1: Taxonomy of random projection methods.

| Method | Setup | $\mathbf{W}^{net}$ | Projection | Dynamics | Application | Citations |
|---|---|---|---|---|---|---|
| **Random Projection** |  | $\begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{W}^{inp} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & (\mathbf{W}^{out}) & \mathbf{0} \end{pmatrix}$ | • linear<br>• down-projection | • none | • dimension reduction | [106], [110], [111], [104], [105] |
| **Extreme Learning Machine (ELM)** |  | $\begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{W}^{inp} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}^{out} & \mathbf{0} \end{pmatrix}$ | • non-linear<br>• up-projection | • none | • function approximation | [33], [114], [95] |
| **Echo State Network (ESN)** $\mathbf{W}^{fdb} = \mathbf{0}$ |  | $\begin{pmatrix} \mathbf{0} & \mathbf{W}^{rec} & \mathbf{0} \\ \mathbf{W}^{inp} & \mathbf{W}^{res} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}^{out} & \mathbf{0} \end{pmatrix}$ | • non-linear<br>• up-projection<br>• spatio-temporal | • transients<br>• attractors | • function approximation<br>• sequence transduction with transient-based short-term memory<br>• system identification | [30], [86], [81] |
| **ESN** $\mathbf{W}^{fdb} \neq \mathbf{0}$ |  | $\begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{W}^{inp} & \mathbf{W}^{res} & \mathbf{W}^{fdb} \\ \mathbf{0} & \mathbf{W}^{out} & \mathbf{0} \end{pmatrix}$ | • non-linear<br>• up-projection<br>• spatio-temporal | • transients<br>• attractors | • (ambiguous) function approximation<br>• autonomous pattern generation<br>• sequence transduction with transient-based short-term memory<br>• system identification | [30], [119], [83] |
| **Associative Reservoir Computing (ARC)** |  | $\begin{pmatrix} \mathbf{0} & \mathbf{W}^{rec} & \mathbf{0} \\ \mathbf{W}^{inp} & \mathbf{W}^{res} & \mathbf{W}^{fdb} \\ \mathbf{0} & \mathbf{W}^{out} & \mathbf{0} \end{pmatrix}$ | • non-linear<br>• up-projection<br>• spatio-temporal | • transients<br>• attractors | • bidirectional function approximation<br>• autonomous pattern generation and classification<br>• bidirectional sequence transduction with transient-based short-term memory | [22], [23], [24] |
| **Associative ELM (AELM)** |  | $\begin{pmatrix} \mathbf{0} & \mathbf{W}^{rec} & \mathbf{0} \\ \mathbf{W}^{inp} & \mathbf{0} & \mathbf{W}^{fdb} \\ \mathbf{0} & \mathbf{W}^{out} & \mathbf{0} \end{pmatrix}$ | • non-linear<br>• up-projection | • mainly attractors<br>• transients | • bidirectional function approximation<br>• bidirectional sequence transduction with attractor-based short-term memory | [55] |

## 3.4 Output feedback dynamics and error amplification

Output feedback is a crucial ingredient for reservoir computing applications like autonomous pattern generation [118, 36, 120, 123] and bidirectional pattern association [23, 22]. But also Jordan-type recurrent neural networks [48] and the more recent parameterized bias networks [28] comprise an output feedback loop very similar to the reservoir networks discussed in this thesis (see Sec. 3.3.3). In these models, feedback of estimated outputs can potentially amplify small deviations from the target pattern. In this context, "staying close to a target pattern" is often called stability without precise definition, e.g. [36, 120]. I formalize this loose notion by introducing a definition of *output feedback stability* that is tailored to network configurations with output feedback loops and ties learning and stability together. Then, strategies are outlined how to cope with output feedback dynamics.

### 3.4.1 Output feedback and teacher-forcing

We first consider ESNs with and without output feedback connections to facilitate the further discussion. Without output feedback ($\mathbf{W}^{fdb} \equiv \mathbf{0}$), only inputs $\mathbf{x}(k)$ drive the reservoir dynamics and the read-out weights $\mathbf{W}^{out}$ do not affect the reservoir state (compare ESN without output feedback in Tab. 3.1). If $\mathbf{W}^{fdb} \neq \mathbf{0}$, the network state is also driven by outputs $\mathbf{y}(k)$ (compare (3.2) and Fig. 3.1). During learning, the outputs are typically *teacher-forced* [118, 31, 36, 120, 22], i.e. clamped to the desired output sequence $\mathbf{y}(k)$. In exploitation mode, the estimated outputs $\hat{\mathbf{y}}(k)$ are fed back into the reservoir which I refer to as *output feedback-driven* mode. The trained feedback loop parameterized by $\mathbf{W}^{out}$ and $\mathbf{W}^{fdb}$ can lead to error amplification when the network is output feedback-driven [118, 36]. This error amplification is commonly referred to as instability but without a precise definition what stability means.

More formally, teacher-forced dynamics for given inputs $\mathbf{x}$ and $\mathbf{y}$ can be written as

$$\mathbf{a}(k{+}1) = \mathbf{f}(\mathbf{x}, \mathbf{y}, \mathbf{a}(k)) \tag{3.21}$$

meaning that the next state is a function of given inputs and outputs plus the last state at the previous time step. In Echo State Networks, for instance, the teacher-forced dynamics are

$$\begin{aligned} \mathbf{a}(k{+}1) &= \mathbf{W}^{inp}\mathbf{x} + \mathbf{W}^{fdb}\mathbf{y} + \mathbf{W}^{res}\sigma(\mathbf{a}(k)) && (3.22)\\ &= I^{inp} + I^{fdb} + I^{res}(k). && (3.23) \end{aligned}$$

Note that only the inflow $I^{res}(k)$ from the reservoir exhibits "free" dynamics in case of teacher-forcing. That means the inflow from inputs and outputs is fixed and the teacher-forced network dynamics will relax to an unique attractor if the reservoir network has the ESP. Note further that in case of the AELM with $\mathbf{W}^{res} = \mathbf{0}$, the teacher-forced network state is uniquely determined by a single network update.

Partially releasing the teacher-forcing, e.g. driving the network only by inputs, alters the situation to

$$\mathbf{a}(k{+}1) = \mathbf{f}(\mathbf{x}, \hat{\mathbf{y}}(k), \mathbf{a}(k)), \tag{3.24}$$

where $\hat{\mathbf{y}}(k)$ denotes the estimated model output from the last time step that is iteratively fed back to fill in the missing teacher signal: The model exhibits *output feedback dynamics*. In reservoir models with linear read-out neurons, we have

$$\begin{aligned} \mathbf{a}(k{+}1) &= \mathbf{W}^{inp}\mathbf{x} + \mathbf{W}^{fdb}\hat{\mathbf{y}}(k) + \mathbf{W}^{res}\sigma(\mathbf{a}(k))\\ &= \mathbf{W}^{inp}\mathbf{x} + \mathbf{W}^{fdb}\mathbf{W}^{out}\sigma(\mathbf{a}(k{-}1)) + \mathbf{W}^{res}\sigma(\mathbf{a}(k)). && (3.25) \end{aligned}$$

Equation (3.25) makes the output feedback dynamics explicit: The network is fed by estimated outputs $\hat{\mathbf{y}}(k)$ at time step $k$ which are read out from the previous reservoir state $\sigma(\mathbf{a}(k{-}1))$. The

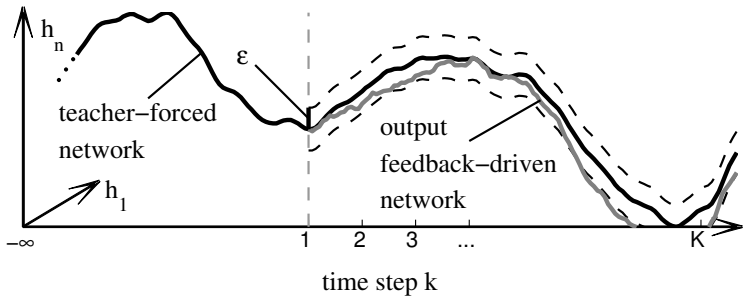



Fig. 3.6: Output feedback stability: Consider the network state sequence for teacher-forced outputs (black line). Releasing the trained network from teacher-forcing beginning at time step $k=1$ results in the output feedback-driven state sequence (gray line). Output feedback stability requires the output feedback-driven network state sequence to stay $\epsilon$-close to the teacher-forced state sequence.

network activity $\mathbf{a}(k-1)$ was itself driven by the estimated output at time step $k-2$. Moreover, (3.25) shows that learning of the read-out weights $\mathbf{W}^{out}$ directly shapes the output feedback loop in combination with the output feedback weights $\mathbf{W}^{fdb}$: The inflow from the output into the network is $\mathbf{W}^{fdb}\hat{\mathbf{y}}(k) = \mathbf{W}^{fdb}\mathbf{W}^{out}\sigma(\mathbf{a}(k-1))$.

### 3.4.2 Output feedback stability

I formalize the concept of stability for ESNs with output feedback. Consider a teacher-forced network state sequence $\mathbf{h}(k)$ with external inputs $\mathbf{x}(k)$ and $\mathbf{y}(k)$ for $k = -\infty, \ldots, 0$. Then, duplicate the network and run one copy further with teacher-forced inputs and outputs. The output neurons of the second copy are released from the teacher signal, i.e. estimated outputs $\hat{\mathbf{y}}(k)$ are fed back into the network, yielding the output feedback-driven sequence $\mathbf{h}_{fdb}(k)$. For input and output sequences $\mathbf{x}(k)$, $\mathbf{y}(k)$ with $k=-\infty, \ldots, 1, \ldots, K$, the output feedback-driven network is said to be *$\epsilon$-output feedback stable* if it remains $\epsilon$-close to the teacher-forced sequence, i.e. $||\mathbf{h}(k) - \mathbf{h}_{fdb}(k)|| < \epsilon$ for $k = 1, \ldots, K$ and some $\epsilon > 0$. Fig. 3.6 illustrates the output feedback stability criterion.

Note that output feedback stability is closely related to the task-specific performance and ties both concepts together: Teacher-forced networks that can not fit desired outputs are not output feedback stable, and non-output feedback stable networks fail to reproduce desired outputs. The output feedback stability criterion is stricter than the related concept of orbital stability which neglects the precise temporal coupling of input and network dynamics [124]. The Echo State Property [30] of the reservoir is a necessary prerequisite for output feedback stability: The network state $\mathbf{h}(k)$ has to be uniquely determined by any left-infinite sequence of external inputs $\mathbf{x}(k)$, $\mathbf{y}(k)$. Output feedback stability extends the notion of echo states to networks with output feedback connections and thereby integrates learning and stability by connecting teacher-forced with output feedback-driven dynamics.

## 3.5 Prospects and challenges

Output feedback stability is crucial for accurate performance in settings with output feedback dynamics. These settings include feed-forward networks that are applied in a recursive prediction loop [25, 26, 20, 27], and (associative) reservoir networks with output feedback connections. Note that the notation and terms also generalize to associative completion and auto-encoder neural networks by combining inputs and outputs to a generalized input vector $\mathbf{u} = (\mathbf{x}^T, \mathbf{y}^T)^T$. Auto-associative models in general can be executed in an output feedback-driven mode. However, convergence of the system, unique or multiple attractor states per driven input and other

dynamic properties depend on many factors ranging from model-specific properties, learning, and the task at hand (compare discussion in Sec. 2.2). In the context of bidirectional association and associative completion, it is essential that the selected inputs have adequate influence on the model, i.e. are sufficient to drive the system. In reservoir models with weak output feedback connections $\mathbf{W}^{fdb}$, output feedback can be neglected and acts rather as noise that perturbs the internal reservoir representation. Reservoirs with small output feedback weights stay output feedback stable despite feedback of erroneous outputs [83]. Then, however, the model can probably not be driven by outputs to recall inputs because the connections $\mathbf{W}^{fdb}$ are too weak. It is therefore important to balance the contributions of inputs and outputs to the reservoir state. This is typically achieved by scaling the matrices $\mathbf{W}^{inp}$ and $\mathbf{W}^{fdb}$ appropriately, which, however, can be rather cumbersome if the input and output data have very different characteristics, e.g. differ in dimensionality, range and energy.

The feedback of outputs into the network introduces a particular difficulty with respect to offline learning. During learning, the outputs are typically teacher-forced to make the target dynamics available for the learning [125, 30]. During network exploitation, estimated outputs, which typically do not match the targets exactly, may result in very different network dynamics [125]. Online learning without teacher-forcing can prevent this problem [23, 101, 102, 103]. Then, however, learning drives the network dynamics through bifurcations which prevents convergence of the learning in the worst case [125]. One-shot programming of output feedback stable dynamics does not suffer from bifurcations during learning. But then, it is essential to cope with the difficulty of teacher-forcing in offline learning scenarios: Releasing the teacher signal may result in error amplification caused by high sensitivity of the trained output feedback dynamics to perturbations. It is therefore important to reduce the gain of the output feedback dynamics in order to achieve a robust network performance. This thesis mainly contributes to the discussion and presents a combination of regularization methods which enable robust offline training of output feedback dynamics. The idea is (i) to implement stable output feedback dynamics by regularizing the reservoir and read-out layer, and (ii) to explicitly program contractive output feedback dynamics. The basic methodology to implement both, reservoir regularization and explicit shaping of attractor dynamics, is introduced in Chapter 4. Then, it is shown in Chapter 5 that regularization of the reservoir actually reduces the gain of the output feedback-driven system. Further, the deeper relation of regularized learning and stability is discussed. The issue of driving the representation equally by either inputs or outputs is addressed in Sec. 5.6. Balancing of contributions is achieved by modeling the propagation of activity in a dendritic extension to the formal neuron model. Robust association utilizing the introduced regularization methods is demonstrated on several tasks in Chapter 6. Explicit shaping of attractor dynamics is finally presented in Chapter 7.

Chapter 4

# Programming dynamics of input-driven recurrent neural networks

In this chapter, a novel technique is introduced that programs desired state sequences into recurrent neural networks in one shot. The approach unifies programming of transient and attractor dynamics in a generic framework. The basic methodology and its scalability to large and input-driven networks is demonstrated by shaping attractor landscapes, transient dynamics and programming limit cycles. Then, the programming dynamics approach is adopted to reservoir regularization.

## 4.1 Programming dynamics

It is a long outstanding question how to determine parameters of dynamical systems that shall display desired behaviors. Traditionally, data-driven parameter estimation for recurrent neural networks (RNNs) has been approached in one of the following contexts: (i) learning of associative memory networks mostly based on correlation matrices, or (ii) learning of general RNNs for approximation of input-output mappings by temporal generalization of supervised backpropagation learning. Learning in the latter case is based on gradient descent with respect to some error criterion. Backpropagation through time and related variants [68, 69] suffer from high computational load, bifurcations of the network dynamics during learning [125], and the typical gradient descent problems like vanishing gradients, local minima, etc. Research in this direction was widely concerned with minimizing the computational load and accelerating convergence of the gradient descent [69], but these efforts can not free learning by gradient descent in recurrent settings from its serious drawbacks.

One-shot learning of RNNs, like it is traditionally applied in the context of associative memory networks [4], is therefore promising. However, attractor and sequence learning are strictly separated in these networks and learning does not easily generalize to input-driven settings. The combination of both, one-shot learning of input-driven RNNs and shaping of transients in addition to desired attractor dynamics, has not yet been accomplished.

In [126], I introduced a generic paradigm to program RNNs efficiently in one shot that applies to a wide range of neural network architectures. Based on the idea to program the state transitions of an observed system into a parameterized model, learning can be formulated as a simple regression problem and can be accomplished efficiently in one shot without descending a gradient. This *state prediction* approach is applicable whenever state transitions can be formulated as linear system of equations with respect to the model parameters. Together

with a trajectory-based sampling strategy, the method unifies programming of transient as well as attractor dynamics in a generic formulation. The combination of sampling desired sequences and one-shot learning solves three problems: First, bifurcations during learning are prevented. Second, shaping of transient and attractor behavior is unified, and third, linear regression is efficient to compute, yields the best parameter estimates with minimal norm and scales to complex network configurations. State Prediction is related to the recently introduced regularization approach for input-driven RNNs [127, 115] and the approach taken by Jaeger in [128] to program the functionality of an external controller into a reservoir network.

The principles of this previous work are presented in a coherent framework for one-shot learning of input-driven RNNs. We first focus on the learning paradigm, its implications, and practical aspects. It is then shown that State Prediction can shape transients, attractor landscapes and input-driven dynamics in a range of scenarios and the results are linked to dynamical system theory.

## 4.2 State Prediction

Consider a system with state $\mathbf{s}(k) \in \mathbb{R}^R$ at time step $k$ that unfolds in time according to a mapping

$$\begin{array}{rccc} \Phi: & \mathbb{R}^{R+D} & \to & \mathbb{R}^R \\ & \Phi(\mathbf{s}(k), \mathbf{u}(k)) & \mapsto & \mathbf{s}(k{+}1), \end{array}$$

where $\mathbf{u}(k) \in \mathbb{R}^D$ are additional input signals that parameterize the transition. Assume one can sample typical flows $\{(\mathbf{s}_i(k), \mathbf{u}_i(k))\}_i$ of the system, where $k = 1, \ldots, K_i$ for the $i$-th sequence. The observed state sequences are modeled with the input-driven recurrent network dynamics

$$\mathbf{s}(k{+}1) = \sigma(\tilde{\mathbf{W}}\tilde{\mathbf{s}}(k)), \tag{4.1}$$

where $\sigma$ is a non-linear activation function, $\tilde{\mathbf{s}}(k) = (\mathbf{s}(k)^T, \mathbf{u}(k)^T, 1)^T$ is the combined input and system state, and $\tilde{\mathbf{W}} = (\mathbf{W}\ \mathbf{W}^{inp}\ b)$ are the model parameters. The recurrent neural network model is illustrated in Fig. 4.1 (left).

Typically, learning is approached by minimizing the error

$$E = \frac{1}{K} \sum_k ||\mathbf{s}^*(k) - \mathbf{s}(k)||^2,$$

where the model parameters $\tilde{\mathbf{W}}$ are adapted in order to bring the sequence $\mathbf{s}(k)$ closer to the desired sequence $\mathbf{s}^*(k)$. Gradient-based approaches lead to recursive dependencies of the states on the weights, i.e.

$$\frac{\partial E(k)}{\partial w_{ij}} = -(s_i^*(k) - s_i(k)) s_j(k{-}1) \sigma'(k) \frac{\partial s_i(k{-}1)}{\partial w_{ij}}.$$

In contrast to the separation of the model dynamics and the target dynamics, the basic idea of State Prediction is to find parameters $\tilde{\mathbf{W}}$ that explain the transitions between successive states $\mathbf{s}^*(k)$ and $\mathbf{s}^*(k{+}1)$ best. That means parameters $\tilde{\mathbf{W}}$ are searched that minimize

$$||\mathbf{s}^*(k{+}1) - \sigma(\tilde{\mathbf{W}}\tilde{\mathbf{s}}^*(k))|| \qquad \text{for } k = 1, \ldots, K, \tag{4.2}$$

where $\tilde{\mathbf{s}}^*(t) = (\mathbf{s}^*(t)^T, \mathbf{u}(t)^T, 1)^T$. This formulation directly incorporates the sampled system transitions $\Phi(\mathbf{s}^*(k), \mathbf{u}(k)) = \mathbf{s}^*(k{+}1)$ for parameter estimation. This state prediction is parameterized by the input $\mathbf{u}(k)$. Note that the prediction in (4.2) is based on the observed state $\mathbf{s}^*(k)$, not on the dynamics of a partially trained model. This optimization by utilizing *harvested* target state sequences is inspired from the reservoir computing paradigm (see Sec. 3.1) and generalized

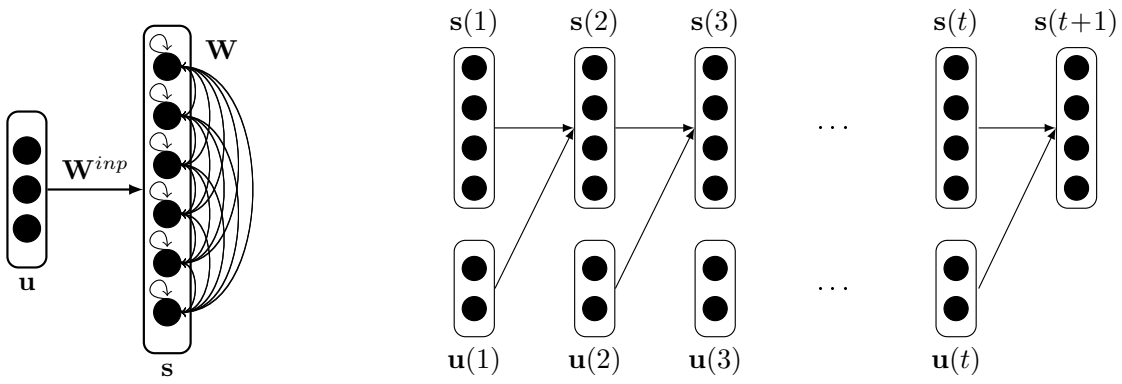



Fig. 4.1: Input-driven recurrent neural network (left). State Prediction (right) models the transitions between successive states given sequences of inputs and states.

to the entire recurrent neural network model in this chapter. Once a desired state sequence is collected, constructive modeling of the sequence $\mathbf{s}^*(k)$ by a RNN for $k = 1, \ldots, K$ can be accomplished using regression techniques. The idea of predicting the next network state given inputs and the last state is illustrated in Fig. 4.1 (right).

To prevent the non-linear activation function $\sigma$ from cluttering the optimization of the model parameters $\tilde{\mathbf{W}}$, I rephrase the *state prediction problem* (4.2) as a linear system

$$\mathbf{a}^*(k+1) \equiv \sigma^{-1}(\mathbf{s}^*(k+1)) = \tilde{\mathbf{W}}\tilde{s}^*(k). \tag{4.3}$$

This linearization is possible if (i) $\sigma$ is invertible and (ii) the observed data is transformed into the output range of $\sigma$. Condition (i) is fulfilled by all common sigmoidal activation functions like the hyperbolic tangent (3.5) or the logistic function (3.4). The second condition can be fulfilled without restriction for all bounded sequences. With one eye on reservoir computing, harvesting state sequences of a recurrent network itself makes targets $\mathbf{a}^*(k)$ in (4.3) directly available.

We are now in a position to solve the linear state prediction problem by simply collecting all $R$ sampled trajectories in a matrix

$$\tilde{\mathbf{S}}^* = \begin{pmatrix} \tilde{\mathbf{s}}_1^*(1)^T \\ \vdots \\ \tilde{\mathbf{s}}_1^*(K_1-1)^T \\ \vdots \\ \tilde{\mathbf{s}}_R^*(1)^T \\ \vdots \\ \tilde{\mathbf{s}}_R^*(K_R-1)^T \end{pmatrix} \text{ and the corresponding targets in } \mathbf{A}^* = \begin{pmatrix} \mathbf{a}_1^*(2)^T \\ \vdots \\ \mathbf{a}_1^*(K_1)^T \\ \vdots \\ \mathbf{a}_R^*(2)^T \\ \vdots \\ \mathbf{a}_R^*(K_R)^T \end{pmatrix}.$$

The optimal solution in the least squares sense to the state prediction problem $||\mathbf{A}^* - \tilde{\mathbf{S}}\tilde{\mathbf{W}}^T||$ is

$$\tilde{\mathbf{W}}_{opt}^T = \left(\tilde{\mathbf{S}}^{*T}\tilde{\mathbf{S}}^* + \beta\mathbb{1}\right)^{-1}\tilde{\mathbf{S}}^{*T}\mathbf{A}^*, \tag{4.4}$$

where $\beta \geq 0$ weights the contribution of a regularization constraint $R(\tilde{\mathbf{W}}) = \sum_{i,j}\tilde{w}_{ij}^2$ corresponding to a Gaussian prior distribution for the model parameters [129, 130].

### 4.2.1 Sampling dynamics for State Prediction

Programming dynamics by State Prediction can be understood as a three-staged process: Observation of the desired system yields a data corpus which is used in a second step to determine

the model parameters by solving the state prediction problem (4.2) by means of $\tilde{\mathbf{W}}_{opt}^{T}$. Then, the programmed model dynamics are ready for exploitation. One can substitute the first stage by synthesizing training data that represent the desired behavior. Observation of the desired dynamics is a key step and there are three basic ways how to sample dynamics for State Prediction:

#### Sampling velocities

Spatial sampling of states $\mathbf{s}$ with corresponding velocities $\mathbf{v}(\mathbf{s})$ of the system at that particular point in state space is one way to acquire training data. Simple integration yields the required pair of successive states $\mathbf{s}(k)$ and $\mathbf{s}(k+1) = \mathbf{s}(k) + \mathbf{v}(\mathbf{s}(k))$. State transitions $\mathbf{s}(k) \rightarrow \mathbf{s}(k+1)$ can also be observed directly. Sampling velocities or state transitions means to observe the system dynamics stepwise for selected states and inputs. This approach elucidates the need for generalization: The programmed model has to operate also in areas of the state space where no training examples are present. A local modeling approach is hopeless because extrapolation is impossible. The network (4.1) models state transitions globally and thus can generalize the system behavior to novel states. However, sampling has to be done carefully: All important regions of the state space have to be included and the number of samples in each region should be sufficiently balanced. This is difficult for high-dimensional systems and therefore restricted sampling can be advantageous.

#### Sampling attractor conditions

Focusing on the target of implementing a specific behavior leads to the idea of sampling only particular conditions. For instance, attractor conditions can be easily formulated in terms of State Prediction using that $\mathbf{s}(k+1) = \mathbf{s}(k)$ if $\mathbf{s}$ is an attractor state. This approach is typically applied to imprint memories into associative networks. Though conceptually curing the problem of sampling from the entire velocity field, there is a serious drawback: The differential equation is only sampled at special points in state space with zero velocity. Surrounding states $\mathbf{s}+\nu$ might not be attracted, i.e. the basin of attraction is too narrow and leaves space for spurious states.

#### Sampling flows

A conceptually and practically appealing way of sampling dynamics is a trajectory-based approach. Simply recording representative flows $\mathbf{s}_i(k)$ for $k = 1, \ldots, K_i$ and sequences $i$ of the system provides exemplary descriptions of the dynamics including, for instance, the size of attractor basins. Recording or synthesizing exemplary state sequences (or a combination of both) solves the previous problems by collecting state transitions only at relevant regions of the state space while preventing degenerated sampling. In addition, trajectory data is typically available in physical systems.

### 4.2.2 State Prediction, associative learning and auto-regression

The state prediction approach combines several ideas from the areas of neural networks and auto-regressive modeling.

#### State Prediction and learning in associative memories

Regarding dynamical associative memories, State Prediction can be understood as generalization to input-driven recurrent neural networks: In addition to a recurrent layer, the model considered here has input neurons that bias, i.e. parameterize, the network dynamics. The discussion above and the experiments that follow this discussion show that State Prediction

conceptually unifies the programming of fixed-point and "sequential" dynamics. From a sequence programming viewpoint, imprinting fixed-point attractor dynamics is a special case where simply attractor conditions are sampled for learning.

Moreover, the formulation of a linear system of equations that is solved by linear regression with additional regularization constraints has not been proposed in this field of research. Although the least squares solution for non-dynamical associative memories has been identified as "optimal auto-associative recollection" [37, 7], research on dynamical associative memories focused on Hebbian and related, local learning rules (see [4, 131] and many others). In the context of dynamical associative memories, learning by linear regression is rarely applied and typically without regularization of the network parameters [132]. Regularization is a common model selection technique in machine learning aiming at improved generalization by preventing over-fitting of the model to the training data. In the context of recurrent neural networks, regularization of learning is important also for stability and therefore recommended. In Chapter 5, the link between regularization and stability is discussed more deeply.

#### State Prediction and learning in general recurrent neural networks

From a recurrent neural network viewpoint, State Prediction unfolds the network dynamics in time (compare Fig. 4.1 (right)). However, standard gradient-based learning in recurrent networks assumes that supervised targets are only available for the output neurons and thus makes backpropagation of errors to the hidden layer necessary. In contrast, the complete state sequence is available in the state prediction framework. This seems to be a strong restriction in the first place, but State Prediction is also applicable to networks with hidden representations. For instance, consider the approach by Atiya and Parlos in [69] who reformulated learning in recurrent networks to implement "virtual" targets in the hidden layer. Even though these virtual targets are mediated by the supervised error at the output neurons, the implementation of these target state sequences, i.e. finding weights that exhibit the desired dynamics, is formalized by the state prediction problem (4.2). The application of State Prediction to reservoir networks without propagating supervised errors is shown later on in this thesis.

#### State Prediction and auto-regressive modeling

From an system identification viewpoint, State Prediction is a special case of non-linear auto-regressive modeling with exogenous inputs (NARX) [133]. Auto-regressive (AR) models with "exogenous" inputs $\mathbf{u}(k)$ have the form

$$\mathbf{s}(k) = F(\mathbf{s}(k-1), \mathbf{s}(k-2), \ldots, \mathbf{u}(k), \mathbf{u}(k-1), \ldots) + \epsilon(k),$$

where the sequence $\mathbf{s}(k)$ is subject of the modeling and $\epsilon(k)$ is the error of the non-linear model $F$ typically due to noise. If $F$ is a non-linear function of the previous sequence steps $\mathbf{s}(k-t)$ and inputs $\mathbf{u}(k-t)$ with $t \in \mathbb{N}$ and $t \leq p$, $F$ is said to be a non-linear auto-regressive exogenous model of order $p$.

State Prediction can be understood as an auto-regressive problem with exogenous inputs of order $p = 1$: Only the last "state" $\mathbf{s}(k-1)$ and input $\mathbf{u}(k-1)$ is utilized for predicting $\mathbf{s}(k)$ (compare Fig. 4.1 (right)). The recurrent model is also non-linear, and thus State Prediction is a NARX model. Note, however, that learning is linear with respect to the parameters (compare (4.3)).

There are also non-linear auto-regressive (NAR) approaches that incorporate recurrent neural networks for non-linear modeling of the state transitions, e.g. [134]. In [134], however, the recurrent network (a Jordan-type network with input delay-line and a delay-line of prediction errors) is trained by standard gradient-based backpropagation methods to approximate the next sequence state at its outputs.

In State Prediction, the perspective is different: We aim at finding recurrent neural network parameters that explain the observed state sequences given external inputs, i.e. the sequence itself is modeled by the state sequence of the recurrent network. The modeling capabilities are therefore restricted by the class of dynamics that the recurrent network can exhibit. This raises the question which kind of dynamics can be learned by State Prediction.

#### Modeling power of State Prediction

A necessary condition for successful learning by State Prediction is that the desired dynamics belong to the class of dynamics which input-driven recurrent neural networks span. In [135, 136], a taxonomy of "classes of dynamics" that recurrent neural networks can display are discussed (for small networks only). Attempts to model state sequences that are not part of this set of dynamics will necessarily fail. If the state sequences can be modeled by the recurrent neural network, it is important that the sampled observations capture the relevant dynamic behavior sufficiently well for learning to be successful (this is also confirmed in experiments in Sec. 4.4). Programming dynamics by State Prediction seems to result in a limited memory capacity of the programmed networks in the first place due to the restricted order ($p = 1$) of the auto-regressive modeling. Note, however, that learning takes state transitions at all time step into account and that the desired state sequence may comprise transients prolonged over several time steps. It is shown in Sec. 4.4 that modeling of state sequences that stem from a recurrent neural network (re-)implements also transient dynamics accurately. In this case, it is clear that the observed dynamics can be modeled by a recurrent network because they have been produced by such a network. Therefore, learning by State Prediction can in principle fully exploit the memory capacity of recurrent neural networks.

## 4.3 Programming dynamics by predicting states

Sampling of velocities and sampling of flows for programming dynamics is demonstrated in this section.

### 4.3.1 Programming the dynamics of a single neuron

We start with the minimal possible scenario and program the dynamics of a single neuron with $\sigma \equiv tanh$ and apply the sampling of velocities approach. A single neuron is used without bias or inputs such that (4.1) has only one parameter $w$. The dynamics of the neuron can be *uni-stable*, i.e. a single globally and asymptotically stable fixed-point exists, or *bi-stable*, i.e. two fixed-points are separated by a saddle [137, 138, 139].

I create training data by sampling states $s$ and their respective velocities $v(s)$ from a potential field

$$P(s) = -0.1 \sum_{i=1}^{2} (5(p_i - s)^2 + 1)^{-1}$$

with two desired attractors located in state space at $p_1$ and $p_2$. I move $p_1$ and $p_2 = -p_1$ from zero (potential field with a unique basin) to 0.7 (two detached basins) and train for each potential field a network model. The State Prediction is conduct by modeling

$$s^* + v(s^*) = tanh(ws^*), \text{ where } v(s^*) = -\frac{\partial}{\partial s} P(s^*)$$

are the respective velocities. States and velocities are sampled near to the charges $p_1$ and $p_2$, and $\beta = 0$ is used in (4.4) for training.

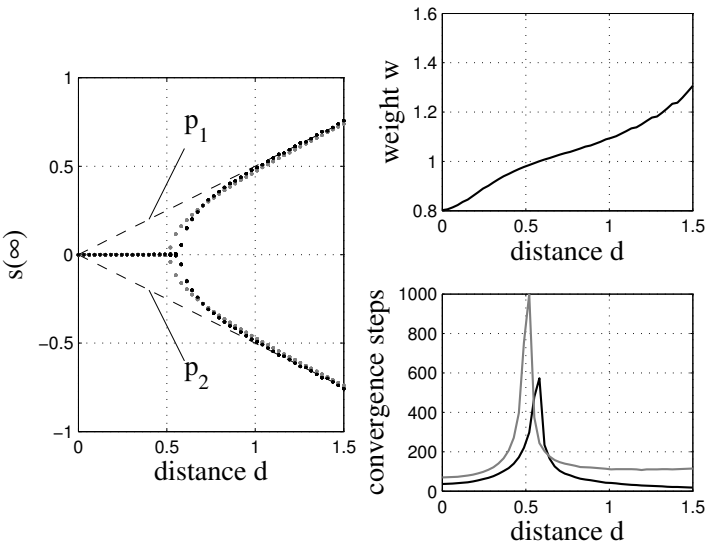



Fig. 4.2: Bifurcation of dynamics depending on the distance $d$ between the charges. Programmed networks (black) and analytic dynamics (gray).

tb!]

Fig. 4.2 (left) shows the attractor states $\bar{h} = s(\infty)$ of the programmed networks (black) and the potential field dynamics (gray) for different initial conditions as function of the distance $d = |p_1 - p_2|$ between the two charges. Note that the discrete dynamics given by $s(k+1) = s(k)+v(s(k))$ as well as the programmed network dynamics bifurcate at $d \approx 0.5$. The bifurcation introduces a saddle, i.e. is a saddle-node bifurcation, which explains the peak number of steps until convergence shown in Fig. 4.2 (bottom right). When both charges are separated further, this effect vanishes and the number of iterations until convergence decreases again. The weight of the programmed system is shown in Fig. 4.2 (top right) and also displays the point of bifurcation: When the weight surpasses unity, global asymptotic stability of the linearized system is not guaranteed anymore [93, 30].

### 4.3.2 Programming two-neuron circuits

In this section, we focus on two-neuron circuits without inputs. It is shown that a variety of behaviors can be programmed into such circuits by simply providing some example sequences, i.e. sampling flows. The sequences are generated synthetically by a simple strategy: For each fixed-point attractor $\bar{\mathbf{s}}^*$, I select random perturbations $\nu_i \in \mathbb{R}^N$ and generate sequences by setting

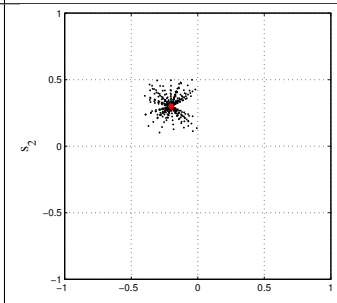
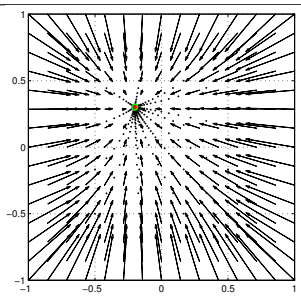
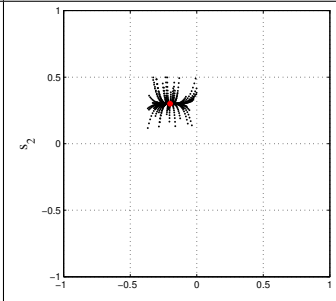
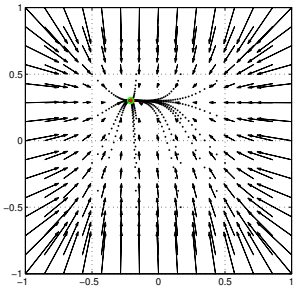
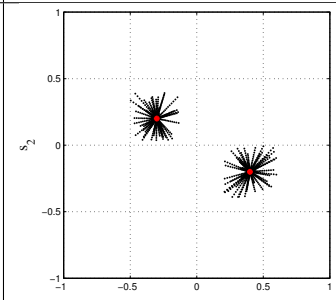
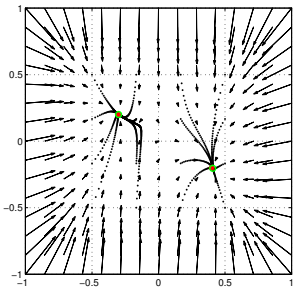
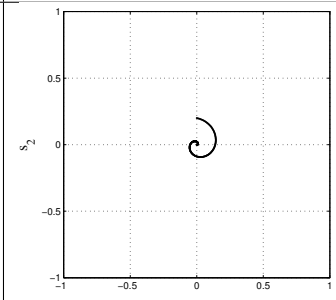
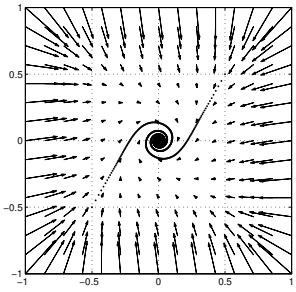
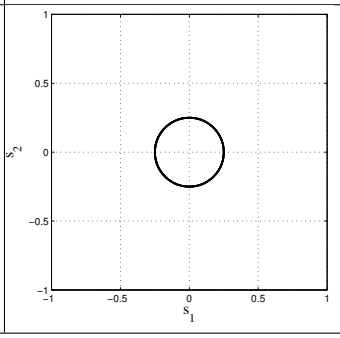
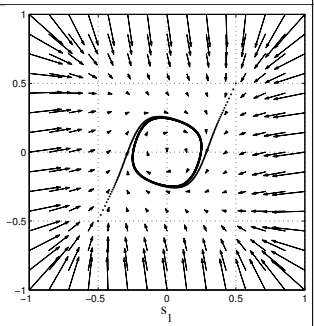
$$\mathbf{s}_i^*(k) = \bar{\mathbf{s}}^* + (1 - k/K_i)^2 \nu_i \qquad \text{with} \qquad k = 0, \ldots, K_i.$$

For each fixed-point, I typically use $K_i = 20$ for $i = 1, \ldots, 50$ and chose perturbations $\nu_i$ uniformly distributed in $[-0.2, 0.2]^2$. The neurons have $\sigma \equiv tanh$ and $\beta = 0$ is used in (4.4) for training.

Tab. 4.1 shows various mappings from target sequences (2nd col.) to programmed dynamics (3rd col.). The network parameters are given in the fourth column. For fixed-point attractors, the desired attractor positions (red dots) and the actual attractors of the programmed networks (green circles) show that the fixed-point conditions are accurately implemented by the networks. Generally, a few example sequences or even a single sequence is sufficient to implement the desired behavior (see rows 4 and 5 in Tab. 4.1). Note that the training data does not only shape the dynamics in the limit case $k \to \infty$, i.e. the location of the attractor, but also the transient behavior as can be seen in case of the spiral pattern in the fourth row of Tab. 4.1. In the second row of Tab. 4.1 I used

$$s_{i2}^*(k) = \bar{s}_2^* + (1 - k/K_i)^4 \nu_{i2}$$

Tab. 4.1: Programming networks with two neurons. The second column shows the training data and the third column shows the resulting phase portrait of the programmed network dynamics with exemplary flows. The desired fixed-point attractors are displayed by red dots and the actual attractors of the trained networks are shown as green circles. The fourth column gives the actual network parameters and the spectral radius $\lambda$ of the recurrent weights $\mathbf{W}$. Rounded values are presented for the network parameters which nonetheless describe very similar dynamics.

| **Description** | **Training Data** | **Phase Portrait** | **Parameters** |
|---|---|---|---|
| Single fixed-point attractor |  |  | $\mathbf{W} = \begin{pmatrix} 0.8 & 0 \\ 0 & 0.84 \end{pmatrix}$<br>$\mathbf{b} = \begin{pmatrix} -0.05 \\ 0.06 \end{pmatrix}$<br>$\lambda = 0.84$ |
| Single fixed-point attractor with modulated transients |  |  | $\mathbf{W} = \begin{pmatrix} 0.96 & 0 \\ 0 & 0.88 \end{pmatrix}$<br>$\mathbf{b} = \begin{pmatrix} -0.01 \\ 0.04 \end{pmatrix}$<br>$\lambda = 0.96$ |
| Two fixed-point attractors |  |  | $\mathbf{W} = \begin{pmatrix} 1.04 & -0.01 \\ -0.04 & 0.94 \end{pmatrix}$<br>$\mathbf{b} = \begin{pmatrix} 0.005 \\ 0.002 \end{pmatrix}$<br>$\lambda = 1.05$ |
| Single fixed-point attractor with spiral transients |  |  | $\mathbf{W} = \begin{pmatrix} 0.99 & 0.03 \\ -0.03 & 0.99 \end{pmatrix}$<br>$\mathbf{b} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$<br>$\lambda = 0.99$ |
| Limit cycle |  |  | $\mathbf{W} = \begin{pmatrix} 1.02 & 0.03 \\ -0.03 & 1.02 \end{pmatrix}$<br>$\mathbf{b} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$<br>$\lambda = 1.02$ |

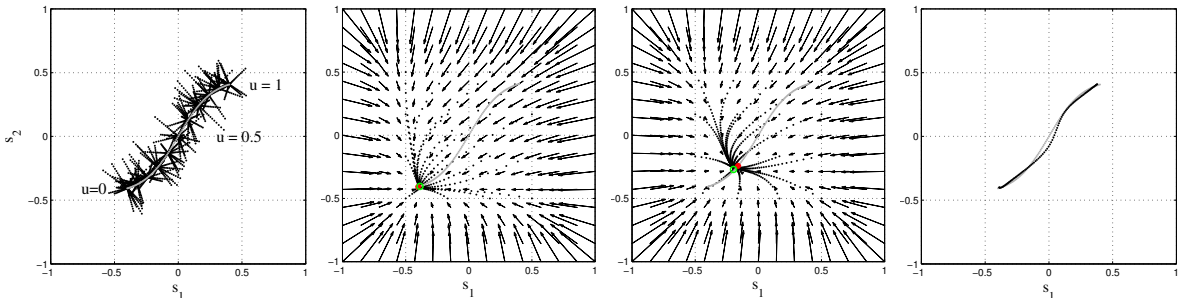



Fig. 4.3: Parameterized network dynamics programmed from training data (left). Phase portraits with flows for $u = 0$ (2nd col.) and $u = 0.2$ (3rd col.). The attractor states for $u \in \{0, 0.01, \ldots, 1\}$ approximate the shape present in the training data (right).

for training data generation. The trained network displays the changed speed of convergence along the second dimension. State Prediction shapes both, the transient dynamics and the behavior in the limit case, which is fundamentally different from learning of associative memories where auto- and temporal hetero-association are strictly separated.

We observe that all uni-stable systems have a spectral radius $\lambda$, the maximal absolute eigenvalue of $\mathbf{W}$, below unity. Bi-stability and cyclic attractors are indicated by spectral radii greater or close to unity. Note, however, that a spectral radius greater unity does not strictly imply loss of global stability if $\mathbf{b} \neq \mathbf{0}$ [93, 30].

### 4.3.3 Programming input-driven network dynamics

In a next step, we program the dynamics of an input-driven RNN that comprises a single input neuron and two internal neurons similar to the network structure shown in Fig. 4.1 (left). Training data is generated as in Sec. 4.3.2 using the same parameters, but this time a sigmoidal structure is hidden in the training data which is only implicitly represented by the one dimensional network input $u$ (see Fig. 4.3 (left)). The phase portraits of the trained network with exemplary flows are shown in Fig. 4.3 for input signals $u = 0$ (2nd col.) and $u = 0.2$ (3rd col.). The flows and velocities reveal that the network has a unique fixed-point for each input. Fig. 4.3 (right) shows the attractor states of the network for a fine-grained sampling of inputs in range $[0, 1]$. The sigmoidal structure is captured well by the parameterized network dynamics. In [127, 115, 128] and this thesis, the functioning of the state prediction approach is confirmed for large networks with multivariate and time-varying inputs.

The example presented in this section is based on the idea to program a parameterized attractor into the network. There is an implicit continuity assumption in the outset of the example: Small changes of the input map to small changes of the attractor position. Learning can be further biased to continuous state prediction mappings by increasing the regularization parameter $\beta$ in (4.4). This will be important for regularization of reservoir networks in the later discussion.

### 4.3.4 Storing sequences in a large network

In this section, I demonstrate the scalability of the approach to recall long and high-dimensional sequences. I program a network with 784 neurons to recall two sequences of digit images taken from the MNIST database [140] (see Fig. 4.4 rows 1 and 5). Samples $\mathbf{s}^1(1) \equiv \mathbf{s}^1(11)$ and $\mathbf{s}^2(1) \equiv \mathbf{s}^2(11)$ are used for training to close the respective sequence loop. The images are scaled into the range $[0.01, 0.99]^{784}$ and $\sigma(a) = 1/(1 + exp(-a))$ is used as activation function. The regularization parameter $\beta = 0.1$ in (4.4) is used for training.

The second and sixth row in Fig. 4.4 show the generated state sequences of the trained network. The network state was initialized with the first pattern of the respective sequence.

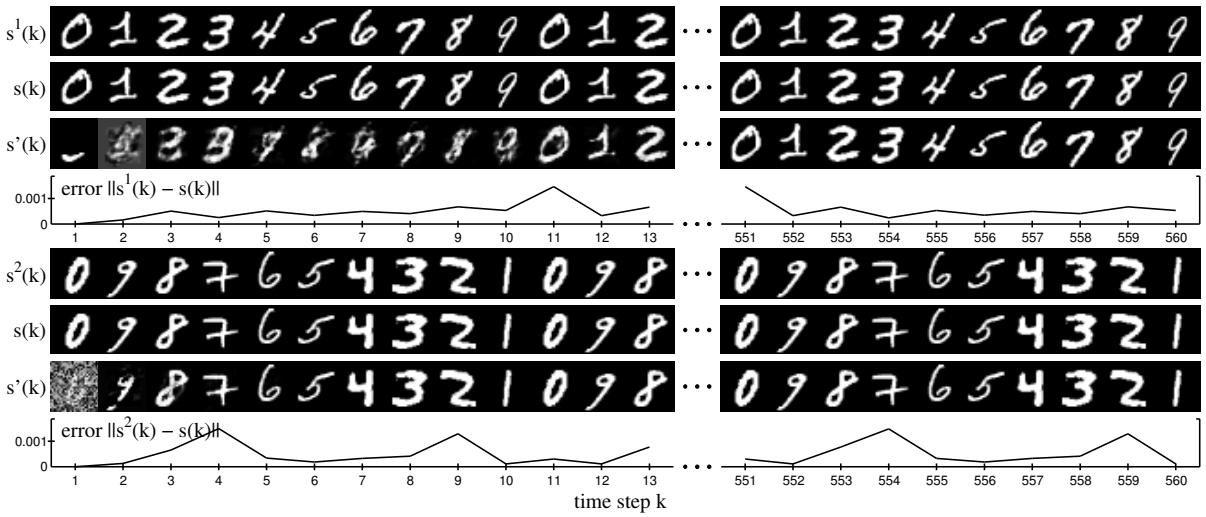



Fig. 4.4: Sequence generation. The 1st and 5th row show the target sequences $\mathbf{s}^1(k)$ and $\mathbf{s}^2(k)$. The 2nd and 6th row show the generated sequences, where $\mathbf{s}(1) = \mathbf{s}^1(1)$ and $\mathbf{s}(1) = \mathbf{s}^2(1)$, respectively. The corresponding instantaneous error is plotted in the 4th and 8th row. The 3rd and 7th row show the robustness against perturbations: The sequences are still recalled when occluding 75% of the initial state or adding noise.

Then, the network update equation (4.1) was iterated without external inputs. Note that the sequences are stably reproduced for hundreds of time steps. There is no amplification of the error in successive cycles which can be seen in the instantaneous error plots in rows 4 and 8. Also, when the network is initialized with strongly corrupted states, the network is attracted to the trained limit cycles (see rows 3 and 7 in Fig. 4.4). This example shows that – in contrast to correlation-based learning of associative memory networks [4] – one-shot storage and robust recall of long sequences is possible with the state prediction approach. Note further that in this example exact prediction of the next state is targeted, i.e. a classical associative memory task. Comparing the rather different tasks of learning input-driven attractor dynamics that generalize to new inputs (see Sec. 4.3.3), or the storage of sequences set up by discrete patterns (this section) shows the unifying character of the state prediction approach.

### 4.3.5 Conclusion

At the advent of dynamical approaches to cognition (see [135] for instance), a lot of attention was directed to the qualitative behavior of RNNs. The main goal was to understand their dynamics, for instance by building up equivalence classes of dynamics and bifurcation manifolds [137], [136], and then construct networks with desired dynamics on demand. This approach is tightly bound to the analytic analysis of dynamical systems, which, unfortunately, is not feasible even for small networks and consequently restricts network construction. The state prediction paradigm in contrast is not restricted to small networks or by architectural assumptions like only piecewise constant inputs. The state prediction approach to program observed dynamics into a parameterized network model circumvents descending a gradient, is efficient to compute, and enables the one-shot construction of dynamical systems. Shaping of transient and attractor dynamics is unified by the state prediction paradigm in a generic framework.

In the next chapter, the state prediction approach is exploited for regularization of random projection networks to cope with error amplification of output feedback dynamics. Before I tackle the output feedback stability problem, a related approach for reservoir regularization is introduced and compared to the state prediction methodology.

## 4.4 Constrained regularization of reservoir networks

In this section, another approach related to State Prediction is introduced that is explicitly derived for regularization of input-driven recurrent neural networks. Based on a Gaussian prior for the network weights, the main idea is that the regularization is additionally constrained to implement a previously harvested network state trajectory. This approach emphasizes the regularization of the network and therefore this explicit regularization approach is conceptually complementary to the State Prediction method where regularization is introduced as additional cost to the State Prediction equation.

Technically, I start with a Gaussian prior for the network parameters, which is natural in many respects: Gaussian distributed network parameters (i) are energy efficient, (ii) are found in biological neural networks [70], and (iii) improve generalization of a model to new data because over-fitting is prevented. This is particularly important in the context of random projection methods: As the recurrent dynamics are fixed after initialization, the question of how to guarantee generalization to new inputs is intuitively even more serious and in practice there is often considerable variance in performance with respect to initialization. However, a Gaussian prior distribution of network parameters alone will force all parameters to be zero when the regularization is not constrained to a target behavior. I constrain the regularized network to implement recorded state sequences following the reservoir computing idea of *harvesting states*. In doing so, recurrence is cut off and the regularization can be accomplished by solving a linear system of equations without risking bifurcations during adaptation.

First, the reservoir network model in its simplest variant, the ESN without output feedback, is recalled. A slightly adopted notation is used in this section to simplify the further calculations. Then, the constrained optimization problem using the Lagrange multiplier method is formulated. In a series of experiments, it is shown that the method is capable to reimplement the network dynamics with regularized weights from a previously harvested state trajectory. That means the regularized network also behaves similar in comparison to the original network when it is excited with novel inputs not used for training. Finally, reservoir regularization improves task-specific generalization on a combined prediction and non-linear sequence transduction task.

### 4.4.1 The basic reservoir network model

Consider the recurrent neural network model depicted in Fig. 4.5. Input, reservoir and output neurons are denoted by $\mathbf{x} \in \mathbb{R}^D$, $\mathbf{h} \in \mathbb{R}^R$ and $\mathbf{y} \in \mathbb{R}^O$, where $D$, $R$ and $O$ are the respective numbers of input, reservoir and output neurons. The network is governed by discrete dynamics

$$\mathbf{h}(k+1) = \sigma\left(\mathbf{W}^{inp}\,\mathbf{x}(k) + \mathbf{W}^{res}\,\mathbf{h}(k)\right), \tag{4.5}$$

$$\mathbf{y}(k+1) = \mathbf{W}^{out}\,\mathbf{h}(k), \tag{4.6}$$

where $\sigma(\cdot)$ are sigmoidal activation functions $h_i(k) = \sigma_i(a_i(k))$ that are applied component-wise to the network activity $\mathbf{a}(k) = \mathbf{W}^{inp}\,\mathbf{x}(k-1) + \mathbf{W}^{res}\,\mathbf{h}(k-1) \in \mathbb{R}^R$. The outputs $\mathbf{y}(k)$ are a linear projection of the reservoir state $\mathbf{h}(k-1)$. We collect all reservoir excitation weights in

$$\mathbf{W}^{net} = \begin{pmatrix}\mathbf{W}^{inp} & \mathbf{W}^{res}\end{pmatrix} \in \mathbb{R}^{R\times N} \tag{4.7}$$

in order to simplify further calculations. Accordingly, the combined input-reservoir state at time step $k \in \mathbb{N}$ is denoted by $\mathbf{s}(k) = (\mathbf{x}(k)^T, \mathbf{h}(k)^T)^T \in \mathbb{R}^N$ where $N = D + R$.

### 4.4.2 The Lagrangian approach to reservoir regularization

Consider the Lagrangian

$$L(\mathbf{W}^{net}, \lambda) = R(\mathbf{W}^{net}) - C(\mathbf{W}^{net}, \lambda), \tag{4.8}$$

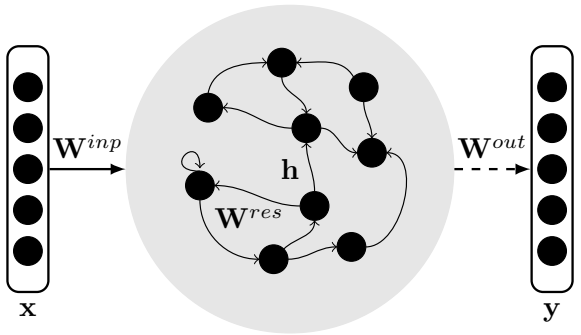



Fig. 4.5: The Echo State Network with reservoir neurons $\mathbf{h}(k)$ that are excited by the input signal $\mathbf{x}(k)$. The network outputs $\mathbf{y}(k)$ are *read out* from the reservoir state.

where $R(\mathbf{W}^{net})$ regularizes the solution by punishing large weights. Let $R(\mathbf{W}^{net})$ take the simple quadratic form

$$R(\mathbf{W}^{net}) = \frac{1}{2}\sum_{i=1}^{R}\sum_{j=1}^{N}\left(w_{ij}^{net}\right)^2, \tag{4.9}$$

which corresponds to a Gaussian prior for the weights $\mathbf{W}^{net}$. $w_{ij}^{net}$ denotes the connection strength from neuron $j$ to neuron $i$.

The second term in (4.8) is a set of constraints $c_i(k)=0$ with Lagrange multipliers $\lambda_i(k)$ per neuron $i$ and time step $k$:

$$C(\mathbf{W}^{net}, \lambda) = \sum_{k=1}^{K}\sum_{i=1}^{R}\lambda_i(k)c_i(k). \tag{4.10}$$

While the regularizer $R(\mathbf{W}^{net})$ in (4.8) forces the weights to be Gaussian distributed, the constraints $c_i(k) = 0$ enforce that the network weights implement the desired, harvested state trajectory. The network update equation (4.5) can be used to derive a constraint for each neuron $i$ and time step $k$:

$$\begin{aligned} a_i(k+1) &= \mathbf{w}_i^{net}\mathbf{s}(k) \\ \Leftrightarrow 0 &= \mathbf{w}_i^{net}\mathbf{s}(k) - a_i(k+1) \equiv c_i(k), \end{aligned}$$

where $\mathbf{w}_i^{net}$ denotes the $i$-th row of matrix $\mathbf{W}^{net}$. Note that $\mathbf{a}(k+1)$ and $\mathbf{s}(k)$ are assumed to be given and fixed for $k = 1, \ldots, K$, which allows to resolve the recurrent dependency $a_i(k+1) = \mathbf{w}_i^{net}(\mathbf{x}(k)^T, \sigma(\mathbf{a}(k))^T)^T$. The minus sign in (4.8) is due to the fact that the overall Lagrangian $L$ is minimized. In the following, the optimality of weights $\mathbf{W}^{net}$ is understood in the sense that $\mathbf{W}^{net}$ minimizes the Lagrangian $L$.

### 4.4.3 Derivation of the optimal network parameters

This section outlines the derivation of the closed-form solution for the optimal network weights $\mathbf{W}_{opt}^{net}$. Basically, a candidate solution $\mathbf{W}_{can}^{net}$ is derived first that depends on the Lagrange multipliers $\lambda \in \mathbb{R}^{K\times R}$. Then, the primary problem of optimizing $L(\mathbf{W}^{net}, \lambda)$ is transformed into its dual form which only depends on $\lambda \in \mathbb{R}^{K\times R}$. The dual problem boils down to a linear system of equations which can be solved efficiently and allows to calculate the optimal network weights $\mathbf{W}_{opt}^{net}$.

Necessary for an extremum of $L$ is

$$\frac{\partial L}{\partial w_{ij}^{net}} = \frac{\partial L}{\partial \lambda_i(k)} = 0$$

for $i=1,\ldots,R$, $j=1,\ldots,N$ and $k=1,\ldots,K$. The candidate that optimizes the Lagrangian $L$ is obtained by partial derivation of $L$ with respect to $w_{ij}^{net}$:

$$w_{ij}^{net} = \sum_{k=1}^{K} \lambda_i(k) s_j(k), \qquad \forall i=1,\ldots,R \quad \wedge \quad j=1,\ldots,N. \tag{4.11}$$

The candidate solution for $w_{ij}^{net}$ now depends on the Lagrange multipliers associated with neuron $i$ for all time steps $k$.

Substituting all $w_{ij}^{net}$ in equation (4.8) with the candidate solution (4.11) yields the dual form of the primary problem which only depends on $\lambda$. Then, partial derivation with respect to a specific $\lambda_m(l)$ yields the following equation:

$$\frac{\partial L_{dual}}{\partial \lambda_m(l)} = a_m(l+1) - \sum_{k=1}^{K} \lambda_m(k) \sum_{j=1}^{N} s_j(l) s_j(k) = 0. \tag{4.12}$$

The detailed derivation of this step is carried out in Appendix A.1.

The equation (4.12) can be reformulated in matrix notation using

$$\lambda_m = (\lambda_m(1), \ldots, \lambda_m(K))^T \tag{4.13}$$

and

$$\mathbf{a}_m = (a_m(2), \ldots, a_m(K+1))^T \tag{4.14}$$

for all reservoir neurons $m = 1, \ldots, R$. Then, (4.12) becomes the linear system of equations

$$\mathbf{C}\lambda_m = \mathbf{a}_m, \tag{4.15}$$

where

$$\mathbf{C} = \begin{pmatrix} \mathbf{s}(1)^T\mathbf{s}(1) & \ldots & \mathbf{s}(1)^T\mathbf{s}(K) \\ \vdots & & \vdots \\ \mathbf{s}(K)^T\mathbf{s}(1) & \ldots & \mathbf{s}(K)^T\mathbf{s}(K) \end{pmatrix}. \tag{4.16}$$

Solving the system (4.15) for each neuron $m \in R$ yields a set of Lagrange multipliers $\lambda_m$ that solves the dual problem. Finally, rewriting equation (4.15) and (4.11) yields a closed-form solution for $\mathbf{W}_{opt}^{net}$. Defining

$$\mathbf{S} = \begin{pmatrix} \mathbf{s}(1)^T \\ \vdots \\ \mathbf{s}(K)^T \end{pmatrix} \text{ and } \lambda = \begin{pmatrix} \lambda_1(1) & \ldots & \lambda_R(1) \\ \vdots & & \vdots \\ \lambda_1(K) & \ldots & \lambda_R(K) \end{pmatrix},$$

equation (4.11) writes for all $i=1,\ldots,R$ and $j=1,\ldots N$

$$(\mathbf{W}_{can}^{net})^T = \mathbf{S}^T\lambda. \tag{4.17}$$

Now, solve (4.15) for $\lambda$ by

$$\lambda = \mathbf{C}^{-1}\mathbf{A}, \tag{4.18}$$

where

$$\mathbf{A} = \begin{pmatrix} a_1(2) & \ldots & a_R(2) \\ \vdots & & \vdots \\ a_1(K+1) & \ldots & a_R(K+1) \end{pmatrix}.$$

**Algorithm 4.3** Constrained regularization of input-driven RNNs

**Require:** initialize network
**Require:** set $k=1$
1: **for** $k < K$ **do**
2: inject external input $\mathbf{x}(k)$ into the network
3: update network state according to (4.5)
4: save $\mathbf{x}(k), \mathbf{h}(k)$ and $\mathbf{a}(k)$
5: $k=k+1$
6: **end for**
7: update network weights $\mathbf{W}^{net}$ according to (4.19)

Combination of (4.17) and (4.18) yields the closed-form expression

$$(\mathbf{W}_{opt}^{net})^T = \mathbf{S}^T\mathbf{C}^{-1}\mathbf{A} = \mathbf{S}^T\left(\mathbf{S}\mathbf{S}^T\right)^{-1}\mathbf{A}. \tag{4.19}$$

The right hand side of (4.19) is the right pseudo-inverse of $\mathbf{S}$, i.e. $\mathbf{S}\mathbf{S}^T\left(\mathbf{S}\mathbf{S}^T\right)^{-1} = \mathbb{1}$. This right-sided solution is obtained since the number of Lagrange multiplier, the free variables here, is $dim(\lambda) = K \cdot R$ which exceeds the number of constraining equations $K$. There are generally infinitely many solutions in this case, and the pseudo-inverse solution is the one that minimizes $||\mathbf{W}^{net}||$. Algorithm 4.3 presents the network optimization process.

### 4.4.4 Discussion of the solution

The main contribution of this section is the analytic, closed-form solution for regularized weights $\mathbf{W}_{opt}^{net}$ of an input-driven recurrent network. The derivation is based on a constrained regularization approach which punishes large reservoir weights $\mathbf{W}^{net}$ and constrains the solution to implement a harvested, i.e. recorded, state trajectory. The derivation of a closed-form solution for the network weights $\mathbf{W}_{opt}^{net}$ is feasible because of the combination of three ingredients: (a) the quadratic regularizer $R(\mathbf{W}^{net})$ that allows to have an explicit expression for the candidate solution $\mathbf{W}_{can}^{net}$, (b) the idea of harvesting states used for the constraints $C(\mathbf{W}^{net}, \lambda)$, which cuts off the recurrent dependence of the network state $\mathbf{h}(k+1) = \sigma(\mathbf{W}^{net}\,\mathbf{s}(k))$ by assuming the recorded states $\mathbf{s}(k)$ for $k = 1, \ldots, K$ as fixed, and (c) the differentiation of internal reservoir and read-out weights, which is natural in many respects [72].

Generally, the extremum $\mathbf{W}_{opt}^{net}$ is a global minimum of $L$ because the Lagrangian $L$ is convex in the parameters $\mathbf{W}^{net}$ due to the quadratic regularizer $R(\mathbf{W}^{net})$ and the constraints $C(\mathbf{W}^{net}, \lambda)$ that depend only linearly on the parameters $\mathbf{W}^{net}$. Furthermore, (4.15) is an inhomogeneous linear system of equations and the optimal solution does either (i) not exist at all, (ii) is not unique, or in the best case (iii) the system is uniquely solvable. On the one hand, $\mathbf{C}$ is a square and symmetric matrix allowing efficient matrix inversion techniques. On the other hand, problems (i) and (ii) can occur depending on the concrete $\mathbf{C}$, for example if $\mathbf{C}$ has not full rank. Unfortunately, $\mathbf{C}$ is typically rank deficient because $rank(\mathbf{C}) = rank(\mathbf{S}) = R$ for $K > R$. I present a common technique to cope with rank deficient matrices $\mathbf{C}$ in the next section. Moreover, this indicates that the optimization based on a harvested state trajectory depends on how well the recorded trajectory captures the dynamic network behavior. I expect the regularized network to implement the same dynamic behavior on new inputs depending on how well the training data captures the overall dynamic behavior of the initial network.

Note also that the solution for each neuron depends on the same matrix $\mathbf{C}$. Specialization of neurons is defined by the inhomogeneous part of the linear system (see the right hand side of equation (4.15)). The solution can be understood as a temporal decorrelation of the network activations, because the Lagrange multipliers and thus the optimal weights depend on the inverse of the temporal state correlation matrix $\mathbf{C}$.

Another interpretation of the solution (4.19) can be obtained by utilizing that $\mathbf{A} = \mathbf{S}(\mathbf{W}^{net})^T$ since $\mathbf{a}(k+1) = \mathbf{W}^{net}\,\mathbf{s}(k-1)$. We obtain

$$\mathbf{W}_{opt}^{net} = \mathbf{W}^{net}\mathbf{S}^T\left(\mathbf{S}\mathbf{S}^T\right)^{-1}\mathbf{S}, \tag{4.20}$$

where now the optimal reservoir weights are expressed in terms of the initial weights $\mathbf{W}^{net}$. The optimized reservoir weights $\mathbf{W}_{opt}^{net}$ are the projections of the initial weights $\mathbf{W}^{net}$ onto the row space of $\mathbf{S}$. Directions of the initial weights perpendicular to the row space of $\mathbf{S}$, i.e. projections of the weights onto the null space of $\mathbf{S}$, are removed from the initial weight matrix. This geometric interpretation of the reservoir optimization process can be grasped intuitively: Weight contributions in the null space of the actually visited reservoir states are removed in order to reduces the overall weight norm while still implementing the harvested state sequence.

### 4.4.5 Regularization of the Lagrange multipliers

Solving the linear system of equations (4.15) can cause numerical difficulties in case it is ill-conditioned. A common approach to prevent such numerical problems is to also regularize the Lagrange multipliers [141]. I punish large values for $\lambda_i(k)$ by introducing an additional term $R(\lambda)$ to the Lagrangian (4.8):

$$R(\lambda) = \gamma\frac{1}{2}\sum_{k=1}^{K}\sum_{i=1}^{R}\lambda_i(k)^2. \tag{4.21}$$

Scaling the regularization by $\gamma$ allows to smoothly adjust the contribution of the regularizer. Derivation of the optimal solution is straight forward and we yield a slightly modified form of $\mathbf{C}$:

$$\tilde{\mathbf{C}} = \mathbf{C} + \gamma\mathbb{1}.$$

The closed-form solution (4.19) adapts to

$$(\mathbf{W}_{opt}^{net})^T = \mathbf{S}^T\left(\mathbf{S}\mathbf{S}^T + \gamma\mathbb{1}\right)^{-1}\mathbf{A}. \tag{4.22}$$

This solution is similar to ridge regression or Tikhonov regularization, where $\gamma$ corresponds to the Tikhonov factor.

### 4.4.6 Proof of concept and method properties

In this section, the feasibility of the Lagrangian approach to reservoir regularization is demonstrated on a simple example that focuses on the reimplementation of a state trajectory. I face the following questions: Does the optimized network reimplement the harvested state trajectory and does the network behave similarly when it is excited with a different input trajectory? How does the reimplementation of the state trajectory depend on the number of reservoir neurons and the number of training samples?

#### Setup

The network is excited with a two dimensional random signal sampled from a uniform distribution in $[-1,1]^2$. Thus the reservoir network has two input neurons $D=2$. First, a randomly initialized reservoir with reservoir weights $\mathbf{W}_{ini}^{net}$ is created. Then, the network is excited with input trajectories $\mathbf{x}^{train}(k)$ and $\mathbf{x}^{test}(k)$. The resulting reservoir state trajectories $\mathbf{H}_{ini}^{train}$ and $\mathbf{H}_{ini}^{test}$ are recorded. Application of the regularization method (4.19) results in a new set of weights $\mathbf{W}_{opt}^{net}$. I excite the regularized network with the same input trajectories $\mathbf{x}^{train}(k)$ and $\mathbf{x}^{test}(k)$ again and compare the new state trajectories $\mathbf{H}_{opt}^{train}$ and $\mathbf{H}_{opt}^{test}$ with the original trajectories $\mathbf{H}_{ini}^{train}$ and $\mathbf{H}_{ini}^{test}$. I also investigate the absolute change of the weight matrices $\mathbf{W}^{inp}$ and

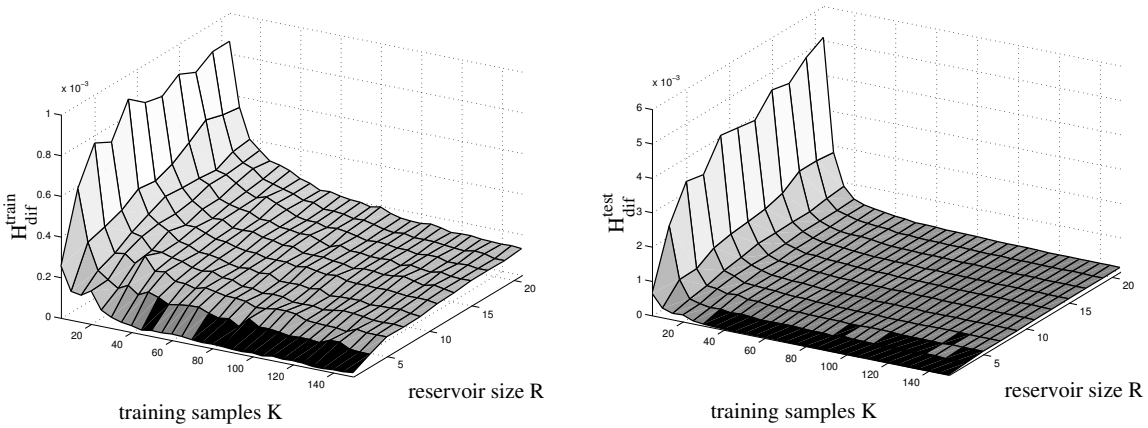



Fig. 4.6: Difference $H_{dif}$ of the reservoir state trajectories for the training set (left) and test set (right) as function of the number of reservoir neurons $R$ and number of training samples $K$.

$\mathbf{W}^{res}$ before and after regularization. Details of the regularized reservoir weights are discussed later on. To account for the random reservoir initialization, the results are averaged over 10 different network weight initializations.

In order to explore the amount of samples needed to reimplement a network behavior sufficiently, I increase the number of training samples from 5 to 150 with step width 5. The test set consists of 1000 samples. In addition, the reservoir network size $R$ is increased from 1 to 21 neurons with step width 2. The hyperbolic tangent function $h_i(k) = tanh(a_i(k))$ for $i = 1, \ldots, R$ is used to calculate the activations of the reservoir neurons. The reservoir weights $\mathbf{W}^{net}$ are initialized with values from a uniform distribution in the range $[-0.1, 0.1]$. For reservoir regularization by (4.22), the regularization parameter $\gamma = 0.001$ is used. Using $\gamma = 0$ will cause numerical instabilities when the temporal state correlation matrix $\mathbf{C}$ is rank deficient, whereas $\gamma \gg 0$ causes the solution to be inaccurate. Therefore, I use a small constant to assure numerical stability of the matrix inversion and do not optimize $\gamma$ explicitly by performing a line search for instance.

### Results

We first focus on the reimplementation of the state dynamics. Fig. 4.6 shows the difference

$$H_{dif} = \sqrt{\frac{1}{K \cdot R} ||\mathbf{H}_{ini} - \mathbf{H}_{opt}||^2} \tag{4.23}$$

between the trajectories generated by the original and the regularized network for the training set (left) and test set ($K = 1000$) (right). I average the difference between the state trajectories with respect to the number of reservoir neurons $R$ and the number of samples $K$ in order to obtain comparable results over different reservoir and training set sizes. Fig. 4.6 indicates that a small training set results in a not well reimplemented state trajectory, whereas for a sufficient amount of samples the state difference $H_{dif}$ is very small. This confirms the expectation I derived from the matrix inversion needed by Algorithm 4.3. In addition, the comparative small errors on the (huge) test set show that the dynamics rather than only the harvested state trajectory are reimplemented. I hypothesize that the precise reimplementation of the dynamic behavior is due to the very well sampled network dynamics: In this example, the network is driven by a low-dimensional random signal covering the whole input space and almost all possible transitions therein. However, Algorithm 4.3 reimplements the network dynamics accurately also for novel input series, if enough training samples are provided.

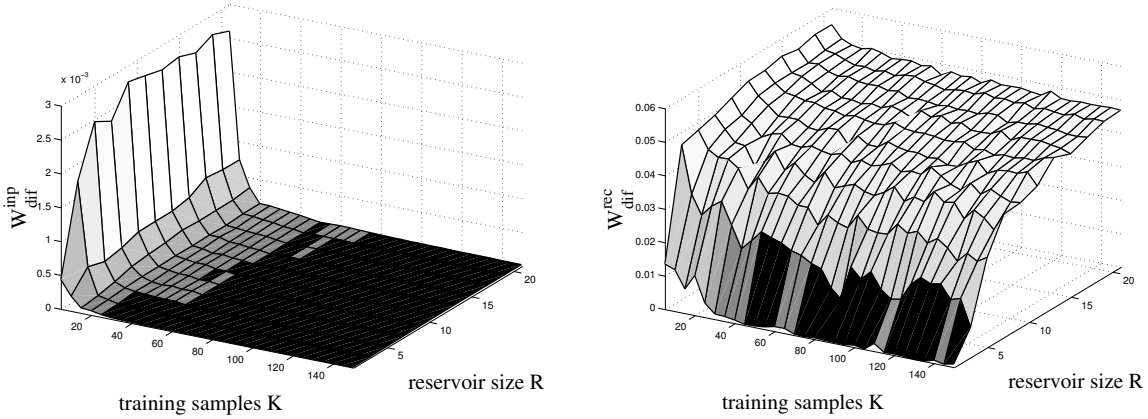



Fig. 4.7: Difference $W_{dif}$ of the reservoir weight matrices $\mathbf{W}^{inp}$ (left) and $\mathbf{W}^{res}$ (right) as function of the number of reservoir neurons $R$ and number of training samples $K$.

Fig. 4.7 shows the difference

$$W_{dif} = \sqrt{\frac{1}{rows \cdot columns} ||\mathbf{W}_{ini} - \mathbf{W}_{opt}||^2}$$

between the weight matrices before and after optimization. I show the difference $W_{dif}$ for input weights $\mathbf{W}^{inp}$ and recurrent weights $\mathbf{W}^{res}$ separately. Again, I average over the number of reservoir neurons to keep $W_{dif}$ comparable for different reservoir sizes. A different behavior for input and recurrent reservoir weights can be observed in Fig. 4.7: While the input weights are not changed significantly in case of sufficient sampling of the dynamics (see Fig. 4.7 (left)), the reservoir weights are significantly changed by Algorithm 4.3 irrespective of the number of training samples (see Fig. 4.7 (right)). Note that the dramatically changed reservoir weights implement very similar state trajectories (see Fig. 4.6). I argue that increasing the reservoir size increases the degrees of freedom to implement the same state dynamics as well. Reservoir regularization then choses the weight configuration with the smallest norm. In case of very small networks, the state trajectory can only be implemented with the initial reservoir weights, whereas larger networks can implement basically the same network state trajectory with significantly different reservoir weights. This strongly indicates that the mapping from reservoir parameters to (input-driven) state dynamics is redundant, i.e. at least two parameter settings implement the same dynamics for the given set of (task related) input sequences. The presented method selects the solution which satisfies both, the regularizer $R(\mathbf{W}^{net})$ and the constraints $C(\mathbf{W}^{net}, \lambda)$, to reimplement the harvested state trajectory. Thereby, the Lagrangian $L$ provides a means to prefer the regularized network to all other possible reservoir weights $\mathbf{W}^{net}$ that implement the same dynamics.

### 4.4.7 Task-specific generalization and network properties

In this section, I demonstrate the utility of the constrained reservoir regularization method in an illustrative input-output mapping scenario. I show the regularization effect on the task-specific performance but also elaborate on network properties in more detail. The following questions are tackled: Does the network implement the same dynamics with a different set of weights also in case of rather smoothly changing time-series inputs? Does the proposed method actually produce weights that are Gaussian distributed, and how does the optimization technique affect the network structure?

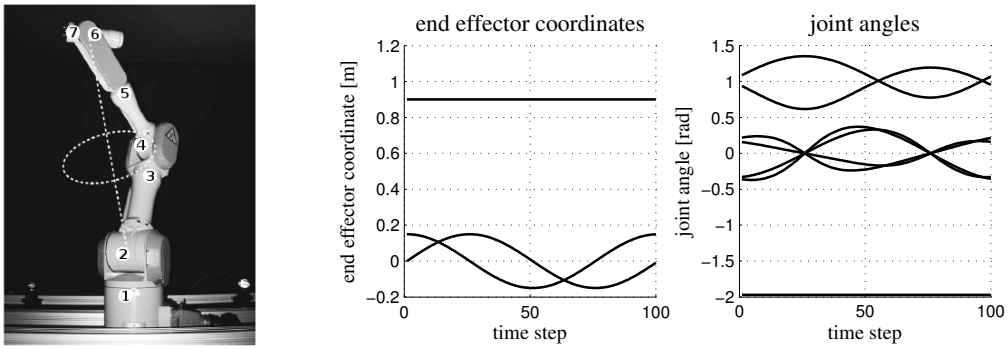



Fig. 4.8: The multi purpose robot arm PA-10 (left). Circular motion in task and joint space (right).

### Setup

We focus on a temporal input-output mapping task from the robotics domain. The network is trained to approximate the inverse kinematics of the PA-10 general purpose manipulator (shown in Fig. 4.8 (left)) from a set of training examples. Inputs $\mathbf{x}(k) = (x_1(k), x_2(k), x_3(k))^T$ are Cartesian coordinates of the end effector at time step $k$. Outputs $\mathbf{y}(k) = (y_1(k), \ldots, y_7(k))^T$ are the corresponding joint angles of the robot arm. The inverse kinematic function $IK$ of the PA-10, which maps end effector coordinates to joint angles, is not unique due to redundant degrees of freedom of its kinematic structure. For generation of supervised training data, I use a consistent redundancy resolution scheme such that the inverse kinematics are convex. Note further that the network has to predict joint angles from end effector coordinates at the previous time step. Hence, the task combines prediction and sequence transduction (instantaneous time-series mapping), both particularly suited for recurrent neural networks. Note further that the network predicts joint angles from target positions in a feed-forward manner without knowledge about the current state of the robot arm.

I define a desired end effector trajectory and solve the inverse kinematics analytically according to [142, 143] in order to generate training data. The end effector trajectories are defined by

$$x_1(k) = r\ sin(\omega k), \qquad x_2(k) = r\ cos(\omega k), \qquad x_3(k) = 0.9,$$

where I set $\omega = \frac{\Delta}{r}$ to control the speed of the end effector independently from the circle's radius $r$. A single period of such a circular pattern is shown in Fig. 4.8 (right) in joint and task space (end effector coordinates). For training, the radius $r$ is set to $0.1m$ and the end effector speed $\Delta$ to 0.005. The network gets end effector coordinates $\mathbf{x}(k)$ in decimeters to excite the network with values in a reasonable range, while results are presented in meter for convenience. The seven joint angles $\mathbf{y}(k)$ are specified in radians. I use six pattern periods ($k = 1, \ldots, K = 6\frac{2\pi r}{\Delta}$) for training, preceded by two periods to washout initial transients. For testing the reimplementation of state dynamics and the performance on the input-output mapping task, ten pattern periods are used. The reservoir network has three input neurons which correspond to the three dimensional end effector coordinates, and seven output neurons which correspond to the seven joints of the robot arm. The reservoir itself consists of $R = 50$ neurons with hyperbolic tangent activation functions $h_i(k) = tanh(a_i(k)), i = 1, \ldots, R$. The reservoir weights $\mathbf{W}^{net}$ are initialized with values sampled from a uniform distribution in range $[-0.1, 0.1]$. For network optimization, I use $\alpha^{out} = 0.01$ in (3.14) and $\gamma = 0.001$ in (4.22).

I basically follow the same test procedure as described in the previous section. However, this time generalization is tested more systematically by varying either the radius $r$ or the end effector speed $\Delta$. The radius $r$ is increased from $0m$ to $0.25m$ in $1cm$ steps, where I set $K = 10\frac{2\pi 0.1}{0.005}$ in case of $r = 0$. The end effector speed $\Delta$ is increased from 0.001 to 0.03 with step width 0.001. We also investigate properties of the regularized weight matrices $\mathbf{W}^{inp}_{opt}$ and $\mathbf{W}^{res}_{opt}$

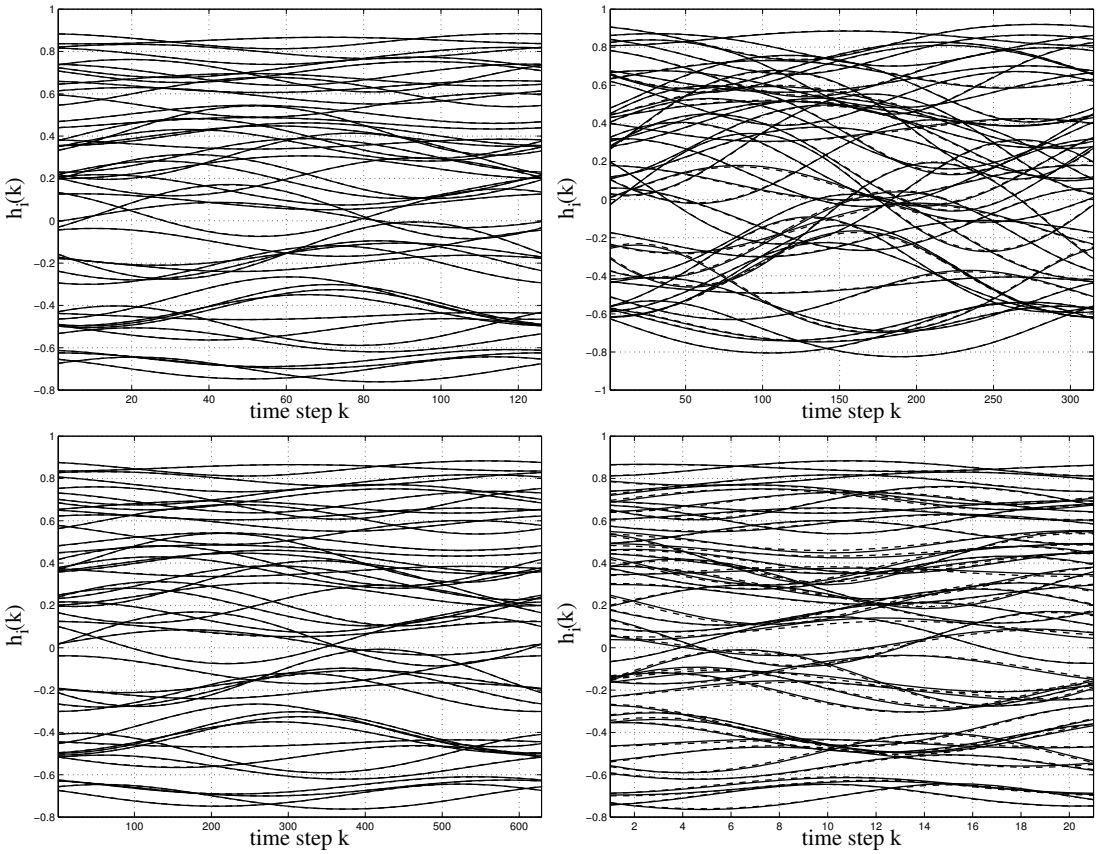



Fig. 4.9: Reservoir state trajectories $\mathbf{h}(k)$ for the initial network (solid lines) and the regularized network (dashed lines): training pattern (top left), maximal radius $r = 0.25$ (top right), minimal end effector speed $\Delta = 0.001$ (bottom left), and maximal end effector speed $\Delta = 0.03$ (bottom right). The regularized network produces qualitatively the same dynamics.

in more detail. To account for the random reservoir initialization, all experimental results are averaged over 100 different network weight initializations.

### Reimplementing dynamics

Fig. 4.9 (top left) shows a single period of the state trajectories for the training pattern before and after adaptation of $\mathbf{W}^{net}$. Fig. 4.9 further shows the state trajectories for the maximal radius $r = 0.25m$ (top right), minimal (bottom left) and maximal (bottom right) end effector speeds $\Delta$. Visual investigation indicates that the same state trajectories are implemented with different weights (overlapping solid and dashed lines in Fig. 4.9). This impression is confirmed by measuring the difference of the trajectories for the whole sequence length by (4.23) again.

Fig. 4.10 shows the difference $H_{dif}$ of the state trajectories before and after reservoir regularization as function of the radius $r$ (left) and end effector speed $\Delta$ (right). Surprisingly, the optimized network behaves almost identically for a wide range of radii and end effector speeds. The difference $H_{dif}$ of state trajectories is clearly minimized for the training pattern ($r = 0.1$ and $\Delta = 0.005$), while setting the radius to zero results even in a lower state difference. The latter is particularly interesting with respect to network stability and worth further attention in the future: The case of $r = 0$ shows that the regularization method reimplements also the attractor states, at least for the constant input $\mathbf{x} = (0, 0, 0.9)^T$, although this network behavior is not captured by the training data. However, the dynamics are qualitatively very similar for the entire set of input patterns (see Fig. 4.9 and Fig. 4.10). The very similar state trajectories confirm the results presented in the previous section also for smooth input time-series. Again,

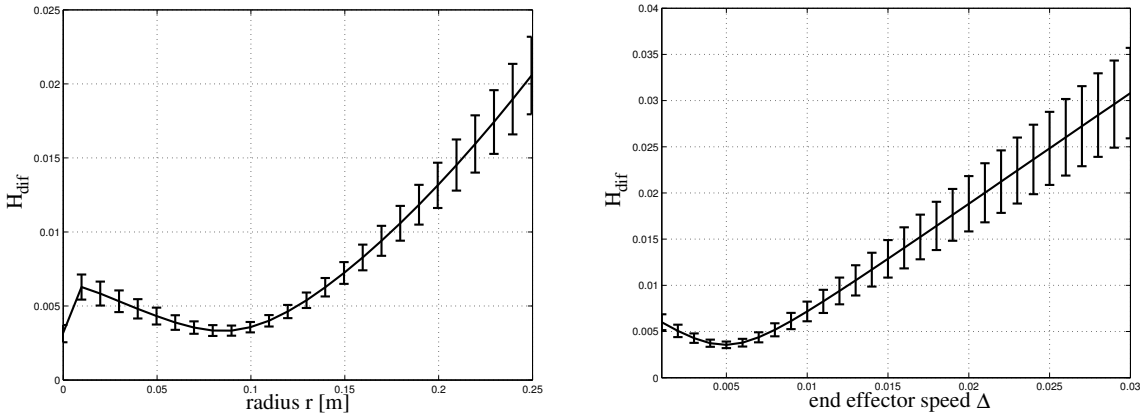



Fig. 4.10: Averaged difference $H_{dif}$ of reservoir state trajectories with standard deviations before and after adaptation by Algorithm 4.3 as function of the circle's radius (left) and effector speed $\Delta$ (right).

the results indicate that the regularization method implements the same dynamic behavior to a far extent rather than only reimplementing the same state trajectory used for training.

### Network properties

Fig. 4.11 shows the weight matrix $\mathbf{W}^{net}$ (top) of a single reservoir and the distribution of values (bottom) before (left) and after (right) network regularization by Algorithm 4.3. The weights have changed significantly while they still implement the same dynamics (see previous section). Remarkable is the structure of $\mathbf{W}^{net}_{opt}$: A decomposition into external excitation weights $\mathbf{W}^{inp}$ and recurrent connections $\mathbf{W}^{res}$ is clearly visible in Fig. 4.11 (top right). This decomposition occurred throughout the experiments. Note that the decomposition is not explicitly enforced by the Lagrangian $L$. I hypothesize that the decomposition roots in a spatial (and non-temporal) contribution of the input to the reservoir state trajectory (implemented by $\mathbf{W}^{inp}_{opt}$) and a temporal contribution by modeling the transients of the reservoir state trajectory with the recurrent weight matrix $\mathbf{W}^{res}_{opt}$. The recurrent weights show a further substructure: Positive self-recurrent connections $w^{res}_{ii}$ are visible in Fig. 4.11 (top right) for almost all reservoir neurons $i = 1, \ldots, R$.

We further analyze the distribution of the adapted weights by regarding each entry of the weight matrix as a sample. Then, applying statistical tests is straight forward: I test the hypothesis of Gaussian and Laplacian distributed weights by the Kolmogorov-Smirnov test, where the tested probability density function is fitted to the data first. I also present mean, standard deviation and excess Kurtosis of the weights. The latter measures how Gaussian the distribution is. Zero excess Kurtosis indicates a Gaussian distribution, while higher values stand for leptokurtic or sparse distributions. Platykurtic distributions have negative excess Kurtosis. However, the excess Kurtosis is sensitive to outliers and it is not a very confident measure: Non-Gaussian distributions can also have zero excess Kurtosis.

Regarding the weight distribution, I expect a Gaussian distribution with zero mean due to the regularizer $R(\mathbf{W}^{net})$. Tab. 4.2 lists the results averaged over 100 different network initializations. All properties are presented for the entire optimized reservoir matrix $\mathbf{W}^{net}_{opt}$ and its parts $\mathbf{W}^{inp}_{opt}$ and $\mathbf{W}^{res}_{opt}$ to account for the functional decomposition into reservoir excitation and recurrent weights we observed in Fig. 4.11 (top right). Tab. 4.2 shows that the reservoir optimization technique results in a weight distribution with approximately zero mean irrespective of the considered matrix part. The excess Kurtosis differs strongly for the sub-matrices $\mathbf{W}^{inp}_{opt}$ and $\mathbf{W}^{res}_{opt}$ as well as for the combined matrix $\mathbf{W}^{net}_{opt}$. While the Kurtosis of the input matrix $\mathbf{W}^{inp}_{opt}$ indicates a rather Gaussian distribution, the recurrent weights $\mathbf{W}^{res}_{opt}$ are leptokurtic which indicates a sparse distribution. The standard deviations and the statistical tests confirm this

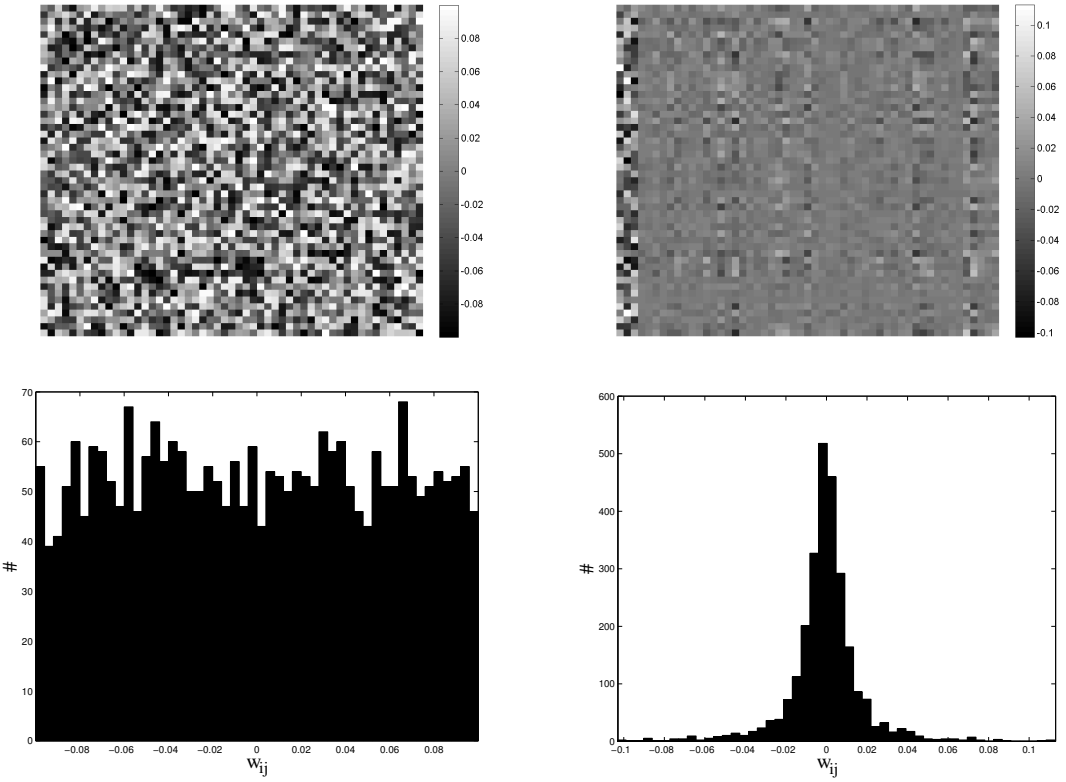



Fig. 4.11: Weight matrix $\mathbf{W}^{net}$ before (top left) and after (top right) adaptation by Algorithm 4.3 and corresponding weight distributions (bottom row). The weights have changed dramatically, while they still implement the same dynamics (see Fig. 4.10).

indication: The last two rows of Tab. 4.2 show the percentage of optimized weight matrices that do not allow to reject the null-hypothesis of a Gaussian or Laplacian distribution (using a significance level of 0.05). The input weights are Gaussian distributed for most of the networks, while the recurrent weights are mostly Laplacian distributed. The statistical tests do not yield meaningful results for the entire reservoir weight matrix $\mathbf{W}_{opt}^{net}$. I suggest the reason for a not always Gaussian distributed weight matrix is the trade-off between regularization $R(\mathbf{W}^{net})$ and the constraint $C(\mathbf{W}^{net}, \lambda)$ to implement a given state trajectory. However, the sparse distribution of the recurrent reservoir weights $\mathbf{W}_{opt}^{res}$ justifies the utility of sparse network initializations often used for reservoir networks [70, 86, 144]. Note that the optimization approach does not explicitly enforce a sparse distribution such as the Laplacian distribution. Sparseness could also be achieved by explicitly using a $L_1$-norm for regularization, but then a closed-form solution is not feasible anymore and iterative optimization techniques have to be applied.

I conclude that the introduced method reimplements the same dynamics with a different, regularized set of weights. The results show a functional decomposition into input and recurrent weights. This input-reservoir decomposition is complementary to the decomposition into reservoir and read-out weights found earlier in [72], the basic idea of reservoir computing.

### Input-output mapping

The experiments presented in the previous sections demonstrate the feasibility of reimplementing the same dynamics with regularized reservoir weights. We now investigate the effects of reservoir regularization on the task-specific performance: Does the regularized network perform better on the inverse kinematics? Firstly, I do not expect any improvement as long as the state trajectories of the original and the regularized network do not differ significantly. However,

| | $\mathbf{W}_{opt}^{inp}$ | $\mathbf{W}_{opt}^{res}$ | $\mathbf{W}_{opt}^{net}$ |
|---|---|---|---|
| **mean** | $-0.00017 \pm 4 \cdot 10^{-3}$ | $0.00022 \pm 2 \cdot 10^{-4}$ | $0.00019 \pm 3 \cdot 10^{-4}$ |
| **std. dev.** | $0.054 \pm 1 \cdot 10^{-3}$ | $0.012 \pm 7 \cdot 10^{-4}$ | $0.017 \pm 6 \cdot 10^{-4}$ |
| **Kurtosis** | $-0.97 \pm 0.13$ | $3.71 \pm 1.45$ | $9.52 \pm 1.34$ |
| **Gaussian** | 97% | 9% | 0% |
| **Laplace** | 77% | 96% | 7.5% |

Tab. 4.2: Different properties of the regularized reservoir weight matrices with standard deviations. Results are averaged over 100 network initializations.

reservoir regularization should result in a smaller variance of the task error. For inputs that result in a slightly different state trajectory, I expect the performance to increase.

The read-out weights $\mathbf{W}^{out}$ are optimized by (3.14) with $\alpha^{out} = 0.01$ in order to approximate the inverse kinematics from the training pattern ($r = 0.1$ and $\Delta = 0.005$). I measure the performance of the networks on the inverse kinematics in task space by calculating the error

$$RMSE = \sqrt{\frac{1}{K}\sum_{k=1}^{K} ||\mathbf{x}(k) - FK(\hat{\mathbf{y}}(k))||^2}, \tag{4.24}$$

where $FK : \mathbf{y} \mapsto \mathbf{x}$ is the actual forward kinematic function that maps the joint angles estimated by the network back into the task space. In doing so, the error of the end effector position is presented in Cartesian coordinates conveniently. Note that ambiguities of the inverse kinematic function are implicitly resolved by the training data.

Fig. 4.12 shows the errors and standard deviations of the initial and regularized networks. The result confirms that there is no significant difference of the task-specific error (4.24) for the radii and end effector speeds where the regularized network behaves very similar to the original network (compare Fig. 4.10 and Fig. 4.12). Variation of the radius does not seem to have a significant effect on the state difference (Fig. 4.10) and the difference of the task-specific performance (Fig. 4.12) at all. However, the regularized networks actually outperform the original networks slightly on average for different radii $r$ and end effector speeds $\Delta$. Fig. 4.12 shows very small error variances for the initial and regularized networks. This excellent generalization behavior and robustness of learning confirms earlier results [98]. Note that the network is only trained on the kinematics for radius $r = 0.1m$ and end effector speed $\Delta = 0.005$. The network regularization has no significant impact on the generalization for different radii indicating that the distributed input representation in the reservoir state is well-suited for spatial generalization. However, reservoir regularization improves the variance of the task-specific generalization error significantly with respect to varying speeds of the input signal (see Fig. 4.12 (bottom right)). This is interesting, since the regularization results in a sparsified network connectivity. The regularized network seems to be more agile and can follow the faster changing input more easily. This result points out the task-specific benefit of the reservoir regularization method.

### Conclusion

A constrained regularization approach for input-driven recurrent neural networks was derived using a Gaussian prior distribution for reservoir weight regularization and constraining the solution to implement recorded network dynamics. This section tackled the question whether state dynamics of random networks can be reimplemented. The method is capable of reimplementing the dynamics of an initial network to a far extent if sufficient training samples are provided.

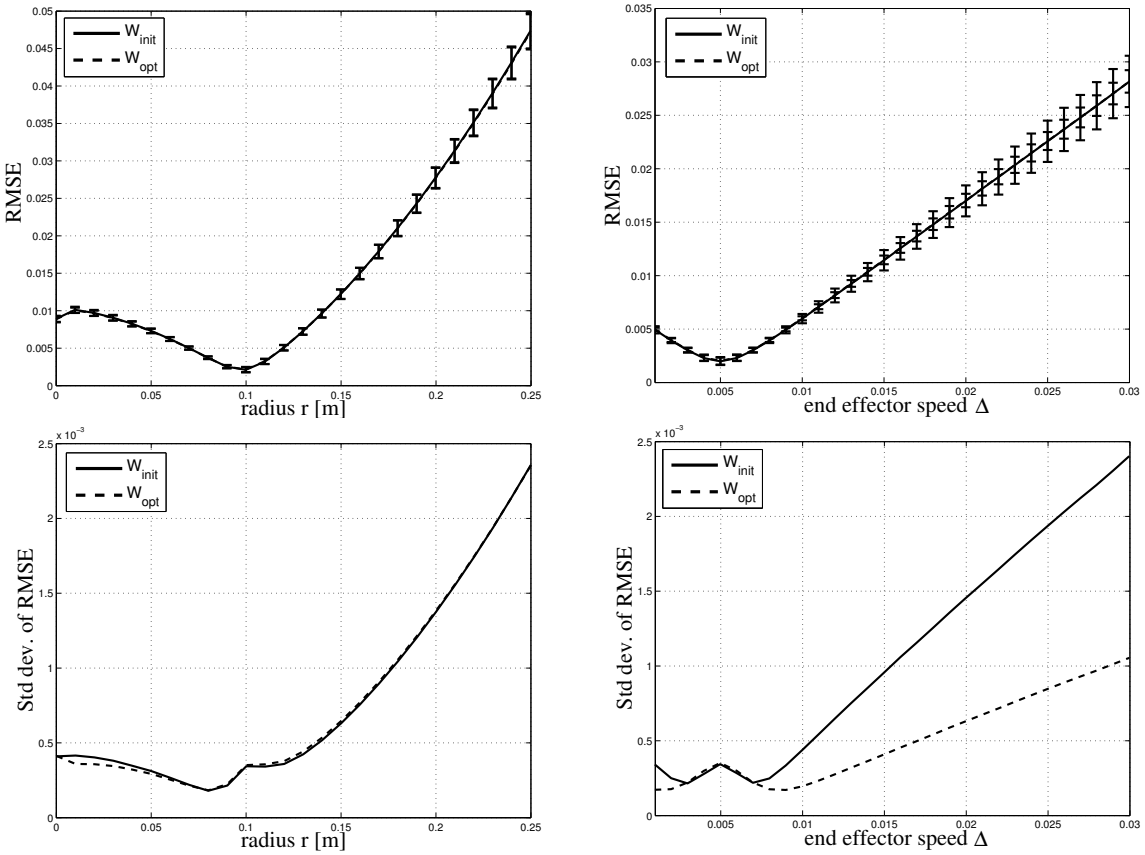



Fig. 4.12: Generalization performance of original and regularized networks for variation of the radius (left column) and variation of the end effector speed $\Delta$ (right column). Absolute errors (top row) and standard deviations (bottom row) are shown.

The derived method further provides a generic criterion to prefer a reservoir to another one: Optimality is based on the reservoir weight matrix itself in contrast to a supervised error criterion like for most other approaches on reservoir optimization. The regularized networks show a typical decomposed structure dividing the reservoir weights into reservoir excitation and recurrent connections. While the reservoir excitation matrix has Gaussian distributed entries, the recurrent reservoir weight matrix shows a sparsified structure. This result sheds light on the internal reservoir structure and gives reason to typical rules of thumb for reservoir initialization. I also compared the task-specific generalization of the regularized networks with the initial ones: Since the state trajectories differ only slightly, no significant error reduction can be achieved. However, error variances are reduced, in particular when the speed of the input signal is increased.

The Lagrangian approach to reimplement harvested reservoir state sequences emphasizes regularization. State prediction solves a very similar problem but the approach focuses more generally on programming input-driven recurrent neural network dynamics: While the Lagrangian method regularizes an initial network with a hidden layer utilizing only input sequences, State Prediction programs desired state sequences without the need of an initial network. In the next section, it is shown that State Prediction is closely related to the Lagrangian approach and that implementing reservoir regularization by State Prediction solves the aforementioned drawbacks of the Lagrangian method (see Sec. 4.4.4).

## 4.5 Constrained regularization and State Prediction

In a first step, it is shown that State Prediction can serve as reservoir regularization technique similar to the explicit Lagrangian approach to reservoir regularization. Then, the relation of both, State Prediction and constrained reservoir regularization, as well as their advantages with respect to numerical computation are discussed in this section.

### 4.5.1 Reservoir regularization based on State Prediction

The state prediction approach can be adopted for regularization of reservoir networks. Basically, the state prediction equation (4.2) can be understood as the reimplementation constraints of the Lagrangian approach. In addition, we can regularize the solution to the state prediction problem. This can be simply accomplished by increasing the weight $\beta$ of the regularization term $||\mathbf{W}^{net}||^2$, which was already assumed in (4.4). In this way, reimplementation constraints can be reduced in weight in favor of smaller, regularized reservoir connections.

The regularized reservoir weights then are

$$(\mathbf{W}_{reg}^{net})^T = \left(\mathbf{S}^T\mathbf{S} + \beta\mathbb{1}\right)^{-1}\mathbf{S}^T\mathbf{A}, \tag{4.25}$$

where $\mathbf{S} = (\mathbf{s}(1), \ldots, \mathbf{s}(K))^T$ with $\mathbf{s}(k)^T = (\mathbf{x}(k)^T, \mathbf{h}(k)^T)$ are the harvested states together with the inputs $\mathbf{x}(k)$. The reservoir targets $\mathbf{A} = (\mathbf{a}(2), \ldots, \mathbf{a}(K+1))^T$ with $\mathbf{a}(k+1) = \mathbf{W}^{inp}\mathbf{x}(k) + \mathbf{W}^{res}\mathbf{h}(k)$ are the neural activities before application of non-linear activation functions $\sigma$ one time step ahead. Note that reimplementing harvested network dynamics by State Prediction obviates the application of the inverse activation function to the state sequence because network activities $\mathbf{a}(k)$ can directly be collected. Similar to read-out learning with Tikhonov regularization, $\beta$ corresponds to the Tikhonov factor which trades off reimplementation constraints against weight norm.

Reservoir Regularization by State Prediction is conducted before read-out learning: First, reservoir states are harvested. Then, the reservoir is regularized according to (4.25). In a last step, read-out learning is applied, which can optionally be computed with freshly harvested states to fully exploit the approximation capabilities of the network.

### 4.5.2 Equivalence of constrained regularization and State Prediction

Interestingly, both, the Lagrangian approach to reservoir regularization and the one based on State Prediction, are equivalent in the limit of $\beta = \gamma \to 0$. It holds that

For any $n \times m$ matrix $A$,

$$P_A = \lim_{\delta\to 0}(A^TA + \delta^2 I)^{-1}A^T = \lim_{\delta\to 0} A^T(AA^T + \delta^2 I)^{-1} \tag{4.26}$$

always exists. (see [145], page 19)

In the context of State Prediction and Lagrangian reservoir regularization, $A$ is the row-wise collection of the harvested network states (compare (4.25) and (4.22)). Both, the left and right pseudo-inverse of $A$, are then used for the regression of the network weights according to targeted neural activities in the next time step. This equivalence makes intuitively sense, because in both cases the solution fulfills the reimplementation constraints as accurately as possible and the solution has the minimal norm. However, the Lagrangian approach introduces too many auxiliary variables such that the system of equations becomes underdetermined. While both approaches are, in the limit case, equivalent, they differ in practical execution with respect to computational efficiency.

### 4.5.3 Computational complexity

The solution of the state prediction equation is based on the left pseudo-inverse that is calculated using typically more equations than parameters. That means the system of equations is overdetermined which circumvents the numerical difficulties of the matrix inversion of the Lagrangian approach. Considering the numerical difficulties and rather inefficient calculations of the Lagrangian approach to reservoir regularization discussed in Sec. 4.4, reservoir regularization based on State Prediction is a computationally significantly more efficient:

In case of reservoir regularization by (4.20), a $K \times K$ matrix has to be inverted, where $K$ is the number of time steps the reservoir is simulated. In case of reservoir regularization by (4.25), a $N \times N$ matrix has to be inverted, where $N$ is the number of input and reservoir neurons in the network. Typically, $K \gg N$ and therefore (4.25) is preferred in the remainder of this thesis. The Lagrangian approach conceptually emphasizes the idea of regularization, whereas State Prediction is superior with respect to numerical and computational properties.

### 4.5.4 Conclusion

A unifying methodology of implementing recurrent neural network dynamics in one shot was presented and demonstrated to be feasible in a series of experiments. Regularization of the model parameters, i.e. the recurrent neural network weights, is in the focus of the Lagrangian approach to State Prediction. However, the introduced Lagrange multipliers render the optimization problem ill-posed with respect to the number of parameters in relation to the number of training samples. The regularized least squares formulation used in the state prediction formalism reduces the computational complexity of the regularization process and is utilized for reservoir regularization in the remainder of this thesis.

Regularization generally diminishes the effective parameters of the model, i.e. is a model selection mechanism, and improves generalization. Although a gain in performance for Echo State Networks without output feedback on a sequence transduction task was rather restricted, regularization of recurrent networks is a versatile tool for stabilizing trained output feedback dynamics which is demonstrated in the next chapter. I conclude that the well-behaved networks obtained in this chapter by applying the presented methodology already shows that learning of complex dynamics is possible in one shot. The absence of networks that, for example, drive into saturation or exhibit totally different dynamics than the desired dynamics shows the robustness of the method and "stability" of the regularized networks. The connection of regularization, learning and stability is in the focus of the next chapter. In particular, it is shown that reservoir regularization stabilizes output feedback dynamics.

Chapter 5

# Output feedback stabilization by regularization

Output feedback is crucial for autonomous and parameterized pattern generation with reservoir networks. Read-out learning affects the output feedback loop and can lead to error amplification. Regularization is therefore important for both, generalization and reduction of error amplification. In this chapter, I show that regularization of the reservoir and the read-out layer reduces the risk of error amplification, mitigates parameter dependency and boosts the task-specific performance of reservoir networks with output feedback. The deeper connection between regularization of the learning process and stability of the trained network is discussed.

## 5.1 Regularization and stability in reservoir networks with output feedback

Output feedback stability, like it was introduced in Sec. 3.4.2, has been achieved in reservoir computing by using heuristics [118], regularization of the read-out learning [36], or online learning techniques like BPDC [23] and FORCE learning [101]. However, the deeper connection between the applied mechanism and stability was not investigated systematically, although Jaeger already observed a relation of weight norm and stability in a series of experiments [118] and the work in [36] is based on the implicit hypothesis that regularization stabilizes networks with output feedback. An exception is the mechanism proposed in [123], which rescales the network weights during online learning such that input-output stability is maintained. However, the stabilization mechanism does not directly inform learning and is therefore not suited for offline training. I show the explicit relation between stability of the output feedback-driven network dynamics and read-out regularization of the learning process in theory and experiments.

I additionally use a generic and efficient regularization approach for the inner reservoir to further reduce the risk of error amplification. Regularization of recurrent neural networks based on gradient descent was already proposed by Wu and Moody in [35]. They also related regularization to input-output stability in the context of prediction tasks. I pursue a one-shot learning method that is based on the recently introduced reservoir regularization for input-driven recurrent networks ([146] and Sec. 4.4). The idea of reservoir regularization is to reimplement a previously harvested reservoir state sequence with the smallest weights possible, i.e. introducing a Gaussian prior distribution with zero mean for the reservoir weights while constraints require to implement the desired state sequence. In this chapter, the improved version of the original constrained optimization formulation of reservoir regularization in [146] is used (see discussion in Sec. 4.5).

Very recently, Jaeger took a similar approach in order to compile the functionality of an external controller into the reservoir [128]. An early variant of this methodology was already described by Mayer and Browne in [147] to "internalize" an input signal into the reservoir network. Both approaches, however, do not tackle the problem of error amplification and do not use regularization of the learning process to promote stability in networks with output feedback. I show that the minimized weight norm of regularized networks is particularly useful for reservoirs with output feedback: Networks with smaller weight norm are more robust against erroneous feedback but also allow to relax regularization constraints of the read-out learning which in turn increases the task-specific performance.

The effects of regularization on output feedback stability are analyzed in a sequence transduction and autonomous pattern generation scenario. Note that the results can then be generalized to bidirectional, associative reservoir computing: Feeding back inputs (driving the outputs), or feeding back outputs (driving the inputs) is structurally equal with respect to the learning and stability. Therefore, it is sufficient to show that regularization stabilizes the output feedback dynamics of an unidirectional reservoir with output feedback connections. However, the analysis and experiments in this chapter are restricted to learning of single-valued functions (but multi-dimensional), i.e. the training data contains no multiple solutions and the unidirectional reservoir is not required to learn multi-stable output feedback dynamics. Read-out learning is modified to train multi-stable output feedback dynamics in Chapter 7. Nevertheless, the general connection of learning, regularization and stability transfers also to multi-stable network dynamics.

First, the Echo State Network (ESN) with output feedback is briefly recapitulated including a decent notation that shows the parallels between reservoir regularization and regularized read-out learning. Then, a series of experiments shows the stabilizing effect of regularization and the relation between stability, learning and regularization are discussed.

## 5.2 Echo State Networks with output feedback

We focus on ESN architectures with output feedback as depicted in Fig. 5.1. The network comprises a recurrent reservoir network of non-linear neurons which are connected randomly with small connection strengths. We consider the discrete and synchronous recurrent network dynamics

$$\mathbf{h}(k+1) = \sigma\left(\mathbf{W}^{inp}\mathbf{x}(k) + \mathbf{W}^{res}\mathbf{h}(k) + \mathbf{W}^{fdb}\mathbf{y}(k)\right), \tag{5.1}$$

$$\mathbf{y}(k+1) = \mathbf{W}^{out}\mathbf{h}(k). \tag{5.2}$$

$\mathbf{x}, \mathbf{h}$ and $\mathbf{y}$ are the input, reservoir and output neurons, where $\mathbf{h}$ is obtained by applying sigmoidal activation functions $\sigma(\cdot)$ component-wise. All connections to the reservoir are subsumed in

$$\mathbf{W}^{net} = \left(\mathbf{W}^{inp}\ \mathbf{W}^{res}\ \mathbf{W}^{fdb}\right).$$

In this chapter, the cases of autonomous pattern generation ($\mathbf{W}^{inp} = 0$) and input-driven pattern generation ($\mathbf{W}^{inp} \neq 0$) are considered.

## 5.3 Regularization of the read-out layer

Originally, learning of ESNs is restricted to the supervised optimization of the read-out weights $\mathbf{W}^{out}$ in order to infer a desired input-to-output mapping from a set of training examples. The reservoir states $\mathbf{h}(k)$ as well as the target outputs $\mathbf{t}(k)$ are recorded for $k = 1, \ldots, K$ time steps

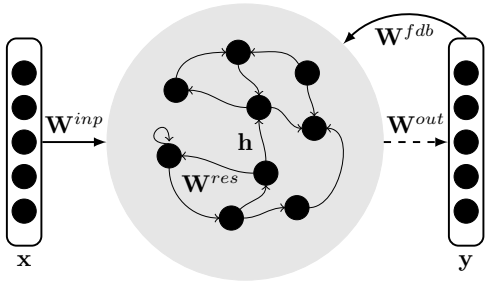



Fig. 5.1: Echo State Network with output feedback connections $\mathbf{W}^{fdb}$.

in a reservoir state matrix

$$\mathbf{H} = (\mathbf{h}(1), \ldots, \mathbf{h}(K))^T \in \mathbb{R}^{K \times R} \quad \text{and target matrix} \quad \mathbf{T} = (\mathbf{t}(1), \ldots, \mathbf{t}(K))^T,$$

respectively. Optimization of the read-out weights $\mathbf{W}^{out}$ is conducted with respect to the task-specific mean square error

$$E^{task} = \frac{1}{K} \sum_{k=1}^{K} ||\mathbf{t}(k) - \mathbf{W}^{out}\mathbf{h}(k)||^2 \; + \; \alpha ||\mathbf{W}^{out}||^2 \tag{5.3}$$

including an additional regularization term. The optimal read-out weights are then determined by the least squares solution

$$(\mathbf{W}^{out})^T = \left(\mathbf{H}^T\mathbf{H} + \alpha \mathbb{1}\right)^{-1} \mathbf{H}^T\mathbf{T},$$

where the Tikhonov factor $\alpha \geq 0$ weights the contribution of the regularization term.

Whereas Tikhonov regularization is typically meant to improve generalization of the approximated mapping to novel inputs, the preference for small weights also reduces the potential for error amplification by the output feedback loop [118, 36]. Thus, regularization of the read-out weights is motivated by both, good generalization and output feedback stability. The Tikhonov factor $\alpha$ in (5.3) is tuned to reduce the potential error amplification on the one hand, while implementing the desired output pattern on the other hand. This can be accomplished by conducting a line search over $\alpha$ for each network, where in the test condition the network is output feedback-driven (see Algorithm 5.4 and [36]). Note that tuning of regularization constraints is typically performed on a validation test in order to optimize the generalization ability of the model. In the context of output feedback stability, tuning of $\alpha$ is also meaningful on the training set in order to stabilize the network dynamics.

Note that online learning of the read-out layer, for example by (3.17), has a regularizing effect on the learning even without explicit decay term. Small learning rates act as a smoothing over several training patterns and thus online learning regularizes by "averaging" errors where

**Algorithm 5.4** Line search for read-out regularization

**Require:** training data $(\mathbf{x}(k), \mathbf{t}(k))$ with $k = 1, \ldots, K$, set of $\alpha$'s, network
**Require:** harvest teacher-forced states $\mathbf{H}$
1: **for** $\alpha \in \{10^{-6}, 5 \times 10^{-6}, 10^{-5}, 5 \times 10^{-5}, \ldots, 1.5, 2\}$ **do**
2: read-out learning by $(\mathbf{W}^{out})^T = \left(\mathbf{H}^T\mathbf{H} + \alpha \mathbb{1}\right)^{-1} \mathbf{H}^T\mathbf{T}$ using $\alpha$
3: output feedback-driven testing yields task-specific error $E^{task}(\alpha) = \frac{1}{K} \sum_{k=1}^{K} ||\mathbf{t}(k) - \hat{\mathbf{y}}(k)||^2$
4: **end for**
5: **return** trained reservoir network using $\alpha^{opt}$ that yields smallest error $E^{task}$

offline regression will come up with a more precise solution at the costs of a larger weight norm. This explains why regularization and thus stability is not a big issue in the respective literature on online learning of output feedback dynamics [48, 15, 20, 101, 102, 23]. For offline training of output feedback dynamics, however, regularization is crucial [36, 118].

## 5.4 Reservoir regularization

In addition to the read-out regularization, I propose to also regularize the reservoir in order to mitigate the dependency on the read-out regularization parameter $\alpha$. I apply the previously introduced reservoir regularization method (see Chapter 4) and adopt it to Echo State Networks with output feedback connections in the following. The idea is to express a preference for small weights conditioned on the constraint to reimplement a previously harvested network state sequence. This can be formulated by the mean square error

$$E^{state} = \frac{1}{K} \sum_{k=1}^{K} ||\mathbf{a}(k+1) - \mathbf{W}^{net}\mathbf{z}(k)||^2 \; + \; \beta||\mathbf{W}^{net}||^2 \tag{5.4}$$

with $\mathbf{a}(k+1) = \mathbf{W}^{net}_{ini}\mathbf{z}(k)$, where $\mathbf{W}^{net}_{ini}$ is the initial reservoir weight matrix and $\mathbf{z}(k) = (\mathbf{x}(k)^T, \mathbf{h}(k)^T, \mathbf{t}(k))^T$ is the combined input, reservoir and output vector. I denote the first term of this objective function by *state prediction error* because it demands the optimized weights to model the state transitions of the harvested state sequence. The second term serves as regularizer which is weighted by $\beta$. Note the close relation of the state prediction objective (5.4) to the task-specific objective (5.3): The optimization assumes that the network states are given and fixed, i.e. harvested beforehand, such that recurrence is cut off. We obtain the regularized reservoir weights

$$(\mathbf{W}^{net}_{reg})^T = \left(\mathbf{Z}^T\mathbf{Z} + \beta\mathbb{1}\right)^{-1}\mathbf{Z}^T\mathbf{A}, \tag{5.5}$$

where $\mathbf{Z} = (\mathbf{z}(1), \ldots, \mathbf{z}(K))^T$ are the harvested states together with the inputs $\mathbf{x}(k)$ and target outputs $\mathbf{t}(k)$. The reservoir targets $\mathbf{A} = (\mathbf{a}(2), \ldots, \mathbf{a}(K+1))^T$ are the neural activities before application of non-linear activation functions $\sigma$ one time step ahead. Reservoir regularization is conducted before read-out learning using the initial reservoir weights $\mathbf{W}^{net}_{ini}$, i.e. reservoir states are harvested, the reservoir is regularized and then Algorithm 5.4 is applied.

## 5.5 Regularization stabilizes dynamics and improves performance

In this section, I show that the proposed reservoir regularization facilitates learning in Echo State Networks with output feedback in two complementary scenarios. All results are averaged over 200 independent trials. The reservoirs comprise 50 neurons with $tanh(\cdot)$ activation functions. The weight matrix $\mathbf{W}^{net}$ is randomly initialized according to a uniform distribution in range $[-0.5, 0.5]$. The spectral radius of the reservoir sub-matrix $\mathbf{W}^{res}$ is scaled to 0.9.

### 5.5.1 Learning inverse kinematics

We first consider a combined prediction and sequence transduction task from the robotics domain. The ESN learns the inverse kinematics of a planar robot arm with two degrees of freedom from a training sequence (see Fig. 5.2), i.e. the network has to map Cartesian end effector coordinates to joint angles. A circular and a figure eight-like motion are used as training and test data (compare Fig. 5.2). Each sequence comprises $K = 500$ samples. Joint angles are encoded in a coordinate representation for reasons of normalization, i.e. $\mathbf{y}(q_1, q_2) = (sin(q_1), cos(q_1), sin(q_2), cos(q_2))^T$.

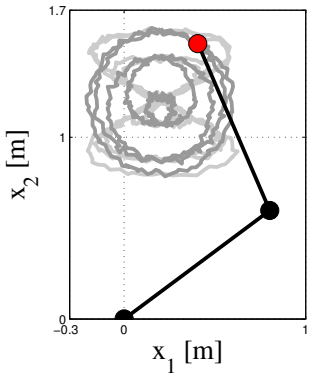



Fig. 5.2: Robot arm with two degrees of freedom. Training and test patterns are shown in gray and light gray, respectively.

### Reservoir regularization stabilizes dynamics

I first show that reservoir regularization stabilizes the output feedback-driven dynamics for a wide range of read-out regularization parameters $\alpha$. I calculate the difference

$$E^{state} = \frac{1}{K \cdot R} ||\mathbf{H} - \mathbf{H}_{fdb}||^2 \tag{5.6}$$

between the state trajectories $\mathbf{H}$ in the teacher-forced case (the network outputs are set to the target values $\mathbf{t}(k)$) and the output feedback-driven case $\mathbf{H}_{fdb}$ (the predicted outputs $\mathbf{y}(k)$ are fed back into the reservoir). Fig. 5.3 (left) reveals that without reservoir regularization the network state trajectory $\mathbf{H}$ in the teacher-forced case is only achieved in the output feedback-driven case if the read-out regularization $\alpha$ is strong enough ($E^{state}_{ini}$). Note that $E^{state}$ is closely related to the output feedback stability criterion introduced in Sec. 3.4.2 by measuring the mean deviation of the output feedback-driven state sequence $\mathbf{H}_{fdb}$ from the teacher-forced state sequence $\mathbf{H}$. The large values of $E^{state}$ in Fig. 5.3 (left) and Fig. 5.5 (left) for $\alpha < 10^{-4}$ thus indicate non output feedback stable networks because the mean deviation is larger than a small $\epsilon > 0$. Reservoir regularization mitigates this dependency on $\alpha$ and makes reservoirs with output feedback more robust against the choice of the parameter $\alpha$ ($E^{state}_{reg}$ shown for $\beta \in \{10^{-6}, 10^{-4}, 10^{-2}\}$ in Fig. 5.3 (left)). The reservoir regularization parameter $\beta$ forces the reservoir weights to have a smaller norm (compare Fig. 5.3 (right)) which damps the amplification of deviations at the output. With reservoir regularization, the network is more robust against deviations of the output trajectory.

### Output feedback stability, classical stability criteria and regularization

In this section, I show that the hypothesized relation of regularization and stability can be made explicit with respect to classical stability criteria. The notion of output feedback stability is discussed in more detail.

We first investigate global asymptotic stability of the output feedback-driven network dynamics depending on regularization of the read-out learning and the inner reservoir. Global asymptotic stability of the output feedback-driven system means that the reservoir state with outputs $(\mathbf{h}(k)^T, \mathbf{y}(k)^T)^T$ gradually approaches an equilibrium as $k \to \infty$ for any initial state $(\mathbf{h}(0)^T, \mathbf{y}(0)^T)^T$ and constant input $\mathbf{x}$ [148, 93, 149], i.e. the output feedback-driven network dynamics have a unique fixed-point attractor depending on the external input $\mathbf{x}$. The output feedback-driven network dynamics are generated by the weight matrix

$$\mathbf{W}^{ofd} = \begin{pmatrix} \mathbf{W}^{res} & \mathbf{W}^{fdb} \\ \mathbf{W}^{out} & \mathbf{0} \end{pmatrix} \in \mathbb{R}^{N \times N},$$

where $N = R + O$. A sufficient criterion for global asymptotic stability is based on the maximal singular value

$$\sigma_{max}(\mathbf{W}^{ofd}) = ||\mathbf{W}^{ofd}|| < 1, \tag{5.7}$$

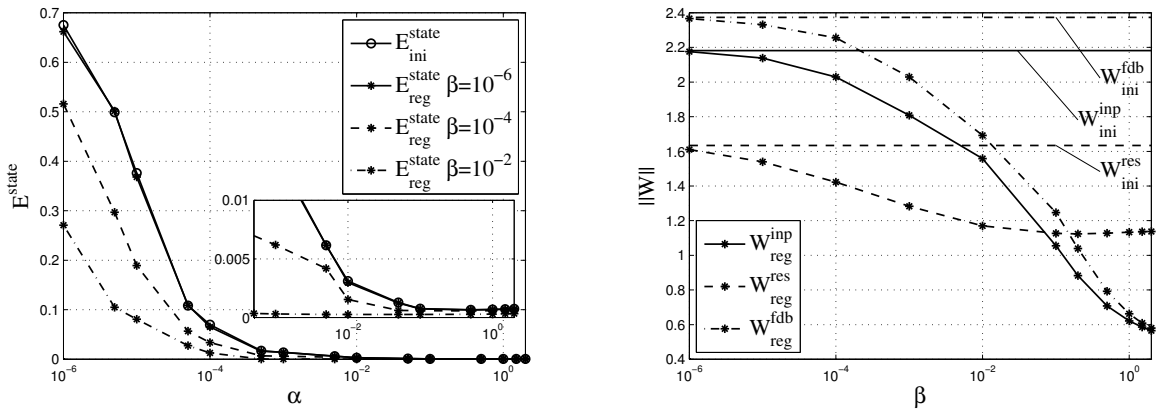



Fig. 5.3: Deviation $E^{state}$ of the output feedback-driven state sequence from the teacher-forced state sequence for the training scenario as function of the read-out regularization parameter $\alpha$ (left). Matrix norm of weights $\mathbf{W}^{inp}$, $\mathbf{W}^{res}$ and $\mathbf{W}^{fdb}$ as function of the reservoir regularization parameter $\beta$ (right). Subscripts indicate whether the networks where regularized ($reg$) before read-out training, or not ($ini$).

which is equal to the spectral norm for the square matrix $\mathbf{W}^{ofd}$ [93, 30]. The left-hand side of (5.7) is plotted in Fig. 5.4 (top left) averaged over all networks. Regularization of the read-out learning strongly reduces the spectral norm of the output feedback-driven networks and thus increases the "degree" of stability. Regularization of the inner reservoir additionally reduces the spectral norm. However, none of the networks is stable with respect to criterion (5.7), but they are output feedback stable for sufficiently large regularization parameters $\alpha$ and $\beta$ (compare Fig. 5.3 (left)). The sufficient criterion based on the spectral norm is very strict, discards possibly stable networks, and proves convergence to a single fixed-point for constant inputs. However, the conservative criterion (5.7) already links weight norm and thus regularization to stability.

Let us further consider a necessary global asymptotic stability criterion based on the spectral radius $\lambda_{max}(\mathbf{W}^{ofd})$ of the output feedback-driven system, i.e. the maximal absolute eigenvalue of matrix $\mathbf{W}^{ofd}$. A spectral radius of $\mathbf{W}^{ofd}$ smaller than unity is necessary for global asymptotic stability but not sufficient [93, 30]. It holds that

$$\lambda_{max}(\mathbf{W}^{ofd}) \leq \sigma_{max}(\mathbf{W}^{ofd}) \qquad [93].$$

Fig. 5.4 (bottom) shows the average spectral radius of the output feedback-driven networks for different read-out and reservoir regularization parameters $\alpha$ and $\beta$. Regularization has also significant impact on the spectral radius and I conclude that regularization increases the degree of stability in the classic sense of global asymptotic stability. Note, however, that the global asymptotic stability criterion is not directly related to the meaning of output feedback stability and I therefore do not expect global asymptotic stability for non-trivial output feedback dynamics: Convergence to a single fixed-point for each input is sufficient but not necessary for output feedback stability. Output feedback stability is non-conservative in the sense that it assigns stability to not globally asymptotically stable networks and it is local because stability is connected to a specific domain of inputs. Output feedback stability ties stability to task-specific inputs and outputs, which makes it useful for practical application to adaptive dynamical systems with output feedback loops. The global asymptotic stability criterion, however, is well-suited to inform learning by means of regularization in a soft constrained optimization manner: Reduction of the spectral norm of $\mathbf{W}^{ofd}$ brings the system closer to the border of stability.

In [123], a sufficient input-output stability criterion was derived from a small gain criterion. Interestingly, this criterion is based solely on matrix norms of the reservoir network's sub-

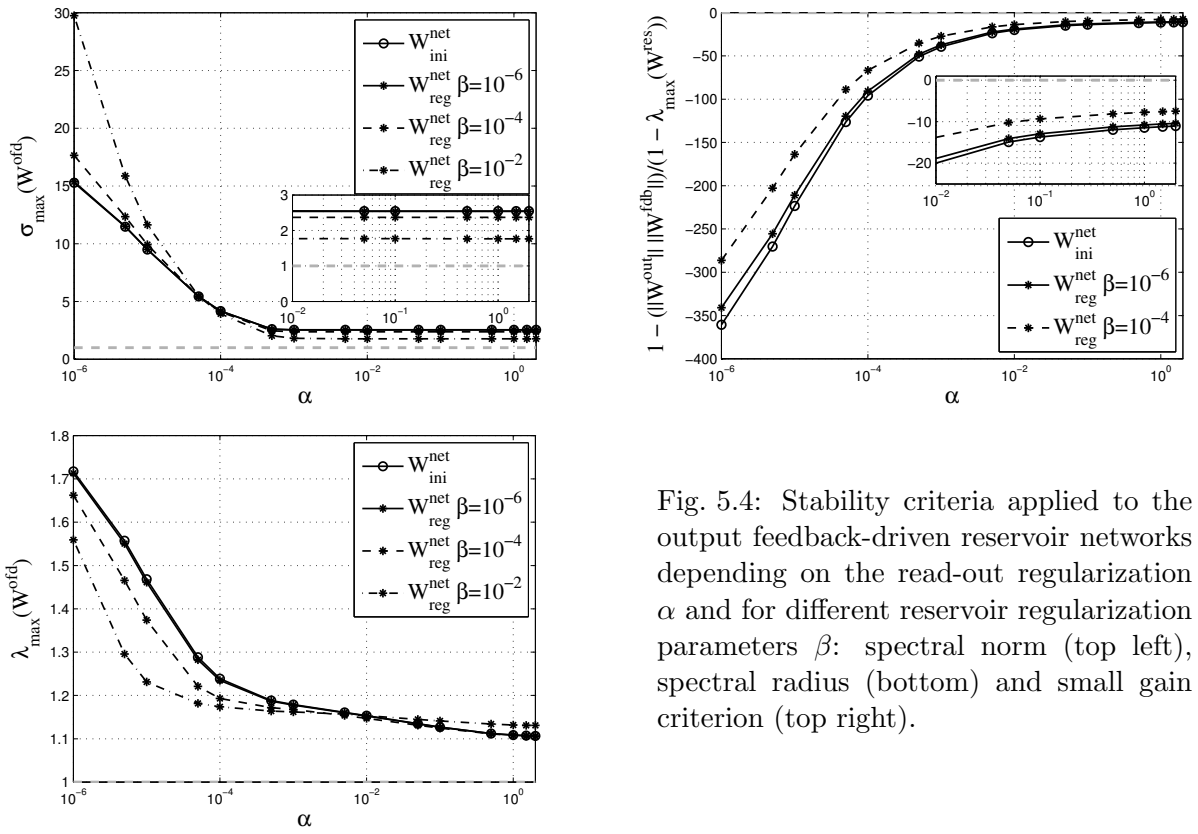



Fig. 5.4: Stability criteria applied to the output feedback-driven reservoir networks depending on the read-out regularization $\alpha$ and for different reservoir regularization parameters $\beta$: spectral norm (top left), spectral radius (bottom) and small gain criterion (top right).

matrices. Stability is guaranteed if the inequality

$$1 - \frac{||\mathbf{W}^{out}|| \, ||\mathbf{W}^{fdb}||}{1 - ||\mathbf{W}^{res}||} > 0 \tag{5.8}$$

holds, where $||\cdot||$ denotes a matrix norm and $||\mathbf{W}^{res}|| < 1$. This criterion directly connects input-output stability of the network with regularization of the weight sub-matrices: A smaller norm of the weights implies a smaller gain of the overall system. For evaluation, I substitute $||\mathbf{W}^{res}||$ by $\lambda_{max}(\mathbf{W}^{res})$ according to the relation $\lambda_{max}(\mathbf{W}^{res}) \leq \sigma_{max}(\mathbf{W}^{res})$ between spectral radius and spectral norm to assure a positive denominator in (5.8). Fig. 5.4 (top right) displays the approximated left-hand side of the stability criterion (5.8) depending on the regularization parameters $\alpha$ and $\beta$. Regularization of both the read-out learning and the inner reservoir brings the network dynamics closer to the border of input-output stability which is consistent with Fig. 5.3 (right) which shows the decreased weight norm with increased reservoir regularization.

I conclude that output feedback stability is an input-specific, non-conservative stability criterion that, in contrast to classical global asymptotic stability and input-output stability criteria, is closer related to task-specific learning and restricted input domains. This section also highlights the deeper connection between regularization and stability. Spectral norm and input-output stability based on the small gain criterion show the connection between stability and regularization explicitly: They demand upper bounds on the weight norms which are part of the learning process in form of regularizers in the objective functions (5.3) and (5.4).

### Reservoir regularization improves performance

Fig. 5.5 (left) clearly points out the connection between learning, regularization and stability. Without output feedback ($\mathbf{W}^{fdb} = \mathbf{0}$), increasing the regularization parameter $\alpha$ increases the training error – there is no interdependency between dynamics and learning. The integration of the read-out learning into the network dynamics by the output feedback loop changes the

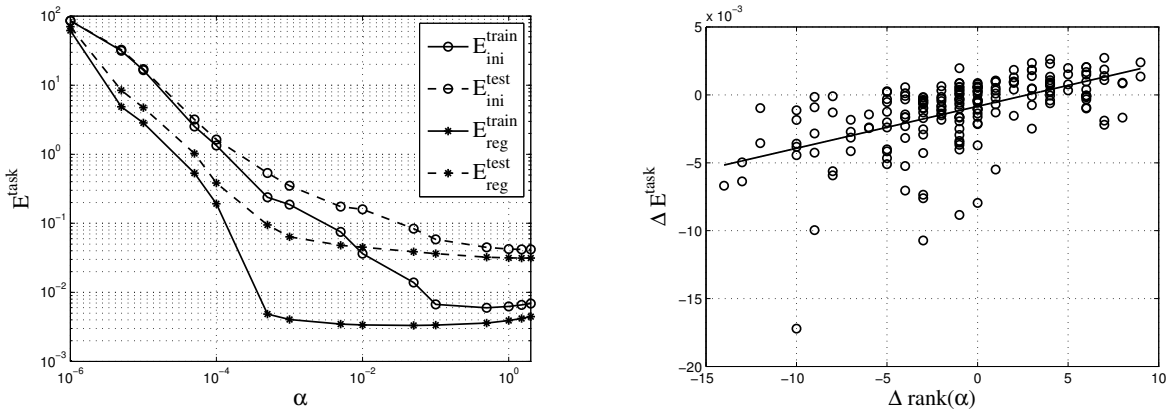


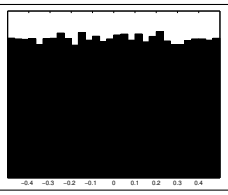
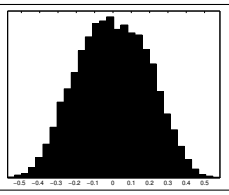
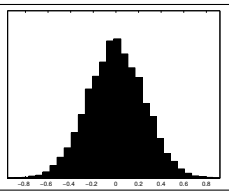
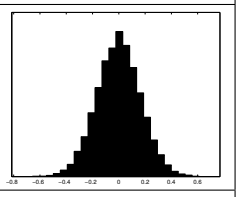
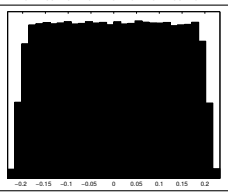
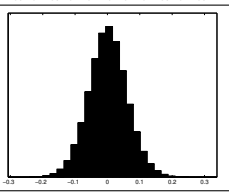
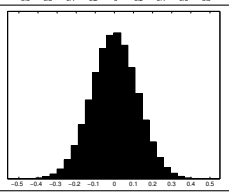
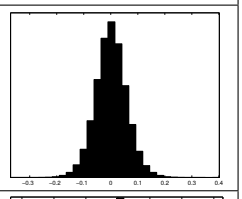
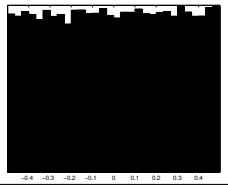
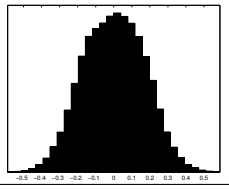
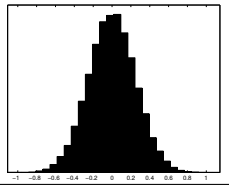
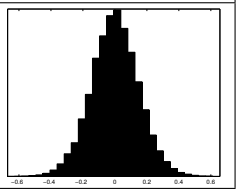

Fig. 5.5: Task-specific error $E^{task}$ on the training and test set as function of the read-out regularization $\alpha$ (left). Change of the task-specific error due to reservoir regularization as function of the changed read-out regularization parameter $\alpha$ when applying Algorithm 5.4 (right). $\beta = 10^{-2}$ is used in both plots.

picture completely: Regularization of the read-out layer decreases the training error because regularization stabilizes the output feedback-driven network dynamics. But regularization also decreases the contribution of the task-specific error term in (5.3) which gives rise to an intimate dependency of regularization of the read-out layer, stability and performance. Regularization of the inner reservoir mitigates this interdependency by shifting stabilization of the dynamics to the reservoir: The regularized reservoir weights reduce the effective gain of the system and cause the grid search Algorithm 5.4 to yield smaller values for $\alpha$ (see Fig. 5.5 (right)). This in turn improves the task-specific performance while keeping the network stable (compare Fig. 5.3 (left) and Fig. 5.5 (left)). I observe a highly significant correlation (Spearman rank correlation test with significance level 0.01) between the change of the read-out regularization parameter $\Delta\alpha = \alpha_{reg}^{opt} - \alpha_{ini}^{opt}$ and the change of the task-specific error $\Delta E^{task} = E_{reg}^{task} - E_{ini}^{task}$ (plotted using ranks for $\Delta\alpha$ in Fig. 5.5 (right)). This means that with reservoir regularization, the read-out regularization can be reduced which consequently results in smaller task-specific errors. The combination of both small task-specific errors and regularized reservoir weights minimizes the risk of error amplification due to the output feedback loop.

### Properties of the regularized network weights

We finally investigate the properties of the regularized network weight matrices. The weight distributions of each sub-matrix for all networks are shown in Tab. 5.1 (first three columns). The weights and their distribution has changed significantly while they still implement the same input-to-state mapping (compare Fig. 5.3 (left)). The initially uniform weight distributions are approximately Gaussian after reservoir regularization.

We investigate statistical properties of the weights for all 200 independent network initializations. I expect a Gaussian distribution of the weights with zero mean due to the regularization term in (5.4). Tab. 5.2 displays different variates of the weights and confirms that the average mean value of the weights is close to zero. I also present the standard deviation of the weights which shows that the reservoir weights $\mathbf{W}^{res}$ have generally more peaked weight distributions. Further, I analyze the distribution of the adapted weights by regarding each entry of a weight matrix as sample. Then, application of statistical tests is straight forward: I test the hypothesis of Gaussian distributed weights by the Kolmogorov-Smirnov test, where the observed samples are compared to a Gaussian probability density function that is fitted to the data first. The last row in Tab. 5.2 shows the percentage of optimized weight matrices that do not allow to reject the null-hypothesis of a Gaussian distribution (using a significance level of 0.05). The

| Weight Matrix | Uniform Distribution Initial | Uniform Distribution Regularized | Gaussian Distribution Initial | Gaussian Distribution Regularized |
|---|---|---|---|---|
| $\mathbf{W}^{inp}$ |  |  |  |  |
| $\mathbf{W}^{res}$ |  |  |  |  |
| $\mathbf{W}^{fdb}$ |  |  |  |  |

Tab. 5.1: Distributions of the network connection weights $\mathbf{W}^{inp}$, $\mathbf{W}^{res}$ and $\mathbf{W}^{fdb}$: Initial distributions (uniform in second column, Gaussian in fourth column) and after reservoir regularization using $\beta = 10^{-2}$ (third and fifth column).

statistical tests indicate Gaussian distributions for the input, reservoir and feedback weights for most of the network initializations. This confirms the results presented earlier in Sec. 4.4.

In Tab. 5.1, the last two columns show that reservoir regularization recomputes the weights also when starting initially from Gaussian distributions. The respective results for the criterion (5.6) are appended to the thesis (see Appendix A.2) and show that reservoir regularization stabilizes the output feedback dynamics also in the case of initially Gaussian weight distributions. Essentially, we obtain the same qualitative result: Reservoir regularization mitigates the dependence of output feedback stability on the read-out regularization parameter $\alpha$ (see Appendix A.2). That this is also the case for initially Gaussian weight distributions emphasizes the effectiveness of reservoir regularization to recompute the weights with smaller norm to reduce the gain of the output feedback loop.

### 5.5.2 Autonomous generation of a unit circle

To complement the results for the input-driven case, I also show that reservoir regularization improves the stability in case of autonomous pattern generation. In this scenario, the networks have no input neurons and shall produce a unit circle pattern at their outputs autonomously. All other parameters are initialized as described above. Five periods of a unit circle $\mathbf{y}(k) = (sin(\omega k), cos(\omega k))^T$ with $\omega = \frac{2\pi}{200}$ are presented for training, where the first four periods are used as wash-out phase. I apply reservoir regularization with $\beta = 10^{-2}$ and optimize $\alpha$ with respect to the training error according to Algorithm 5.4. For testing, the networks have to reproduce the circle running freely for $10^6$ steps within an error margin of 5% (compare Fig. 5.6). I neglect

| | $\mathbf{W}^{inp}$ | $\mathbf{W}^{res}$ | $\mathbf{W}^{fdb}$ |
|---|---|---|---|
| **mean** | 0.0014 | 0.0019 | −0.00003 |
| **std. dev.** | 0.19 | 0.061 | 0.17 |
| **Gaussian** | 100% | 94.5% | 99.5% |

Tab. 5.2: Statistics of the weight matrices $\mathbf{W}^{inp}$, $\mathbf{W}^{res}$ and $\mathbf{W}^{fdb}$ after reservoir regularization using $\beta = 10^{-2}$.

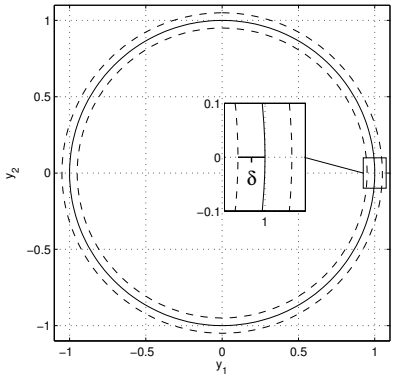



Fig. 5.6: Pattern generation scenario: The circular pattern has to be generated for $10^6$ steps within the error bound $\delta$.

phase lags typically occurring during long-term recursive prediction, i.e. monitor the orbital stability of the output pattern $\hat{\mathbf{y}}(k)$ with respect to the target pattern $\mathbf{y}(k)$. Orbital stability can be understood as temporal relaxation of the output feedback stability criterion, which is reasonable for autonomous and parameterized pattern generation.

Without reservoir regularization, only 69% of the networks comply with the 5% error bound for $10^6$ steps, whereas 94% of the networks satisfy this tight criterion if the reservoir is regularized with $\beta = 10^{-2}$. This confirms the previous result that reservoir regularization improves the task-specific performance and makes ESN learning more robust in scenarios where output feedback is crucial. Additionally, I count the number of cycles the networks reproduce the target motion and relate this actual frequency to the frequency of the target pattern. This yields a frequency reproduction factor

$$f = \frac{\text{number of generated periods}}{\text{number of desired periods}},$$

where the number of desired periods is derived from the number of steps per period of the target pattern and the number of time steps the network produced the pattern within the error bound $\delta$. Considering the frequencies of the reproduced patterns, the regularized networks stick much closer to the target frequency than the initial networks: While the initial networks display an average frequency reproduction factor $f_{ini} = 0.63 \pm 0.09$, the regularized networks achieve $f_{reg} = 0.85 \pm 0.05$ which indicates a higher frequency of the reproduced circular pattern on average that matches the target pattern significantly better. I hypothesize that regularization reduces the "inertia" of the network by pruning unnecessary weights that tend to slow down the network dynamics. A similar connection between regularization and network "agility" was already observed when using the Lagrangian approach to reservoir regularization (see [146] and Sec. 4.4).

I also evaluate global asymptotic stability of the networks trained for autonomous generation of the circular pattern. Note that sustained dynamics are required for ongoing pattern generation and thus global asymptotic stability is not directly targeted. However, the trained network dynamics should be as close as possible to the border of global asymptotic stability such that pattern generation is robust against perturbations and error amplification is reduced. Tab. 5.3 shows that regularization decreases the weight norm of the output feedback-driven system significantly (compare $\sigma_{max}(\mathbf{W}^{ofd})$ for initial and regularized networks in Tab. 5.3). On the one hand, regularization of the reservoir brings the value of the spectral norm much closer to the sufficient stability bound. On the other hand, learning keeps connections that are important for sustained network dynamics large: Though the spectral norm is heavily decreased, the spectral radius of the system stays almost equal on average (compare $\lambda_{max}(\mathbf{W}^{ofd})$ for initial and regularized networks in Tab. 5.3). Regularization yields to nearly optimal network configurations in the sense that sustained dynamics are implemented with small gains, i.e. optimizing the global asymptotic stability criterion under the necessary condition that $\lambda_{max}(\mathbf{W}^{ofd}) \geq 1$. Finally, I remark that autonomous pattern generation clearly exemplifies the tight relation of

| | $\sigma_{max}(\mathbf{W}^{ofd})$ | $\lambda_{max}(\mathbf{W}^{ofd})$ |
|---|---|---|
| **ini** | $2.37 \pm 0.11$ | $1.09 \pm 0.03$ |
| **reg** | $1.29 \pm 0.14$ | $1.11 \pm 0.03$ |

Tab. 5.3: Global asymptotic stability criteria of the output feedback-driven networks before and after reservoir regularization.

learning and stability by demanding the ongoing reproduction of an oscillatory pattern trained from an exemplary trajectory.

## 5.6 Balancing contributions by distributing activities

In the previous sections, it turned out that reservoir regularization stabilizes learning of Echo State Networks with output feedback. This methodology is important for robust bidirectional association with ESNs. However, there is another open issue concerning the balancing of contributions of inputs and outputs to the overall reservoir dynamics. In this section, this issue is tackled by explicitly balancing the contributions of inputs and outputs to the reservoir dynamics. To facilitate the further discussion, inputs, outputs and also the reservoir state are more generally referred to as *(input) modalities* in this section.

Typically, the contribution of input and output modalities to the reservoir state is determined by the initialization ranges of the respective weight matrices. However, it is quite difficult to balance these contributions, in particular when input and output signals have different dimensions or energy spectra. Moreover, the contribution of the reservoir to the neural activities depends non-trivially on the scaling of inputs and outputs as well as the reservoir size and its spectral radius. Finally, the reservoir regularization methods as introduced in Sec. 5.4 do not necessarily respect the initial weighting of the diverse input modalities: The regression solution assigns weights to each variable, i.e. input, reservoir or output neuron, in order to predict the desired network activity in the next time step. This can result in an unintended re-weighting of the respective modalities despite the initialization. Although this effect depends on the data at hand and did not cause problems in the experiments presented in Sec. 4.4 and Sec. 5.5, the explicit weighting of each modality is recommended in bidirectional association scenarios because then the balanced contribution of each modality is particularly important for the performance.

I complement the reservoir regularization approach with an additional concept to balance the contribution of inputs and outputs in this section. The balancing of contribution is integrated in the regularization process such that both can be accomplished in one step. The scheme can be applied in case of attractor- as well as transient-based computation.

### 5.6.1 State Prediction with activity distribution

I apply the idea of controlling the contribution of each modality to the reservoir activity $\mathbf{a}(k)$ in the state prediction framework. Consider inputs $\mathbf{x}(k)$, outputs $\mathbf{y}(k)$, and the reservoir states $\mathbf{h}(k)$ for $k = 1, \ldots, K$. Note that these variables can also correspond to attractor states of the system for certain inputs $\mathbf{x}(k)$ and $\mathbf{y}(k)$. According to the synchronous network update (3.2), the activity in the next time step is computed by

$$\begin{aligned}\mathbf{a}(k+1) &= \mathbf{W}_{ini}^{inp}\mathbf{x}(k) + \mathbf{W}_{ini}^{res}\mathbf{h}(k) + \mathbf{W}_{ini}^{fdb}\mathbf{y}(k) \\ &= I_{ini}^{inp}(k) + I_{ini}^{res}(k) + I_{ini}^{fdb}(k).\end{aligned}$$

The target of state prediction learning is the activity $\mathbf{a}(k+1)$ which is predicted from the combined system state $\mathbf{z}(k) = (\mathbf{x}(k)^T, \mathbf{h}(k)^T, \mathbf{y}(k)^T)^T$. The regression (5.5) computes new weights $\mathbf{W}_{reg}^{inp}$, $\mathbf{W}_{reg}^{res}$ and $\mathbf{W}_{reg}^{fdb}$ which generally yield different contributions from each modality ($I_{ini}^{inp}(k) \neq I_{reg}^{inp}(k)$, $I_{ini}^{res}(k) \neq I_{reg}^{res}(k)$ and $I_{ini}^{fdb}(k) \neq I_{reg}^{fdb}(k)$) even though the target $\mathbf{a}(k+1)$ is approximated accurately.

The approach for balancing contributions is to split up the regression into separate computations for each modality. Consider weights $g^{inp}$, $g^{res}$ and $g^{fdb}$ such that $g^{inp} + g^{res} + g^{fdb} = 1$. Then, the target activity $\mathbf{a}(k+1)$ can be split into a weighted sum

$$\mathbf{a}(k+1) = g^{inp}\mathbf{a}(k+1) + g^{res}\mathbf{a}(k+1) + g^{fdb}\mathbf{a}(k+1)$$

which yields target contributions $g^{\star}\mathbf{a}(k+1)$ from each modality $\star$ to the entire activity $\mathbf{a}(k+1)$. The regularized reservoir weights per modality are then given by

$$(\mathbf{W}_{reg}^{inp})^T = g^{inp}\left(\mathbf{X}^T\mathbf{X} + \beta^{inp}\mathbb{1}\right)^{-1}\mathbf{X}^T\mathbf{A} \tag{5.9}$$

$$(\mathbf{W}_{reg}^{res})^T = g^{res}\left(\mathbf{H}^T\mathbf{H} + \beta^{res}\mathbb{1}\right)^{-1}\mathbf{H}^T\mathbf{A} \tag{5.10}$$

$$(\mathbf{W}_{reg}^{fdb})^T = g^{fdb}\left(\mathbf{Y}^T\mathbf{Y} + \beta^{fdb}\mathbb{1}\right)^{-1}\mathbf{Y}^T\mathbf{A}. \tag{5.11}$$

Balancing of inflows by $g^{\star}$ and regularization of weights by $\beta^{\star}$ can now be controlled separately per modality.

The weighted contribution of the input modalities can be understood as modeling the dendritic structure of neurons. Fig. 5.7 illustrates the introduction of a dendritic tree structure to the common model of a neuron. Each reservoir neuron receives influx from the input neurons $\mathbf{x}$, reservoir neurons $\mathbf{h}$ and the output neurons $\mathbf{y}$. In the standard formal neuron model (see Fig. 5.7 (top left)), the weighted sum $a = \sum_i w_i z_i$ is computed which does not distinguish between the origin of each input $z_i$. The equal weighting of each input is made explicit in Fig. 5.7 (top right), where $a = g\sum_i w_i z_i = \sum_i (g w_i) z_i$. This scaling of the entire weights to that particular neuron by $g$ is equivalent to the slope $s$ of the neuron's activation function (see (3.4) and (3.5)). The intrinsic plasticity mechanism presented in Sec. 3.2.5 scales the weights in a coupled manner which corresponds to scaling $g$. The idea of balancing contributions is to break this coupled scaling to achieve distinct scalings of weights per modality (see Fig. 5.7 (bottom)). Then, the activity of a single reservoir neuron is calculated according to

$$a = g^{inp}\sum_{i=1}^{D} w_i^{inp} x_i + g^{res}\sum_{i=1}^{R} w_i^{res} h_i + g^{fdb}\sum_{i=1}^{O} w_i^{fdb} y_i.$$

The inflows from each modality can be scaled such that each modality contributes in a controlled fashion to the neuron's activity, for instance equally. This balancing of contributions is integrated into the reservoir regularization methodology by (5.9)–(5.11) and is particularly helpful for bidirectional association. Note that (5.9)–(5.11) directly include the scaling into the computed weights such that there is no additional computation of weightings $g^{\star}$ necessary during network exploitation.

### 5.6.2 Balancing contributions for bidirectional association

In this section, the principle functioning of the weighting mechanism to balance contributions of modalities to the reservoir activity is shown for the same kinematic learning scenario as in Sec. 5.5. This time, however, the forward and inverse kinematics are learned in a single network and both mappings are queried for evaluation of the balancing approach.

#### Setup

Fig. 5.8 (top left) shows the kinematic setup for the robot arm with two degrees of freedom. End effector positions $\mathbf{x}$ are mapped to joint angles $\mathbf{y}$ using the righty elbow solution of the inverse kinematics. The forward mapping from inputs $\mathbf{x}$ to outputs $\mathbf{y}$ is the inverse kinematic mapping (following the typical setup of networks), and the backward mapping from outputs $\mathbf{y}$ to inputs $\mathbf{x}$ is the forward kinematic mapping. Both mappings will be evaluated in the following

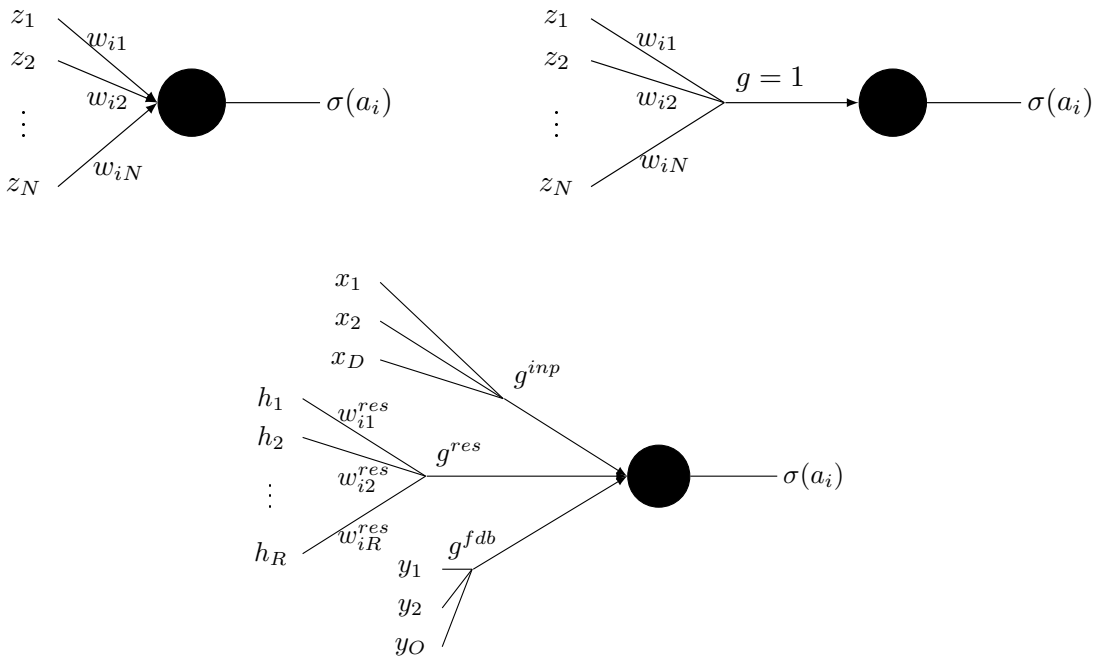



Fig. 5.7: Formal neuron model with a dendritic tree of depth 1 (top left). The neuron's activity is calculated by the weighted sum $a = \sum_i w_i z_i$. Explicit illustration of tree structure with unique branch and corresponding weighting factor $g = 1$ (top right). Generalized dendritic tree model of neuron with depth 2 (bottom): For each modality, the weighted sum per modality is weighted by the corresponding gating factor $g^\star$.

experiment depending on the weighting factors $g^\star$ for the contributions of inputs and outputs to the reservoir activity. That means the network is either driven by inputs $\mathbf{x}$ and network estimates are read-out at the outputs $\mathbf{y}$ or vice versa. Note that training data is provided comprising a righty elbow solution only, which renders the mapping from inputs to output bijective and invertible.

The training trajectory (gray line in Fig. 5.8 (top left)) is used for reservoir regularization and read-out learning, where the validation trajectory (light gray line in Fig. 5.8 (top left)) is used to tune the regression parameters. I use a grid search over $\alpha = \alpha^{out} = \alpha^{rec}$ and $\beta = \beta^{inp} = \beta^{res} = \beta^{fdb}$ analogously to Algorithm 5.4, where for each network initialization the configuration of $\alpha$ and $\beta$ that minimizes the validation error is determined in a brute-force manner. In addition to the error of the forward mapping in Algorithm 5.4, the error of the backward mapping is also considered in order to select the optimal configuration of $\alpha$ and $\beta$. Such brute-force tuning of $\beta$ is typically not necessary, but in this example the model changes dramatically with the weights $g^\star$ of the modalities which renders an explicit search of the regularization constants necessary.

The test error is evaluated on a grid of target points (light gray dots in Fig. 5.8 (top left)) using attractor-based computation. That is, the network is driven by inputs (outputs) until convergence of the network dynamics before the estimated outputs (inputs) are used for error calculation. To account for random network initialization, the results are averaged over ten independent trials. Initialization proceeds as described in the kinematic setting in Sec. 5.5.

The performance for the forward and inverse kinematics is evaluated depending on the factor of input and output weighting. For $g \in \{0, 0.2, \ldots, 1\}$, the input and output contributions are weighted according to $g^{inp} = 0.6(1 - g)$ and $g^{fdb} = 0.6g$, while the contribution of the reservoir $g^{res} = 0.4$ is held constant. I expect that for extreme values of $g$, e.g. $g = 0$ and $g = 1$, the

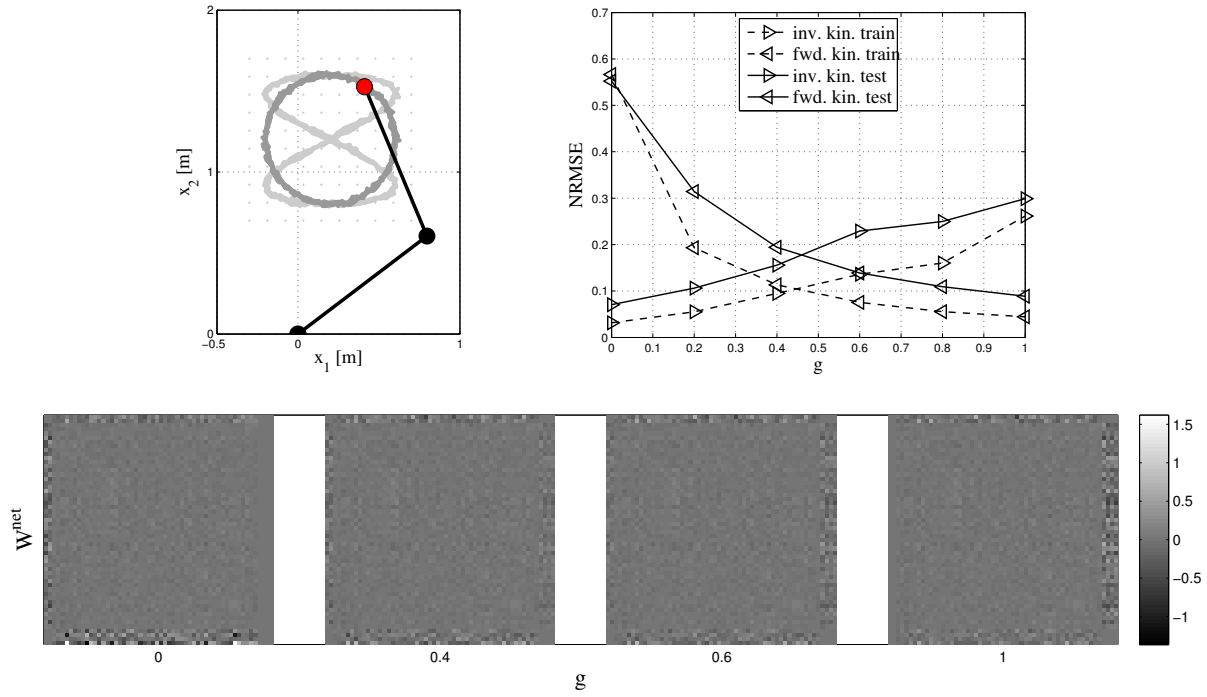



Fig. 5.8: Robot arm with two degrees of freedom (top left) with training, validation and test patterns shown in gray and light gray, respectively. Normalized root mean square errors (NRMSE) for the inverse and forward kinematic mappings depending on the activity distribution factor $g$ (top right). Weight matrices $\mathbf{W}^{net}$ of a reservoir network after reservoir regularization with different activity distribution factors $g$. The model is tuned to an input-driven ($g = 0$), output-driven ($g = 1$), or associative ($g \approx 0.5$) mode which is reflected by the corresponding errors and weight matrices. The sub-matrices of $\mathbf{W}^{net}$ display increased values depending on the model directionality (see (3.1) for the construction of $\mathbf{W}^{net}$).

network will either resemble a forward or a backward model between the input and joint space. In between, values of $g \approx 0.5$ result in a mixed forward and backward model, i.e. an associative model that bidirectionally connects input and joint space.

I use the normalized root mean square error (NRMSE)

$$NRMSE = \sqrt{\frac{1}{K}\sum_{k=1}^{K}\frac{1}{D}\sum_{i=1}^{D}\frac{1}{\sigma_i^2}\left(y_i(k) - \hat{y}_i(k)\right)^2}, \tag{5.12}$$

for evaluation, where $y_i(k)$ is the $i$-th component of target $\mathbf{y}(k) \in \mathbb{R}^D$ and $\hat{y}_i(k)$ the corresponding estimation of the model. $\sigma_i^2$ is the variance of the target signal $\mathbf{y}(k)$ in the $i$-th component. If the variance is zero, then $\sigma_i^2$ is set to unity, which, however, does not apply in the following experiments.

Note that $\mathbf{y}$ plays the role of the currently queried model output modality. For instance, when the backward model is queried, the outputs are driven by the external signal and the estimated inputs $\mathbf{x}$ are evaluated according to (5.12). In the context of bidirectional association, the $NRMSE$ has the advantage to be comparable between input and output modalities which typically differ in dimensionality and distribution of values.

### Results

Fig. 5.8 (top right) shows the NRMSE for the forward and inverse kinematic mappings depending on the gain $g$. The errors for the forward and backward model depend on the weighting

factor $g$ as expected: For $g = 0$, only the inverse kinematics are approximated while the forward kinematics are effectively turned off ($g^{fdb} = 0$). For $g = 1$, the model operates well in the backward direction, i.e. approximating the forward kinematics, whereas the model receives no contributions from the input neurons ($g^{inp} = 0$). The value of $g$ for which both error curves intersect is the optimal regime for associative computation: The combined errors for both forward and backward model take their minimal values approximately at $g = 0.4$.

The results can be related to the work in [23] which showed that learning the backward mapping in addition to the forward mapping does not affect the performance *if both network variants have output feedback*. In the experiments presented here, models range from purely unidirectional to bidirectional which can be parameterized continuously by $g$. Although the unidirectional forward or backward models yield the best performances on their respective tasks, the associative models are still competitive. Moreover, the associative models have essentially more computational power than the purely unidirectional models: Output feedback dynamics in associative models can represent ambiguities of the inverse mapping, whereas unidirectional models without output feedback dynamics can not. The representation of ambiguities in associative reservoir computing networks is demonstrated later on in Chapter 7.

## 5.7 Concluding remarks

Reservoir regularization enables robust offline training of ESNs with output feedback. By minimizing the norm of the weights that excite the reservoir, reservoir regularization mitigates the sensible balance of task-specific performance and read-out regularization. This in turn allows to increase the weight norm of the read-out weights and consequently improves the task-specific performance and stability. Although reservoir regularization introduces extra computational costs and an extra parameter, the gained robustness of learning is a necessary prerequisite for the convenient application of reservoir networks with output feedback. The presented results point out the close relation of stability and regularization which is of major importance for many application areas where adaptive recurrent systems generate patterns autonomously or implement feedback mechanisms.

In a second step, I extended the reservoir regularization approach to balance contributions of input, reservoir and output neurons to the hidden representation. Balancing of contributions to the reservoir activity assures that the regularized model is still responsive to inputs or outputs, i.e. can be driven by the respective modality. In particular, balancing the distribution of activities in a controlled manner is a powerful tool for bidirectional association. Setting the weighting factors $g^{inp}$, $g^{res}$ and $g^{fdb}$ enables the continuous adjustment of the model to a desired functionality. The activity distribution approach thereby elucidates the trade-off between pure forward or backward models and bidirectional association.

In conclusion, the problem of output feedback stability stated in Sec. 3.4.2 is resolved to a considerable extent by the reservoir and read-out regularization techniques described in this chapter. In particular, offline learning of associative reservoir networks can be accomplished robustly, which is demonstrated in a series of examples in the following chapter.

Chapter 6

# Robust offline learning of associative reservoir networks

## 6.1 Combined forward and inverse models

The previous chapter deeply discussed the stabilization of reservoir networks with output feedback by regularization techniques. The networks were mostly trained as unidirectional function approximators. In this chapter, we focus on robust offline training of bidirectional mappings with invertible forward models. That is, the forward and inverse mapping are one-to-one. The associative capabilities of the networks with output feedback connections are extensively evaluated and demonstrated on a set of tasks.

First, a simple example demonstrates the principle functioning of bidirectional association by associative reservoir computing for an invertible, one-dimensional function. Then, the combined forward and inverse kinematic models of two robotic manipulators are learned and evaluated. The first one is a simple planar arm with two degrees of freedom. The other one is the industrial Puma robot arm with six degrees of freedom. Also the release of controlled variables and mixed constraints in input and output space are evaluated. In both cases, the inverse model is trained on a single solution of the inverse kinematics. Learning multiple solutions of the inverse model is subject to the next chapter. Finally, the combination of dimensionality reduction and the backward reconstruction of high-dimensional data samples is demonstrated on images of handwritten digits. In all examples, static relations between inputs and outputs are trained in Associative Extreme Learning Machines (AELM, Sec. 3.3.4) and attractor-based computation is applied.

### 6.1.1 A tiny example

Learning the bidirectional mapping of a non-linear input-output relation is demonstrated on a simple example in this section. Additionally, the role of noisy observations with respect to invertibility is discussed and evaluated.

Consider the non-linear function shown in Fig. 6.1 (a). Because the function $y(x)$ is bijective, it has an inverse function $x(y)$ (compare Fig. 6.1 (a) and (d)). Training of an AELM with 50 hidden neurons using reservoir regularization and a line search for read-out regularization parameters $\alpha^{out}$ and $\alpha^{rec}$ in (3.14) and (3.16) results in robust association of the noisy data (see Fig. 6.1 (c) and (f)). More details about the network initialization can be found in Appendix A.3.1. Note that the noisy data in Fig. 6.1 (b) and (e) does not corrupt learning of a smooth bidirectional mapping by the AELM.

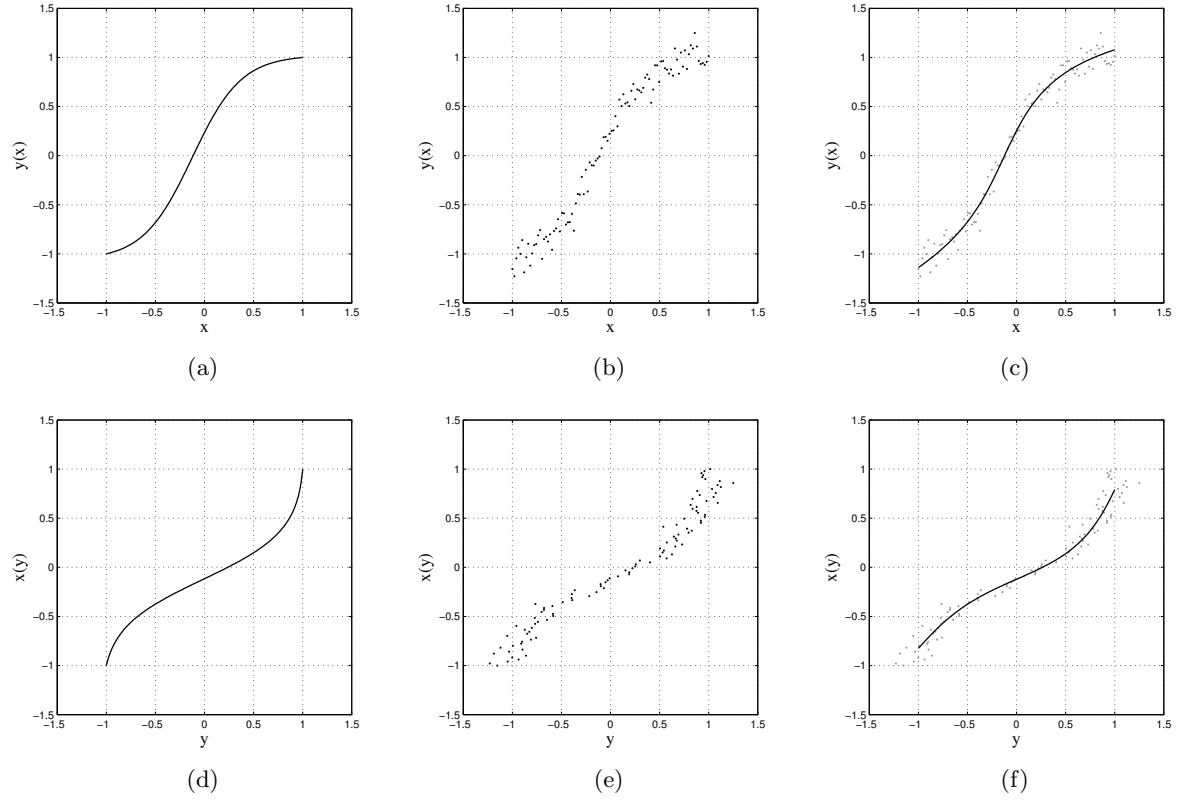



Fig. 6.1: Invertible relationship between two variables $x$ and $y$: Variable $y(x)$ as a function of $x$ (a) and variable $x(y)$ as a function of $y$ (d). Noisy observations of relationship $y(x) + \nu$ (b) and their inversion $x(y + \nu)$ (e). Network approximation of the forward and inverse function from noisy data (c) and (f).

### 6.1.2 Noise and invertibility

However, noisy observations of the forward function violate its uniqueness: Typically, multiple output samples are observed for the same input, although the function underlying the data generation process may be one-to-one. Thus, the mapping implied by the data is – strictly speaking – not invertible (compare Fig. 6.1 (b) and (e)). In fact, this is true for both, the forward and the inverse mapping. Moreover, noise is amplified by the projection from one space to the other unless the mapping is $y = f(x) = x$. Since noisy data of an invertible relation is principally ambiguous, it is important to discuss the influence of noise in the context of learning bidirectional associations.

Generally, noisy data sampled from the forward relation yield a convex solution set. Therefore, the average of the output samples for the same input is a valid solution. At the same time, several outputs for the same input render the inverse mapping non-injective: Several outputs map to the same input value, i.e. the forward model is ambiguous (see Sec. 2.2).

For a more systematic evaluation of the robustness of associative learning to noise, the following experiment is conducted: Consider the linear relationship $y(x) = x + \nu$ with additive Gaussian-distributed noise $\nu$ and $x \in [0, 1]$. The variance of the Gaussian noise is increased from 0 to 0.09 which creates heavily noisy data. 50 independent networks are trained per noise level and the data is randomly split up into training and test set for each trial in a cross-validation manner. The networks are initialized and trained with the same parameters as above (see Appendix A.3.1).

Fig. 6.2 shows the mean square error for the forward and backward mappings on the training and test set, respectively. The straight line in Fig. 6.2 shows the noise level. Errors below this

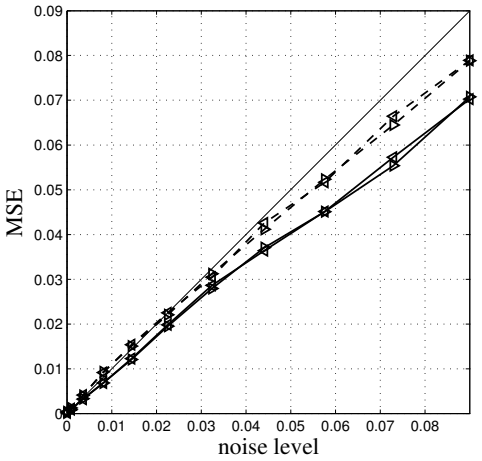



Fig. 6.2: Fitting of bidirectional mappings to a linear relation with noise. MSE on training (solid) and test set (dashed line) for the forward ($\triangleright$) and backward mapping ($\triangleleft$), respectively.

line indicate over-fitting of the model to the data. Fig. 6.2 shows that the bidirectional mapping behaves as expected with respect to the increased noise level, i.e. the overall error increases smoothly with the noise level. The strictly non-invertible training samples do not corrupt learning in a catastrophic manner.

Note that I used the identity as forward and backward mapping in this example to not amplify noise in one direction. For this reason, a symmetric relation and similar errors for both, the forward and the backward mapping, are obtained (compare Fig. 6.2). Moreover, Fig. 6.2 shows that over-fitting is rather moderate. Tuning of the regularization parameters with respect to output feedback stability on the training set by Algorithm 5.4 typically yields to a good generalization performance on the test set.

## 6.2 Learning kinematics of a planar robot arm

We now come back to the kinematics example of a planar robot arm with two degrees of freedom encountered previously. In this section, the robustness of learning forward and backward models of invertible relations is evaluated for a range of parameters. In particular, I analyze the effects of the training set size and the hidden layer size on the learning systematically. Moreover, associative completion is demonstrated and evaluated in this section.

### 6.2.1 Experiment setup

The inverse kinematics of the planar arm (see Fig. 6.3) has two discrete solutions when controlling the end effector position only: The righty or lefty elbow solutions. In this section, the redundancy of the inverse kinematics is resolved by explicitly providing only righty elbowed solutions for training (see Appendix A.4 for details). Data is collected on a grid $G$ of task space coordinates with $15 \times 15$ vertices (see Fig. 6.3).

I evaluate properties of the AELM when learning associative mappings. I vary the network size and amount of training data systematically. All results are averaged over 100 independent trials per network size and training set size. In each trial, the network is initialized with random weights, and training and test set are randomly chosen from the grid $G$. The size of the training and test set is parameterized by the percentage $p$, i.e. $100p$ percent are selected as training

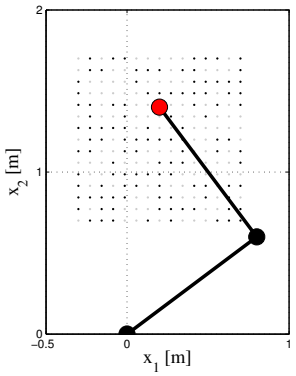



Fig. 6.3: Robot arm with two degrees of freedom. Training and test patterns are shown in black and gray, respectively.

and $100(1-p)$ percent as test set. Fig. 6.3 shows such a distribution of the samples on the grid $G$ in training and test set for $p = 0.5$.

Each network is initialized with weights $\mathbf{W}^{inp}$ and $\mathbf{W}^{fdb}$ drawn from an uniform distribution in ranges $a^{inp} = 1/D$ and $a^{fdb} = 1/O$, respectively. The biases $\mathbf{b}$ and slopes $\mathbf{s}$ of the activation functions (3.5) are initialized also uniformly in $[-1, 1]$ and $[0.1, 1.1]$. These parameters have proven to work fine if the input and output data resides in a reasonable range, e.g. $[-1, 1]$. The latter is achieved for the joint angle values by using the coordinate representation of each joint angle $q_i$, i.e. $q_i \rightarrow (\mathbf{y}_{i_1}, \mathbf{y}_{i_2}) = (sin(q_i), cos(q_i))$, as described in Sec. 5.5.1. In the following, $q_i$ denotes the respective output components in the network. This normalization strategy in combination with the activity distribution method presented in Sec. 5.6.2 makes further tuning of the scaling of input and output weights in most cases dispensable. Here, only the number of input and output dimensions is considered to achieve a rather equal-ranged inflow from input and output neurons into the hidden layer. Due to the static input-output training data, the internal dynamics of a reservoir for transient encoding is not required, i.e. $\mathbf{W}^{res} = \mathbf{0}$, and therefore I use the lightweight AELM in the following.

Training is done offline applying the regularization technique with activity distribution (Sec. 5.6.2) first. The contributions of inputs and outputs are set to be equal, i.e. $g^{inp} = g^{fdb} = 0.5$ in (5.9) and (5.11), and the respective regularization parameters are $\beta^{inp} = \beta^{out} = 0.1$. Then, the read-out weights $\mathbf{W}^{out}$ and $\mathbf{W}^{rec}$ are trained using the grid-search for the regularization parameter $\alpha^{out}$ and $\alpha^{rec}$ according to Algorithm 5.4 to assure output feedback stability.

The trained models are evaluated with respect to the forward and inverse kinematics. Errors of the forward kinematics are calculated by

$$E^{FK} = \frac{1}{N} \sum_{n=1}^{N} ||\mathbf{x}_n - \hat{FK}(\mathbf{y}_n)|| \tag{6.1}$$

and of the inverse kinematics by

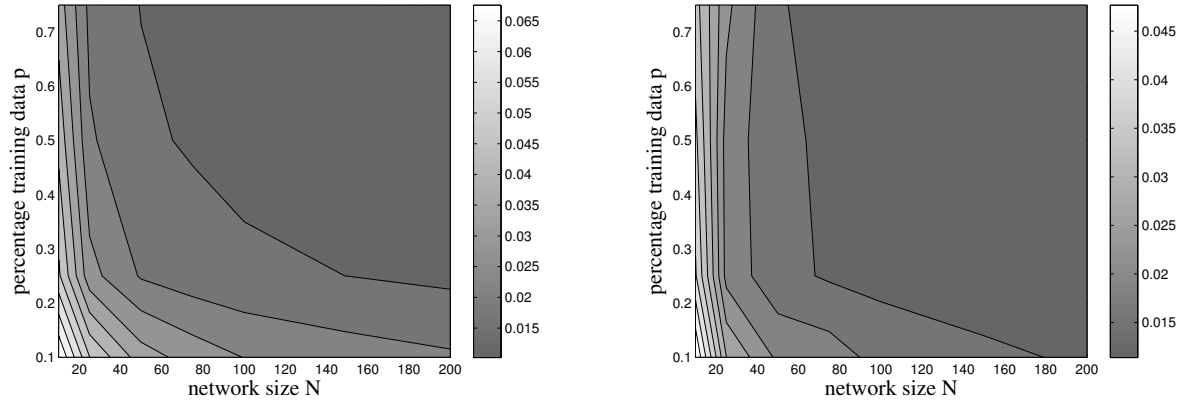
$$E^{IK} = \frac{1}{N} \sum_{n=1}^{N} ||\mathbf{x}_n - FK(\hat{IK}(\mathbf{x}_n))|| \tag{6.2}$$

for samples $(\mathbf{x}_n, \mathbf{y}_n)$ with $n = 1, \ldots, N$, where $FK$ and $\hat{FK}$ denote the actual and estimated forward kinematics, respectively. The same notation applies to the inverse kinematics $IK$ and $\hat{IK}$.

### 6.2.2 Robust association

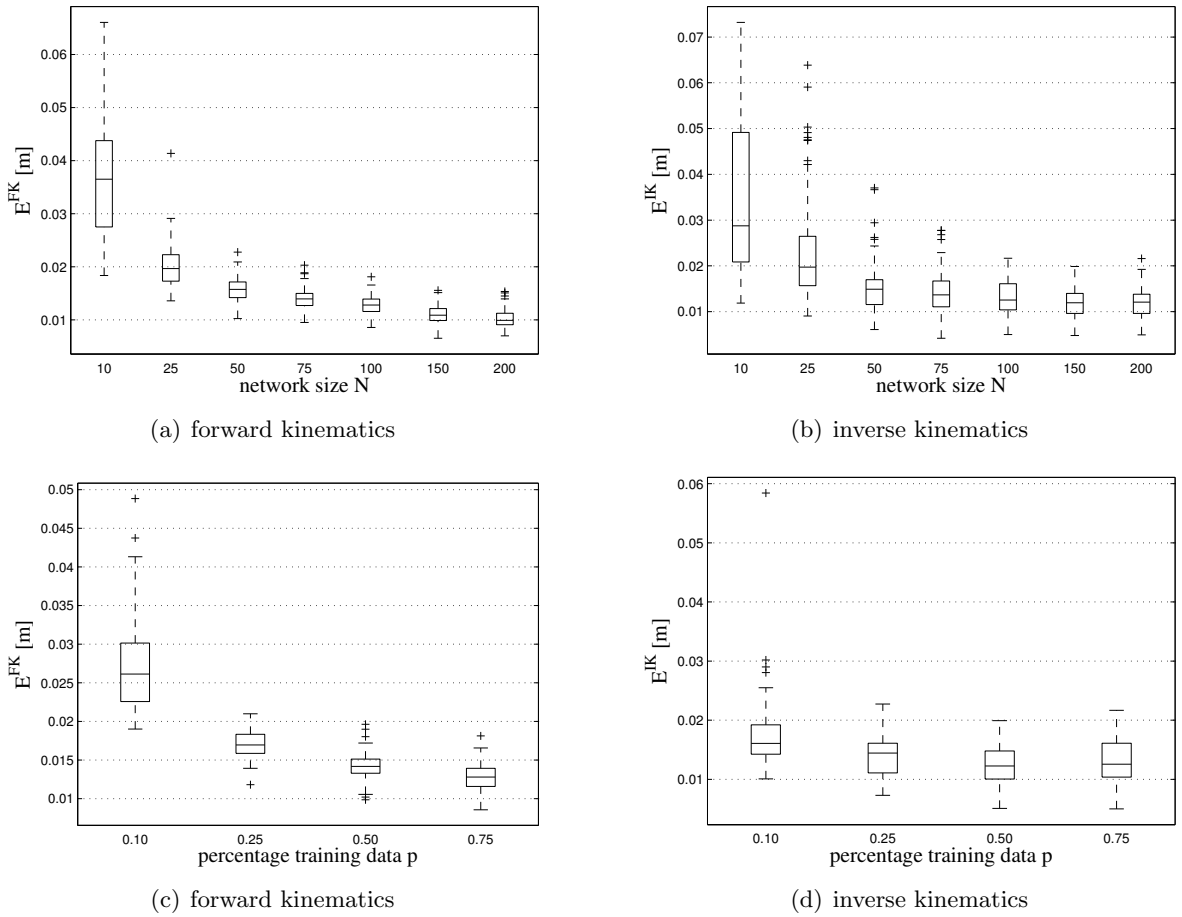
Fig. 6.4 shows the average test errors of the networks on the forward (a) and inverse kinematics (b) as function of the network and training data size. Already a few training samples and a medium sized network are sufficient to train accurate forward and inverse models.

To investigate the influence of both, network and training set size, more deeply, Fig. 6.5 displays the statistics of the trained models in boxplot diagrams. Increasing the network size improves the model accuracy as expected. Note further that there are compact error distributions in Fig. 6.5 which indicates a robust training of networks irrespective of the rather large range of network and training set sizes. Regularization of the network weights and the grid-search for read-out weight regularization make robust learning of associative models possible: While the regularization of weights $\mathbf{W}^{inp}$ and $\mathbf{W}^{fdb}$ mitigates a strong dependency on the read-out regularization, the line search for the read-out regularization parameters fine-tunes the weight norm to the range of output feedback stability. This robust learning behavior of the highly dynamic output feedback-driven systems is a key contribution of this thesis.




(a) forward kinematics (b) inverse kinematics

Fig. 6.4: Average test errors of the forward and inverse kinematic models depending on network and training set size. The errors (coded in gray scale) are presented in meters.




(a) forward kinematics (b) inverse kinematics

(c) forward kinematics (d) inverse kinematics

Fig. 6.5: Statistics of test errors for the forward (left column) and inverse (right column) kinematic models. For $p = 0.75$, the top row shows the influence of the network size on the performance of both models. For $R = 100$, the bottom row shows the influence of the amount of data used for training on the performance of both models.

### 6.2.3 Unconstrained control variables

In a next step, the flexibility of the associative approach is demonstrated. In the following experiments, control variables are released, i.e. instead to query joint angles for a specific position $\mathbf{x}$, the solution is only constrained to fulfill $x_1 = 0.5$ for instance. The remaining control variables, $x_2$ in case of the planar arm, remain *unconstrained*. Then, also the released input runs in a recurrent loop in addition to the the output feedback loop, i.e. $x_2$ is associatively completed (see Sec. 2.4 and (3.11)). Note that the networks are not trained and the regularization parameters are not tuned for the release of inputs.

Note further that the inverse kinematics are then no longer unique: A manifold of joint angles fulfill the required constraint. The output feedback dynamics become continuous attractors: The unconstrained input was trained for a quasi-continuous set of its values from which one is now dynamically selected. Continuous attractor dynamics can be understood as elongated valleys with the same energy of the feedback dynamics. In practice, continuous attractors are never perfectly level in the valley which results in a slow drift along the attractor manifold [27, 57, 150]. It is therefore meaningful to stop iterating the dynamics as soon as the change of the state becomes small, i.e. the dynamics reach the attractor valley. This can already be accomplished using Algorithm 3.1 with an increased threshold $\delta$. In this section, however, I use an adopted convergence criterion based on the acceleration of the network state (see Algorithm 6.5). This convergence criterion yields to more accurate identification of slow drifts of the network state along continuous attractor valleys and results in early stopping of the network iteration.

The network performance is evaluated in case of solely controlling either $x_1$ or $x_2$. Errors are calculated with respect to the controlled variable: For controlling $x_1$ only, the error is computed according to

$$E(x_1) = ||x_1 - FK(\hat{IK}(x_1))_1||, \tag{6.3}$$

where $FK$ is the actual forward kinematic mapping and $\hat{IK}$ is the learned inverse kinematic mapping. Fig. 6.6 shows the error of the inverse estimate with respect to the controlled variable in meters. For both task dimensions $x_1$ and $x_2$, the constraints are fulfilled accurately. Although there is a slow drift along the manifold of solutions, the network stays in the range of learned configurations using Algorithm 6.5. This can also be explained by the fact that there is a – though output feedback-driven – rather stable estimate of the unconstrained variable at the input which is sufficient to slow down the drift.

Exemplary solutions are displayed for controlling $x_1$ or $x_2$ in Fig. 6.7 (a) and (b), respectively. The constraints are visualized as gray lines in Fig. 6.7, where in the case of controlling only one task space variable the line represents the set of solutions fulfilling the constraint. Although the associative model was not trained to operate in these novel configurations of driven inputs, the results show that the model still accurately resolves the ambiguous inverse mapping. I.e. the model generalizes to associative completion settings with infinitely many solutions.

**Algorithm 6.5** Associative completion with convergence criterion for continuous attractors

**Require:** get combination of external inputs components $x_i$ and $y_i$
**Require:** set $t=0$, $\Delta^2 h=\infty$, $\delta=10^{-6}$ and $t_{max}=1000$
1: **while** $\Delta^2 h > \delta$ and $k < k_{max}$ **do**
2: inject external inputs into network
3: execute network iteration (3.11) and (3.3)–(3.7)
4: compute state change $\Delta h(k) = \|\mathbf{h}(k) - \mathbf{h}(k-1)\|^2$
5: compute state acceleration $\Delta^2 s = \|\Delta h(k) - \Delta h(k-1)\|^2$
6: $k=k+1$
7: **end while**

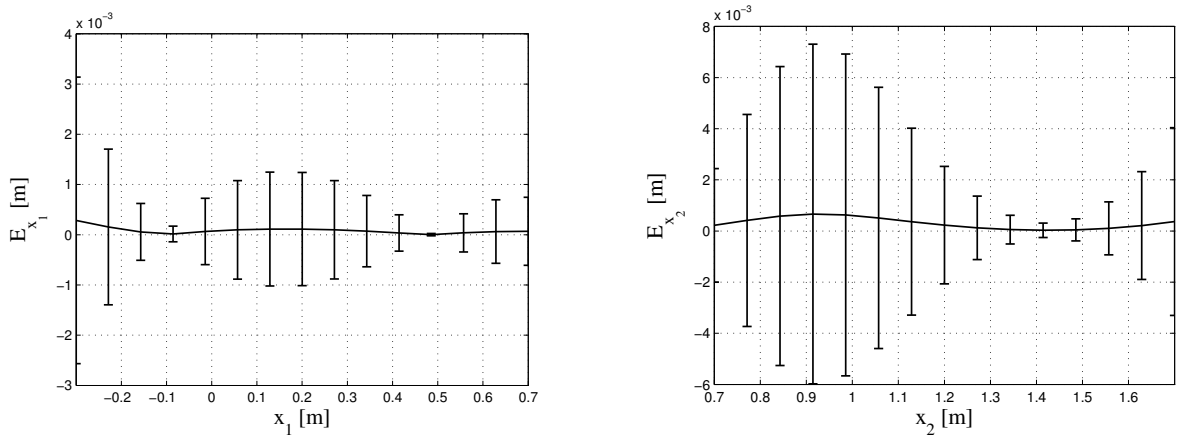



Fig. 6.6: Inverse model performance when control variables are unconstrained. Error of the constrained control variable $x_1$ (left) and $x_2$ (right) with standard deviations over all 100 networks with $R=100$ and trained using $p = 0.5$.

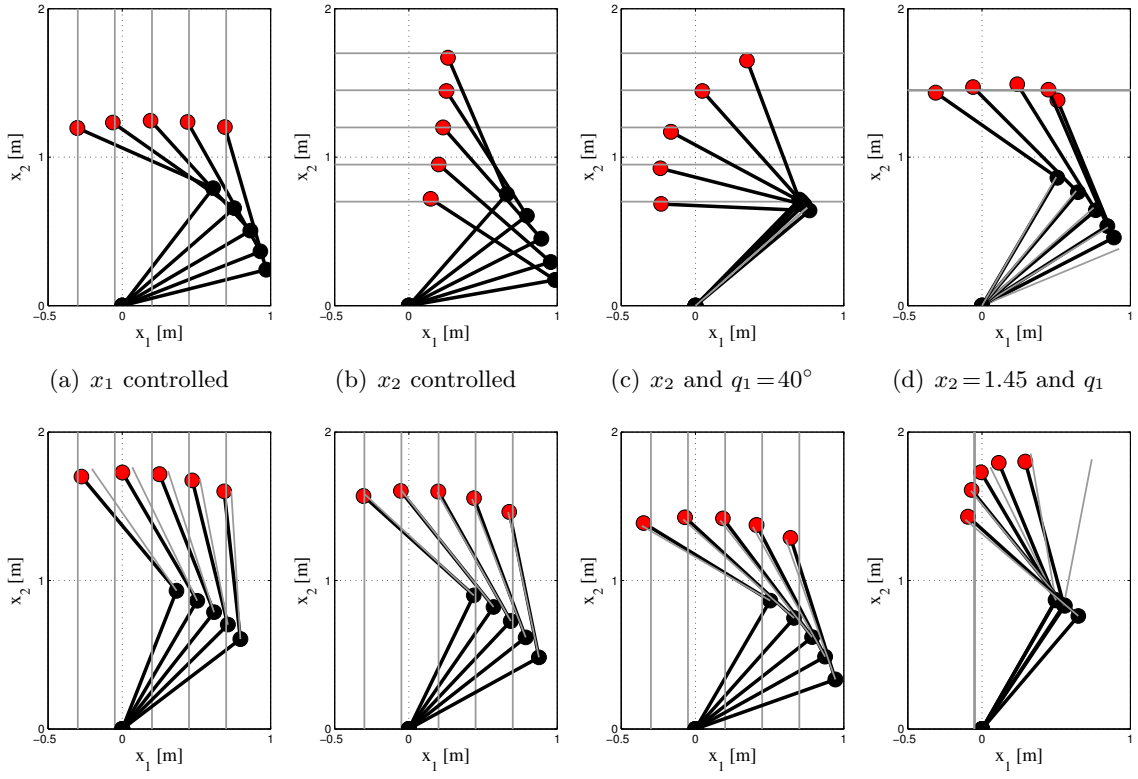
### 6.2.4 Mixed constraints: Flexible control by associative completion

In a second step, the application of mixed constraints in task and joint space is demonstrated. This means that for example $x_1$ and $q_2$ should fulfill certain constraints at the same time. Note that the combination of task and joint constraints does typically not lead to a manifold of solutions, but rather is often ill-posed in the sense that no joint configuration exists that fulfills both constraints at the same time. The network is expected to find a rather exact solution when the constraints are not ill-posed, or an approximate solution that tries to fulfill both constraints in the ill-posed case.

Fig. 6.7 (c) - (h) show joint configurations estimated by the network for various combinations of constraints. In Fig. 6.7 (c), the constraint for the first joint $q_1$ is held fixed while the network is queried to fulfill a set of desired values for $x_2$. In Fig. 6.7 (d), the opposite is the case: For various joint angles $q_1$, the network is demanded to keep the height $x_2$ constant. In both cases, the network generates accurate solutions in the sense that both constraints are satisfied. In particular, the sequence Fig. 6.7 (e) - (g) shows this effect strikingly: For the same set of task space constraints $x_1$, three different values for the second joint angle $q_2$ are given to the network. While in Fig. 6.7 (e) and (g) the constraint in joint space results in a slightly ill-posed problem, the joint space constraint $q_2=70°$ allows to fulfill task and joint constraints (see Fig. 6.7 (f)).

In case of ill-posed constraints, it remains an open question whether one constraint can dominate another one or if both constraints influence the solution equally. The presented network configuration, however, seems to prefer task space constraints as one can see in Fig. 6.7 (d) and (h). This might be due to a slight imbalance of contributions from the input and outputs to the network activity. Although the activity distribution method can balance the contributions of inputs and outputs to a great extent, differences in the input statistics of the respective modalities can cause a slight imbalance.

For robotic applications, it is of interest to weight the different constraints during system operation. This requires additional mechanisms to parameterize the network dynamics. To this end, the learning of forward and inverse kinematics in an associative model enables flexible combination of constraints in task and joint space.




(a) $x_1$ controlled (b) $x_2$ controlled (c) $x_2$ and $q_1 = 40°$ (d) $x_2 = 1.45$ and $q_1$

(e) $x_1$ and $q_2 = 56°$ (f) $x_1$ and $q_2 = 70°$ (g) $x_1$ and $q_2 = 90°$ (h) $x_1 = -0.05$ and $q_2$

Fig. 6.7: Reduced constraints for the inverse kinematic mapping by releasing driving inputs (a-b): Controlling left-right position of the end effector only by driving $x_1$ (a). Controlling the height over ground by driving only $x_2$ (b). The gray lines display the (here one-dimensional) manifold of end effector positions that satisfy the respective constraint of the input variable. Mixed constraints in task and joint space (c-h): The network is externally driven with one component from each input and output modality. One constraint is held fixed while the other is varied. (c) The height over ground $x_2$ is varied (see gray horizontal lines) and joint angle $q_1 = 40°$ is held constant. (d) The height over ground $x_2$ is held fixed and joint angle $q_1$ is varied. Read (e-h) analogously. The gray lines at the joint angles display the desired angle configuration. This angle is fed into the network's output neurons in addition to the task space constraint (horizontal or vertical lines).

## 6.3 Learning kinematics of the Puma robot arm

Finally, the associative capabilities of the AELM are demonstrated on an industrial robotic platform, namely the Puma 560 robot. In contrast to the planar robot arm which was so far well-suited to illustrated the principle functioning of the associative approach to learn combined forward and inverse models, the Puma robot arm has six degrees of freedom and scales the problem of inverse kinematics to application level.

### 6.3.1 The Puma robot arm

The end effector of the Puma robot (see Fig. 6.8[1]) is controlled in a three dimensional cube, while the orientation of the end effector is held constant. Hence, there is also variance in the last three wrist joints which is learned by the network. The capability of reservoir networks to learn the inverse kinematics of the Puma arm and also the entire upper body of the humanoid Honda Research Robot comprising the orientation of the end effector was previously demonstrated in [152] and [98, 153], respectively. While in the cited work only the inverse kinematics have been learned online in a backpropagation-decorrelation network with output feedback, here both forward and inverse kinematics are trained offline in an AELM.

Fig. 6.8: The Puma robot arm with six degrees of freedom.

### 6.3.2 Experiment setup

The training data is generated on a three-dimensional grid in task space which builds up a cube of $50\times50\times50cm$. The corresponding joint angles are produced with the analytical inverse model provided by the MATLAB robot toolbox [154]. Again, in this chapter we focus on a unique solution of the inverse kinematics which is in this case the default lefty upper elbow configuration without wrist flip (see `ikine560.m` of Corke's toolbox [154]).

All results are averaged over 100 independent trials. The networks have 200 hidden neurons and 50% of the data in the cube are randomly selected for training per trial. The remaining data samples serve as test set such that the model performance is evaluated in a cross-validation manner. The networks are initialized and trained as described in section Sec. 6.2.

### 6.3.3 Robust association

Again, the capability to learn forward and inverse kinematics in a single network is first evaluated. Fig. 6.9 displays the performance statistics of the AELMs for forward (left) and inverse (right) kinematics on the training and test set. Both mappings are approximated accurately, where the trained inverse models are slightly worse than the forward models. This offset reflects the different difficulties of both tasks. However, in both cases no over-fitting behavior can be observed, i.e. training and test errors are comparable, although no tuning of regularization parameters with respect to a validation set has been applied and only half of the data in the cube is used for training. This confirms the previous result that the selection of read-out regularization parameters $\alpha^{rec}$ and $\alpha^{out}$ on the training set but with respect to the output feedback stability criterion tested by Algorithm 5.4 is also beneficial for task-specific generalization. As pointed out in Chapter 5, output feedback stability and task-specific generalization are tightly coupled: Over-fitting networks to the data will affect generalization as well as output feedback

[1]Visualization of the robot was done with help of the software made available by Don Riley [151].

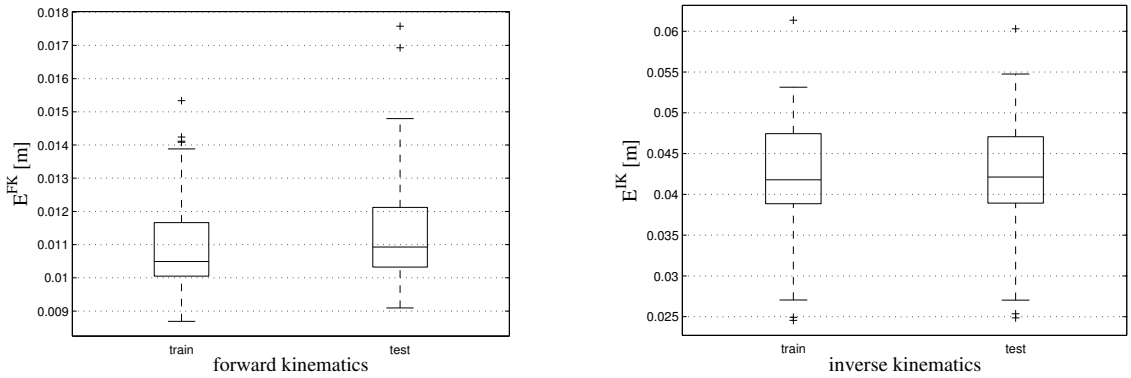



Fig. 6.9: Errors of the learned forward (left) and inverse (right) kinematic models on the training and test set, respectively. Errors are presented in task space.

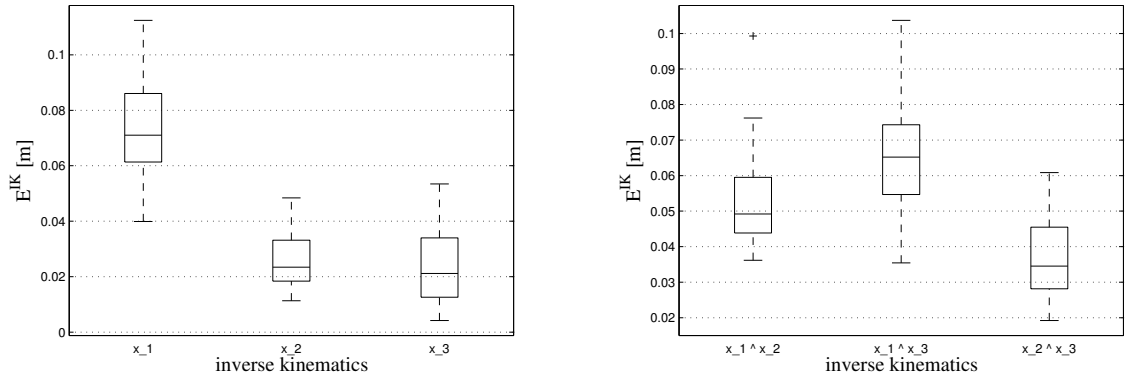



Fig. 6.10: Performance statistics for task space constraints for all 100 networks: Single constrained input (left) and combination of two constrained inputs (right).

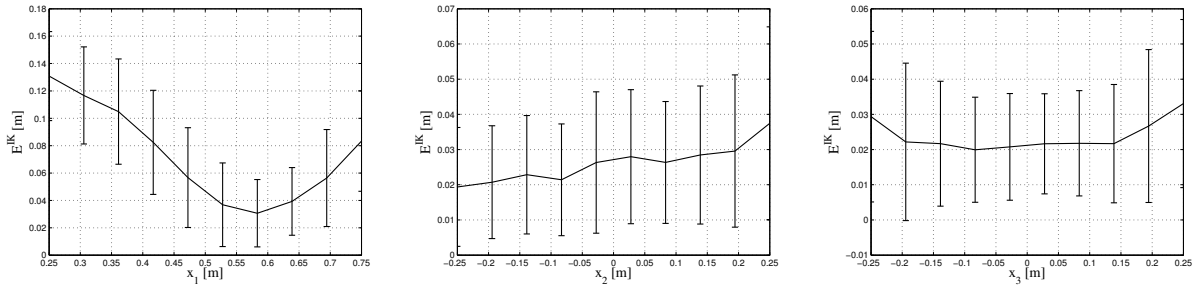



Fig. 6.11: Performance errors (6.3) when only one task dimension is controlled. Errors are averaged over all 100 network configurations and are displayed with standard deviations.

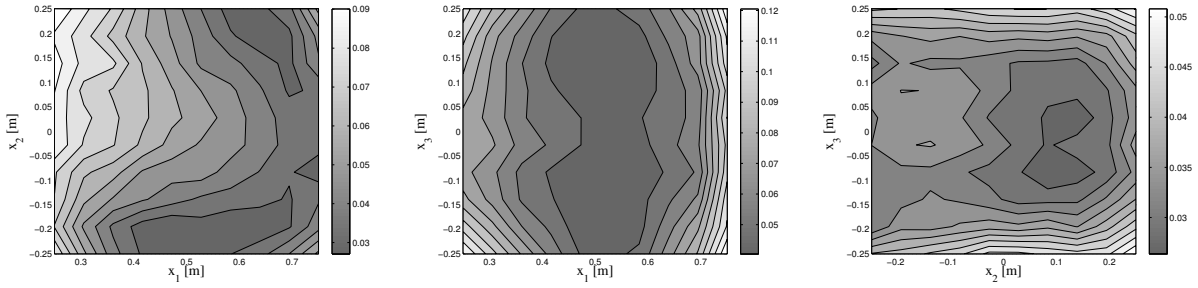



Fig. 6.12: Spatial resolution of performance errors (6.3) for combinations of two task constraints averaged over all 100 networks.

stability due to larger weight norms. Hence, regularization of the read-out weights with respect to the output feedback stability criterion by Algorithm 5.4 on the training set renders association robust and supports generalization.

### 6.3.4 Unconstrained control variables

Finally, the performance of the inverse estimate is evaluated for flexibly queried constraints. All combination of task space constraints are investigated, where for each configuration there is at least a one-dimensional manifold of solutions that fulfills the constraints. This multiplicity of solutions results in continuous attractor dynamics of the output feedback-driven network. Again, the modified convergence criterion in Algorithm 6.5 (see discussion in Sec. 6.2.3) is applied to stop the network iteration as soon as the manifold of solutions is reached and the network slowly drifts along this continuous attractor. I sample driving inputs regularly from a line or grid for a single constrained input or two constrained inputs, respectively. The network is then queried to associatively complete the other components, where the samples are presented in a random order to the network.

First, only one task input is driven externally such that two other task inputs remain unconstrained. In this case, the network has to choose a solution from a two-dimensional manifold. Fig. 6.11 shows the resulting performance error (6.3) on the respective constraint averaged over all 100 network initialization. The constraints are kept rather accurately. Only for $x_1 < 0.4$, the error is slightly increased. Note that the networks are not trained or tuned for their exploitation by associative completion.

A compact representation of the results is displayed in Fig. 6.10 (left), where for each constraint configuration the performance statistics are computed according to (6.3). The task constraints are fulfilled in the range of the inverse model precision for single controlled inputs besides for $x_1$ (compare Fig. 6.9 (right) and Fig. 6.10 (left)).

Interestingly, for the combined constraints – controlling two task space inputs at a time – the performance depends also on the combination of driven input variables (compare Fig. 6.10 (right)). While the combination of $x_1 \wedge x_2$ and $x_2 \wedge x_3$ yield good results, the combination of $x_1 \wedge x_3$ is significantly worse than in the other cases. Some networks display outlier performances with errors up to $10cm$ on average. This statistically robust effect of deteriorated performance indicates an underlying regularity of this particular combination of constraints.

Investigating the spatial distributions of errors more closely in Fig. 6.12 (and also Fig. 6.11 (left) for driving only $x_1$), sheds light on this issue. We obtain smooth error surfaces (compare Fig. 6.12 (left) and (right)) with moderate errors which means that we can expect similar errors for nearby targets. At the borders of the cube, however, errors increase rather rapidly. In particular, the error increases for small inputs $x_1$. Note that the distribution of inputs $x_1$ is not centered around zero. This configuration seems to have a rather destructive influence on the approximated mapping. Note, however, that the networks are not trained for this variable selection of control inputs and hence associative completion may be limited in some application scenarios. In particular, continuous attractor dynamics of the output feedback loop due to released control inputs are naturally brittle [150]. Further tuning of the regularization parameters by Algorithm 5.4 for different control configurations may alleviate these difficulties. Another approach to cope with multi-stable and continuous attractor dynamics is presented in the next chapter. Then, the associative model is also more robust when operated in associative completion modes. However, the results presented in this chapter show that robust offline learning of bidirectional association and, to some extent, associative completion is achieved by the combination of read-out and reservoir regularization with balancing of contributions.

## 6.4 Dimensionality reduction and data reconstruction

In this section, the scalability of the AELM approach is demonstrated in an dimension reduction scenario. The network is trained to embed handwritten digits into a plane and to reconstruct digits from their low-dimensional representation. This scenario shows that learning of bidirectional mappings of unequal dimensioned input and output variables is possible in the AELM.

Besides the exemplifying character of this task with respect to learning of bidirectional mappings, there is an upcoming thread of papers that considers neighborhood-preserving embedding *functions* [155, 156]. This approach is appealing because embedding new data points after learning of the function does not require to rerun the embedding algorithm with the additional data points. In this thesis, bidirectional mappings are learned that additionally to the embedding function enable the reconstruction of data in the original space by "navigating" through the embedding space.

First, the data and the embedding method to generate supervised training data in the embedding space are presented. Then, the performance of trained bidirectional mappings is evaluated on the embedding and reconstruction function.

### 6.4.1 Embedding handwritten digits using t-SNE

I consider images of handwritten digits from the MNIST data set. The MNIST data set is commonly used to benchmark image reconstruction and classification methods. For the following experiments, a subset of the "MNIST-basic" data set [157] is used, which comprises 2.000 images $\mathbf{x}$ of handwritten digits from 0 to 9 in a centered and normalized $28 \times 28$ pixel format.

I embed the data into the two-dimensional plane by t-Stochastic Neighbor Embedding (t-SNE) [158] in order to generate training data pairs $(\mathbf{x}, \mathbf{y})$. The t-SNE embedding yields for each image $\mathbf{x}$ a two-dimensional embedding vector $\mathbf{y}$. The MATLAB implementation of t-SNE [159] is used with default parameter configuration. The resulting embedding is shown in Fig. 6.13 (left) for some example images and in Fig. 6.13 (right) the embedding is shown for all 2000 images (black dots). Note that the embedding tries to preserve the topology of the data in both spaces with a rather smooth relation, i.e. similar images are mapped to similar positions, which indicates an homeomorph mapping between both spaces. Thus, an unique inverse mapping to the embedding function exists and can be learned by an AELM.

Note that t-SNE is used for training data generation only. It is also possible to integrate a neighbor-preserving criterion directly in the optimization of the parameterized function [155, 156]. Here, however, I focus on the principle functioning of the bidirectional mapping approach in case of high-dimensional data and leave the integration of the embedding process into the network training to future work.

To learn both, the embedding projection and the inverse mapping, 100 independently initialized AELM networks with $R = 400$ hidden neurons are used. Thus, the hidden layer is a non-linear down-projection of the combined, 786-dimensional input-output space. The hidden neurons have *tanh* activation functions (3.5) and the input weights $\mathbf{W}^{inp}$ and $\mathbf{W}^{fdb}$ are initialized in $[-\frac{1}{784}, \frac{1}{784}]$ and $[-1, 1]$, respectively. For training, a randomly chosen subset of the data comprising 75% of the samples are chosen. The remaining 500 samples are used to evaluate the generalization abilities of the learned mapping.

Read-out learning is conducted with $\alpha^{out} = \alpha^{rec} = 0.1$. I balance input and output contributions to the network activity with $g^{inp} = g^{fdb} = 0.5$ and regularization parameters $\beta^{inp} = 0.01$ and $\beta^{fdb} = 0.1$. This balancing is particularly important in this scenario in order to compensate for the strong inequality of the input and output dimensions.

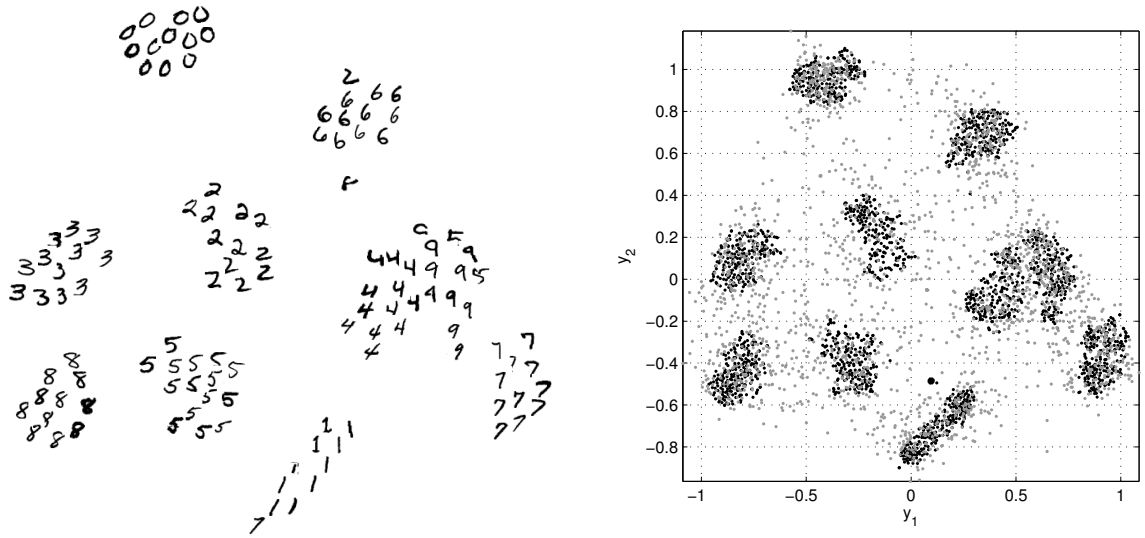



Fig. 6.13: Exemplary images plotted according to the t-SNE embedding (left). Positions of images in embedding space for the entire data set (right): t-SNE (black dots) and trained AELM (gray dots).

### 6.4.2 Learning to project handwritten digits to a plane

We first consider the embedding projection from the image space to the plane. The estimated projections for the entire data set are shown in Fig. 6.13 (right, gray dots) for an exemplary network and in Fig. 6.14 (left) for a subset of the input images. The embedding is not as clustered as the output of the t-SNE algorithm, but nevertheless shows a decent, smooth embedding with similar inputs at similar positions in the embedding space.

To access the network performance quantitatively, I compute the dimension-normalized mean square error

$$NMSE = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{O}||\mathbf{y}_k - \hat{\mathbf{y}}_k||^2, \tag{6.4}$$

where $K$ is the number of samples, $O$ is the dimension of the outputs, $\mathbf{y}_k$ are the embeddings according to t-SNE and $\hat{\mathbf{y}}$ the approximated embedding by the AELM. The error statistics of the AELM embeddings with respect to the embedding generated by t-SNE is displayed in Tab. 6.1 for training and test sets. The errors and their variances are small indicating robust learning of the output feedback-driven network dynamics. The overall topology of the t-SNE embedding is captured well by the trained networks.

### 6.4.3 Learning to reconstruct handwritten digits

We now consider the inverse mapping of the embedding which maps coordinates in the two-dimensional plane to data samples $\hat{\mathbf{x}}$, i.e. reconstructs images. Fig. 6.14 (right) shows reconstructions of images for a subset of the embedded data points. The characteristics of all ten digits are reproduced well and variations of the digit shape and orientation are also associated with neighboring positions in the embedding space.

The average reconstruction errors according to (6.4) calculated for input images $\mathbf{x}$ and reconstructed images $\hat{\mathbf{x}}$ are listed in Tab. 6.1. The errors show a similar approximation capability of the networks for the backward reconstruction in comparison to the learned forward embedding on the test set. Slight blurring of edges in the reconstructed images (see Fig. 6.14) is due to the "noisy" embedding by t-SNE which causes averaging effects in the read-out learning of the associative network: Quite different images are mapped to very similar positions $\mathbf{y}$ by t-SNE,

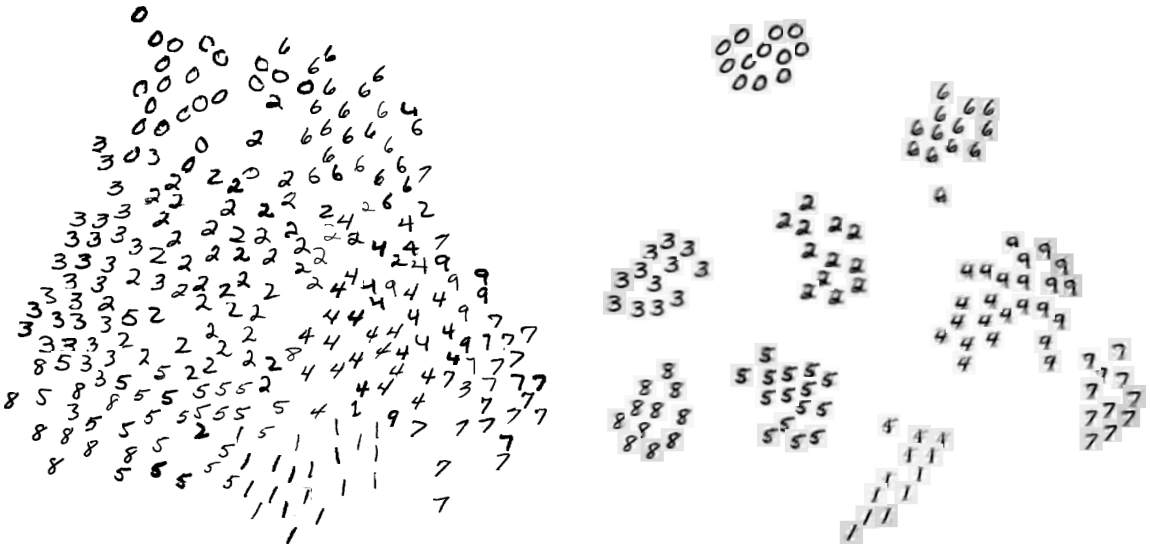


Fig. 6.14: Embedding and generation of digits by the trained associative network: Input images $\mathbf{x}$ are plotted at estimated positions $\hat{\mathbf{y}}$ (left). Reconstructed images $\hat{\mathbf{y}}$ are queried from and plotted at embedding positions $\mathbf{y}$ of t-SNE (right).

resulting in a good inter/intra cluster variance ratio. In contrast to this sample-based clustering, the connectionist representation in the AELM is biased towards more continuous mappings which can be seen also in the forward, embedding mapping direction (compare stretching of clusters in Fig. 6.13 (right)).

Picking up the discussion about learning, regularization and stability, I conclude from the small error variances in Tab. 6.1 that bidirectional mappings of invertible mappings can be robustly learned in AELM networks. I.e. error amplification of the output feedback dynamics are prevented by applying regularization of the hidden layer in combination with balanced contributions of inputs and outputs to the network activity. The unequal number of dimensions of inputs and outputs can be handled and do not disturb the bidirectional mapping. This final example of learning embeddings and their inverse mapping for data reconstruction is a promising application domain for AELM networks.

| | **Embedding** | **Reconstruction** |
|---|---|---|
| **Train** | $0.012 \pm 3.58 \cdot 10^{-4}$ | $0.044 \pm 7.15 \cdot 10^{-4}$ |
| **Test** | $0.047 \pm 4.01 \cdot 10^{-3}$ | $0.043 \pm 2.54 \cdot 10^{-4}$ |

Tab. 6.1: Normalized mean square errors (6.4) of the embedding and reconstruction mapping averaged over 100 network initializations with standard deviations.

## 6.5 Discussion

The combination of read-out and reservoir regularization introduced in Chapter 4 and Chapter 5 yields robust performance when learning bidirectional mappings of invertible problems. The balancing of contributions from the inputs and outputs to the hidden layer contribute to this robust performance. I report that conducting the experiments above without these mechanisms fails in the sense that almost none of the trained networks achieves a comparable performance. The presented results show also that regularization to achieve output feedback stability tunes the model into a range of excellent generalization. That is, stability and generalization benefit from regularization.

Chapter 7

# Representing and resolving ambiguity with output feedback

## 7.1 Learning and selecting multiple solutions

The previous chapter considered the associative learning of invertible models by providing data with a unique inverse mapping for learning. This restriction to invertible relations between inputs and outputs is, however, not the typical case. There are often multiple valid solutions to the inverse problem due to a non-injective forward mapping, e.g. multiple joint configuration that cause the same end effector position. It is particularly difficult to cope with multiple solutions in learning architectures: The non-convexity of inverse mappings typically causes the average of two valid solution to be an invalid solution.

In this chapter, it is first shown that output feedback resolves the non-convexity problem and principally enables the representation of multiple solutions of an inverse model in a single network. During network exploitation, ambiguities are resolved dynamically by iterating the output feedback loop, i.e. the network settles to an attractor that represents a particular solution depending on initial conditions. The recall of one of the stored solutions, however, largely depends on the network initialization: If learning only imprints fixed-point conditions into the output feedback dynamics, the basins of attraction can be arbitrary in size and shape. I therefore propose an algorithm to program attractor dynamics that shapes the basins of attraction to each solution manifold explicitly. In a next step, desired features of output feedback dynamics to represent and resolve ambiguities are derived which address important issues that I identify in the initial discussion. Finally, the dynamical approach to cope with ambiguous mappings is demonstrated in a series of experiments and properties of the proposed methodology are analyzed.

We focus on Associative Extreme Learning Machines (AELMs, Sec. 3.3.4) in this chapter and concentrate on the output feedback dynamics of the forward path first. Then, full AELM training of forward and ambiguous inverse models is presented in kinematic learning scenarios. In these examples, static forward and inverse mappings are trained using attractor-based computation. Finally, the AELM with multi-stable output feedback dynamics is compared to transient-based resolution of ambiguities in a sequence transduction task.

### 7.1.1 Output feedback resolves ambiguity

Consider a real-valued inverse mapping with two solution branches. For instance, the inverse mapping of a parabola has two solutions (see gray data points in Fig. 7.1 (a)). If a pure feed-forward model is trained on such an ambiguous data set, learning averages over both solution branches (see black dots produced by a feed-forward network in Fig. 7.1 (a), details of the

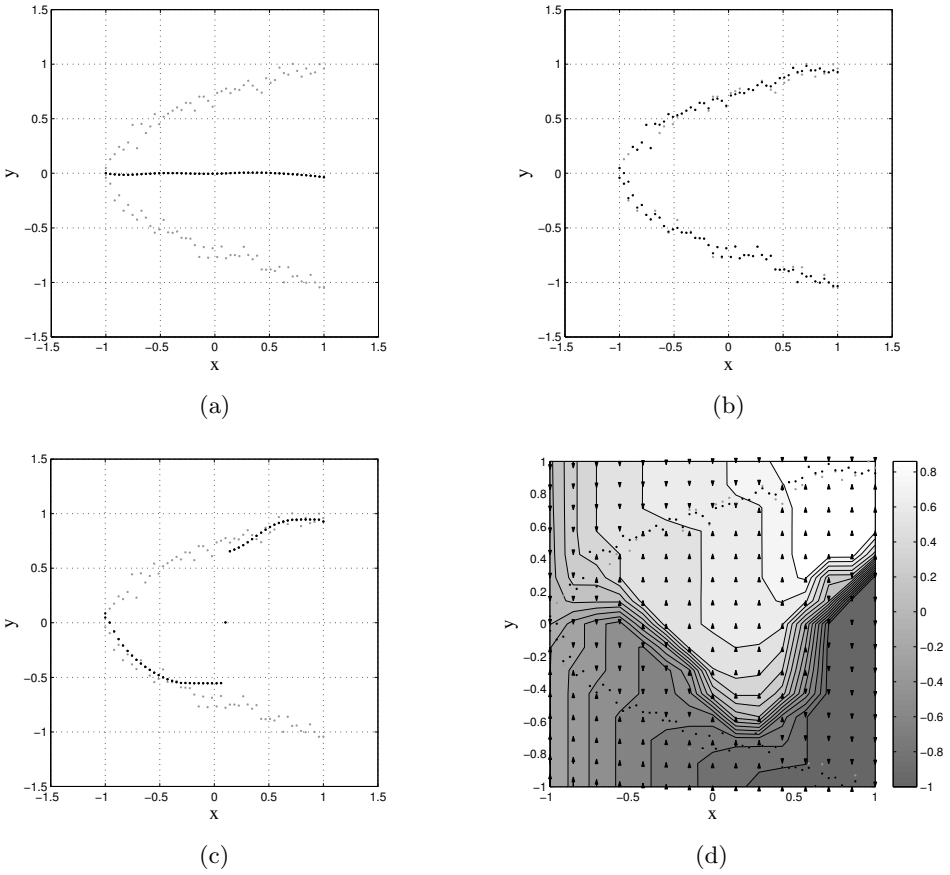



Fig. 7.1: Learning an ambiguous mapping with two solution branches. Pure feed-forward models average both solution (a). AELM with output feedback but without Algorithm 7.6 (b)–(d): Training imprints fixed points at the data examples (b). Switching ambiguity resolution for smoothly varying input $x$ from $-1$ to 1 (c). Point of attraction $\hat{y}$ (coded in gray scale) as function of the input $x$ and initial condition $y$ (d). Additionally, the phase portrait of the output feedback dynamics is shown by arrows in (d).

training and initialization can be found in Appendix A.3.2). This can be understood as the common behavior of feed-forward networks with least squares learning to noisy data: Variation of target outputs for the same input are assumed to be due to noise and are averaged. The average solution, however, is not necessarily a valid solution if the valid output set is non-convex (see Fig. 7.1 (a) and discussion in [15]).

More formally, consider a feed-forward network, e.g. an Extreme Learning Machine (ELM, see Sec. 3.3.2 for details of the model). Then, ambiguous data pairs $(\mathbf{x}_n, \mathbf{y}_n)$ and $(\mathbf{x}_m, \mathbf{y}_m)$ with $\mathbf{x}_n = \mathbf{x}_m$ and $\mathbf{y}_n \neq \mathbf{y}_m$ are mapped to the same hidden state and thus ambiguous outputs can not be distinguished:

$$\begin{aligned}\hat{\mathbf{y}}_n(\mathbf{x}_n) &= \mathbf{W}^{out}\mathbf{h}(\mathbf{x}_n) \\ &= \mathbf{W}^{out}\sigma(\mathbf{W}^{inp}\mathbf{x}_n) \\ &= \mathbf{W}^{out}\sigma(\mathbf{W}^{inp}\mathbf{x}_m) = \hat{\mathbf{y}}_m(\mathbf{x}_m)\end{aligned}$$

Learning of the feed-forward model can consequently not disentangle both solutions and averages the target outputs for the same inputs [15].

Averaging effects can be prevented by utilizing a combined representation of inputs and outputs, for example in the associative variant of Extreme Learning Machines (AELM, see

Sec. 3.3.4 for details of the model). The associative setup resolves the non-convexity problem arising from ambiguous data pairs by utilizing a mixed representation $\mathbf{h}(\mathbf{x}_n, \mathbf{y}_n) \neq \mathbf{h}(\mathbf{x}_m, \mathbf{y}_m)$ of inputs and outputs in the hidden layer. We have

$$\begin{aligned}\hat{\mathbf{y}}_n(\mathbf{x}_n, \mathbf{y}_n) &= \mathbf{W}^{out}\mathbf{h}(\mathbf{x}_n, \mathbf{y}_n) \\ &= \mathbf{W}^{out}\sigma(\mathbf{W}^{inp}\mathbf{x}_n + \mathbf{W}^{fdb}\mathbf{y}_n) \\ &= \mathbf{W}^{out}\sigma(\mathbf{W}^{inp}\mathbf{x}_m + \mathbf{W}^{fdb}\mathbf{y}_n) \\ &\neq \mathbf{W}^{out}\sigma(\mathbf{W}^{inp}\mathbf{x}_m + \mathbf{W}^{fdb}\mathbf{y}_m) = \hat{\mathbf{y}}_m(\mathbf{x}_m, \mathbf{y}_m)\end{aligned}$$

for ambiguous data pairs with $\mathbf{x}_n = \mathbf{x}_m$ and $\mathbf{y}_n \neq \mathbf{y}_m$ as well as $\mathbf{W}^{fdb} \neq \mathbf{0}$. Therefore, learning of $\mathbf{W}^{out}$ can disentangle the ambiguity based on the different hidden states.

During network exploitation, outputs $\mathbf{y}$ are not available and the trained network is therefore applied in an output feedback-driven mode, i.e. estimated outputs are fed back into the network (compare Fig. 3.2 and (3.9)). I show that successful training according to Sec. 3.2.4 causes ambiguous data pairs to be fixed-points of the output feedback dynamics. Assume the outputs are reproduced sufficiently accurate by the trained network, i.e. $\hat{\mathbf{y}}(k) = \mathbf{W}^{out}\mathbf{h}(k, \mathbf{x}, \mathbf{y}) \approx \mathbf{y}$, if the network state $\mathbf{h}(k, \mathbf{x}, \mathbf{y})$ is teacher-forced to the desired input-state-output configuration at time step $k$. Then, releasing the output neurons from teacher-forcing fulfills the fixed-point condition $\mathbf{h}(k{+}1, \mathbf{x}, \hat{\mathbf{y}}(k)) = \mathbf{h}(k, \mathbf{x}, \mathbf{y})$. We have

$$\begin{aligned}\hat{\mathbf{y}}(k) &= \mathbf{y} \\ \Leftrightarrow \mathbf{W}^{out}\mathbf{h}(k, \mathbf{x}, \mathbf{y}) &= \mathbf{y} \\ \Leftrightarrow \mathbf{W}^{fdb}\mathbf{W}^{out}\mathbf{h}(k, \mathbf{x}, \mathbf{y}) &= \mathbf{W}^{fdb}\mathbf{y} \\ \Leftrightarrow \mathbf{W}^{inp}\mathbf{x} + \mathbf{W}^{fdb}\mathbf{W}^{out}\mathbf{h}(k, \mathbf{x}, \mathbf{y}) &= \mathbf{W}^{inp}\mathbf{x} + \mathbf{W}^{fdb}\mathbf{y} \\ \Rightarrow \sigma(\mathbf{W}^{inp}\mathbf{x} + \mathbf{W}^{fdb}\hat{\mathbf{y}}(k)) &= \sigma(\mathbf{W}^{inp}\mathbf{x} + \mathbf{W}^{fdb}\mathbf{y}) \\ \Leftrightarrow \mathbf{h}(k{+}1, \mathbf{x}, \hat{\mathbf{y}}(k)) &= \mathbf{h}(k, \mathbf{x}, \mathbf{y}).\end{aligned}$$

That is, the training samples are fixed-points of the output feedback dynamics.

Fig. 7.1 (b) shows the estimated outputs of an AELM (black dots) after convergence of the output feedback dynamics. The network was initially teacher-forced to the training data (gray dots). Fig. 7.1 (b) shows that the training samples are fixed-points of the network dynamics, i.e. the training patterns were imprinted into the output feedback dynamics. This confirms the calculation above empirically and shows that the output feedback dynamics are not globally asymptotically stable: The output feedback dynamics have at least two fixed-points for the same input.

I analyze the local stability of the output feedback dynamics at the training samples in more detail. I therefore consider the eigenvalues of the linearized output feedback dynamics. That is, I calculate the Jacobian

$$\mathbf{J} = \begin{pmatrix} \frac{\partial y_1(k+1)}{\partial y_1(k)} & \cdots & \frac{\partial y_1(k+1)}{\partial y_O(k)} \\ \vdots & & \vdots \\ \frac{\partial y_O(k+1)}{\partial y_1(k)} & \cdots & \frac{\partial y_O(k+1)}{\partial y_O(k)} \end{pmatrix} \in \mathbb{R}^{O\times O} \quad \text{with} \quad \frac{\partial y_i(k+1)}{\partial y_j(k)} = \sum_{r=1}^{R} w_{ir}^{out} w_{rj}^{fdb} \sigma_r'(\mathbf{a}(k+\mathbf{1}))$$

of the output feedback dynamics

$$\mathbf{y}(k+1) = \mathbf{W}^{out}\sigma(\mathbf{W}^{inp}\mathbf{x} + \mathbf{W}^{fdb}\mathbf{y}(k)),$$

where the input $\mathbf{x}$ is externally driven and remains constant during network iteration. The Jacobian $\mathbf{J}$ is a linear approximation of the change at the outputs $\mathbf{y}(k{+}1)$ for a small perturbation of the outputs $\mathbf{y}(k)$ in the previous iteration step. The linearized system is locally stable if all eigenvalues of $\mathbf{J}(\mathbf{x}, \mathbf{y}(k))$ stay inside the unit circle of the complex plane [148].

For the AELM shown in Fig. 7.1, which was trained by the standard learning described in Sec. 3.2.4, most of the absolute eigenvalues stay in the region of local stability. We yield a distribution of absolute eigenvalues with mean 0.99 and standard deviation $10^{-3}$. However, for nine percent of the training samples, the absolute eigenvalue is greater than one which indicates an instable linear system for this training sample. The maximal absolute eigenvalue of $\mathbf{J}$ for all inputs and outputs in the data is 1.0026. Although absolute eigenvalues above unity indicate that the network was not programmed to have a fixed-point at these samples, note that this is only a linear approximation of the true output feedback dynamics. It might be that the output feedback dynamics have an equilibrium very close to the training samples with eigenvalues of the Jacobian greater than unity and in this sense learning of fixed-point conditions is successful in Fig. 7.1 (b).

The important question is whether these fixed-points have a local basin of attraction. For example, does the network reside in the same solution manifold if the input to the network changes smoothly? Fig. 7.1 (c) shows the network output for a linearly increasing input $x = -1, \ldots, 1$ starting from the lower solution. First, the network stays in the lower solution branch. But then the dynamics bifurcate with increasing $x$, i.e. a small change of the input causes to switch the ambiguity resolution scheme and the upper solution branch is selected. This indicates that the basin of attraction of the lower solution becomes narrow for inputs $x \approx 0.1$ which results in an undesired bifurcation. That this is indeed the case can be seen in Fig. 7.1 (d) which illustrates the point of attraction (coded in gray scale) depending on initial states $\mathbf{h}(x, y)$ in the input-output plane. The basin of attraction of the lower solution for $x \approx 0.25$ (along the vertical slice in Fig. 7.1 (d)) is much smaller than the basin of attraction for the upper solution at this input. The key question is how basins of attraction be shaped more robustly.

### 7.1.2 Programming multi-stable output feedback dynamics

Although the associative network setup is sufficient to prevent averaging of ambiguous data samples during learning, the output feedback dynamics will not exhibit equally spread basins of attraction when only the fixed-point conditions are programmed. To imprint multiple basins of attraction for a single input, I follow the programming dynamics approach introduced in [126] (Sec. 4.2 of this thesis) and extend the training data set by synthesized sequences to promote attraction to the data samples. For a given input sample $\mathbf{x}_n$, synthesized output sequences

$$\tilde{\mathbf{y}}_n^s(k) = \mathbf{y}_n + (1 - k/K)^2 \nu^s \quad \text{with } k = 0, \ldots, K \tag{7.1}$$

are generated, where $s = 1, \ldots, S$ denotes the index of the sequence and $\nu^s \in \mathbb{R}^O$ is a small perturbation. Note that also different sequence generation rules than (7.1) can be considered. In particular, modeling the kind of attraction to the fixed-points by utilizing trajectory data is possible. In this thesis, however, I focus on sequence generation by (7.1). The concept of sequence generation to program multi-stable attractor dynamics is illustrated in Fig. 7.2.

For each sequence, the corresponding network states $\mathbf{h}(\mathbf{x}_n, \tilde{\mathbf{y}}_n^s(k))$ are collected. Attraction to the desired output $\mathbf{y}_n$ is enforced by mapping states $\mathbf{h}(\mathbf{x}_n, \tilde{\mathbf{y}}_n^s(k))$ to outputs $\tilde{\mathbf{y}}_n^s(k+1)$. This can be accomplished by the linear regression

$$\mathbf{W}^{out} = (\mathbf{H}^T \mathbf{H} + \alpha^{out} \mathbb{1})^{-1} \mathbf{H}^T \tilde{\mathbf{Y}}, \tag{7.2}$$

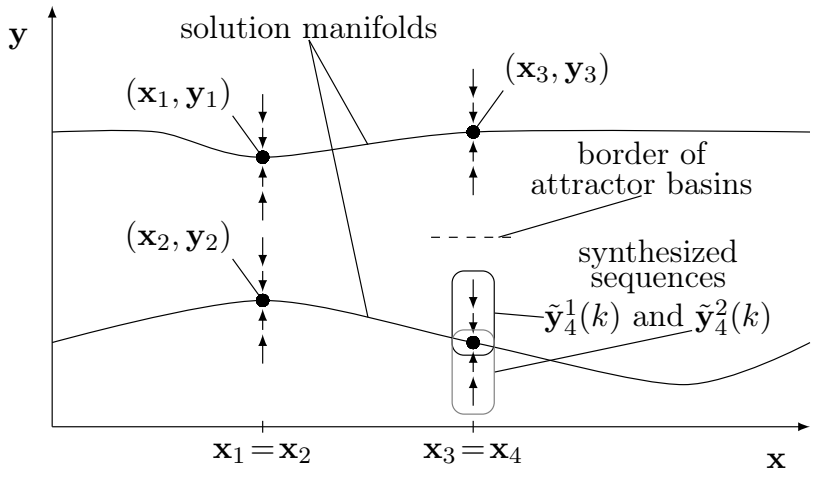



Fig. 7.2: Programming multi-stable attractor dynamics. Consider a scalar input-output mapping with two solution branches set up by data points $(\mathbf{x}_1, \mathbf{y}_1)$ and $(\mathbf{x}_2, \mathbf{y}_2)$ with $\mathbf{x}_1 = \mathbf{x}_2$ and $\mathbf{y}_1 \neq \mathbf{y}_2$. Attraction to manifolds is promoted by synthesized sequences $\tilde{\mathbf{y}}_n^s(k)$ for each data point $n = 1, \ldots, N$. The border of attractor basins will be formed between both solution manifolds.

where

$$\mathbf{H} = \begin{pmatrix} \mathbf{h}_1^1(0)^T \\ \vdots \\ \mathbf{h}_1^1(K-1)^T \\ \vdots \\ \mathbf{h}_1^S(0)^T \\ \vdots \\ \mathbf{h}_N^S(K-1)^T \end{pmatrix} \text{ and } \tilde{\mathbf{Y}} = \begin{pmatrix} \tilde{\mathbf{y}}_1^1(1)^T \\ \vdots \\ \tilde{\mathbf{y}}_1^1(K)^T \\ \vdots \\ \tilde{\mathbf{y}}_1^S(1)^T \\ \vdots \\ \tilde{\mathbf{y}}_N^S(K)^T \end{pmatrix}$$

collect all hidden states and desired outputs in respective matrices. The parameter $\alpha^{out}$ in (7.2) weights the contribution of a regularization constraint. Multiple attractors for the same input are trained if there are data samples $(\mathbf{x}_n, \mathbf{y}_n)$ and $(\mathbf{x}_m, \mathbf{y}_m)$ with $\mathbf{x}_n = \mathbf{x}_m$ but $\mathbf{y}_n \neq \mathbf{y}_m$. Note that there is no additional knowledge about the data necessary to implement uni-, bi- or multi-stable attractor dynamics.

Algorithm 7.6 summarizes the attractor programming technique, which is analogously also applied to the input reconstructing weights $\mathbf{W}^{rec}$ of the backward model: Then, input sequences $\tilde{\mathbf{x}}_n^s(k)$ are synthesized and the resulting hidden states $\mathbf{h}(\tilde{\mathbf{x}}_n^s(k), \mathbf{y}_n)$ are harvested. Equation (7.2) alters accordingly to

$$\mathbf{W}^{rec} = (\mathbf{H}^T\mathbf{H} + \alpha^{rec}\mathbb{1})^{-1}\mathbf{H}^T\tilde{\mathbf{X}}. \tag{7.3}$$

I apply a simple scheme to chose perturbations $\nu^s$ such that the sequences $\tilde{\mathbf{y}}_n^s(k)$ cover all directions in the output space by generating two sequences along each dimension (compare Fig. 7.2). I yield $S = 2 \cdot O$ sequences for the forward model from inputs to outputs with $\nu^s = l\mathbf{e}_i$, where $\mathbf{e}_i$ is the canonical basis vector with the $i$-th component equal to unity and $l$ scales the length of the sequence. Thus, Algorithm 7.6 enlarges the training set by factor $S \cdot (K-1)$, where I typically use $S = 2 \cdot O$ and $K = 3$. Learning is nevertheless feasible due to the efficient read-out training by linear regression. Note that for $K = 1$ or $l = 0$, Algorithm 7.6 is equal to the standard attractor-based learning discussed in Sec. 3.2.4.

**Algorithm 7.6** Programming fixed-point attractor dynamics

**Require:** network, training data $(\mathbf{x}_n, \mathbf{y}_n)$ with $n = 1, \ldots, N$,
sequences per sample $S$, steps per sequence $K$

```
for n = 1, ..., N do
  for s = 1, ..., S do
    synthesize sequence ỹ^s_n(k) = y_n + (1 − k/K)^2 ν^s for k = 0, ..., K
    harvest states h(x_n, ỹ^s_n(k)) for k = 0, ..., K − 1
    append harvested states to H and outputs ỹ^s_n(k) for k = 1, ..., K to Ỹ
  end for
end for
read-out learning by (7.2)
```

### 7.1.3 Robust solution selection by output feedback dynamics

The application of algorithm Algorithm 7.6 results in larger and more equally distributed basins of attraction. Fig. 7.3 shows results for the AELM trained using Algorithm 7.6 on the inverse parabola example. Fig. 7.3 (a) displays the fixed-points of the output feedback dynamics if the network is teacher-forced to the training samples first. In comparison to Fig. 7.1 (b) where almost each training samples is a fixed-point on its own, the training samples are merged to smooth solution manifolds in the network using Algorithm 7.6. The fused attractor manifold is important for generalization along the manifold which will be demonstrated later on in this chapter.

Fig. 7.3 (b) and (c) show further that both solution branches are robustly imprinted into the network dynamics in the sense that changing the input $x$ does not change the ambiguity resolution scheme. The robust attraction to both solutions is confirmed by the results displayed in Fig. 7.3 (d) which shows the point of attraction (coded in gray scale) depending on the initial condition $\mathbf{h}(x, y)$. Both solutions have a well-distributed basin of attraction with a sharp border which minimizes the risk for spurious states and the spatial extension of saddle nodes.

I again analyze the linearized output feedback dynamics by means of the eigenvalues of the Jacobian as in Sec. 7.1.1. The distribution of absolute eigenvalues for all training samples displays a decreased mean value of 0.58 with a standard deviation of 0.15. The maximal absolute eigenvalue resides below the border to instability at 0.998, i.e. the eigenvalues of the linearized system indicate that all training samples are fixed-points of the output feedback dynamics. Note, however, that the analysis based on the Jacobian is a linear approximation of the nonlinear output feedback dynamics and the trainings samples are actually merged to a solution manifold (compare gray data samples and black fixed-points of the output feedback dynamics in Fig. 7.3 (a)). However, the decreased eigenvalues of the linear system show that learning of multi-stable output feedback dynamics is clearly more robust if Algorithm 7.6 is used.

Note that in case of a single solution branch it is not necessary to shape attractor basins of the output feedback dynamics explicitly. The results in Chapter 6 clearly show that the fixed-point conditions are sufficient for training globally stable output feedback dynamics in combination with regularization of the hidden and read-out layers. Algorithm 7.6 enables to program bistable output feedback dynamics such that the network can represent two solution branches. That is, the network learns a multi-valued function. Moreover, the synthesized sequences make the fine-tuning of regularization parameters $\alpha^{out}$ and $\alpha^{rec}$ for each network, e.g. according to Algorithm 5.4, dispensable. The synthesized sequences by themselves act as regularization by providing data pairs in the vicinity of the original data samples. Therefore, learning is rather robust for sufficiently large reservoir regularization ($\beta^\star$ in (5.9)–(5.11)) and read-out regularization parameters ($\alpha^\star$ in (3.14) and (3.16)). The generation of many output sequences can even make output regularization dispensable. However, without reservoir regularization, I could not achieve (locally) stable output feedback dynamics in the presented example.

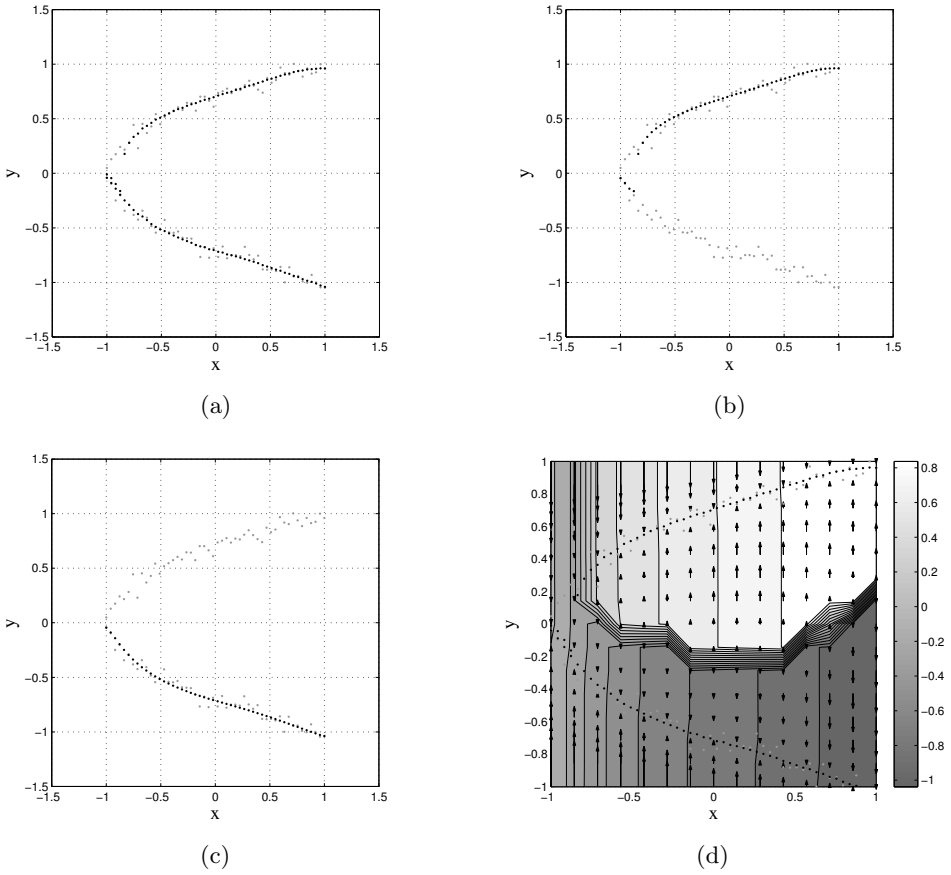



Fig. 7.3: Learning an ambiguous mapping with two solution branches in an AELM using Algorithm 7.6. Starting from teacher-forced network states on the training samples, the dynamics are attracted to the learned solution manifolds (a). Both solutions are robustly recalled depending on the initial condition (inputs $x$ from 1 to $-1$) (b)–(c). Point of attraction $\hat{y}$ (coded in gray scale) as function of the input $x$ and initial condition $y$ (d). Additionally, the phase portrait of the output feedback dynamics is shown by arrows in (d).

### 7.1.4 Multi-stable and continuous attractor dynamics

The initial example in Sec. 7.1.1 and Sec. 7.1.3 demonstrates the capability of learning two solution branches with an AELM, i.e. the output feedback dynamics are bi-stable. The output feedback dynamics are further parameterized by an input which enables to smoothly query outputs along each solution branch. In this section, I show that multi-stable output feedback dynamics can be trained by Algorithm 7.6. Moreover, in the limit of many desired attractors with decreasing spatial distance, Algorithm 7.6 programs continuous attractor dynamics (for an introduction of continuous attractor dynamics Sec. 2.3.6). In particular, I tackle the following questions: How many solutions can be stored by the output feedback dynamics? How does the number of solutions depend on their spatial distribution in output space? And, are there spurious attractor states?

We focus on the output feedback dynamics only for visualization purposes, i.e. the AELM has no inputs. The results nevertheless transfer to network dynamics parameterized by inputs. The output space is two-dimensional and the AELM has two output neurons accordingly. Consider a set of desired outputs $\{\mathbf{y}_n\}$ that shall be attractors of the output feedback dynamics. I train AELMs with $R = 300$ hidden neurons according to Algorithm 7.6 using regularization of the hidden layer. Detailed model parameters can be found in Appendix A.3.3.

Starting with a single training sample $\mathbf{y}_1 = (0, 0)$, we obtain a single fixed-point of the output feedback dynamics shown in the first row of Tab. 7.1: The second column in Tab. 7.1 shows the effective point of attraction starting from several initial conditions in the output space. The third column in Tab. 7.1 displays the phase portrait of the output feedback dynamics. The fourth column in Tab. 7.1 shows the absolute values of the change in the output neurons which corresponds to the step width the network dynamics descent the energy landscape in one iteration of the output feedback dynamics. The single fixed-point dynamics correspond to the case of global attractor dynamics shown in Fig. 2.8 (left). Parameterizing the point of attraction by an additional input enables the approximation of a single solution branch. This is the typical case demonstrated throughout Chapter 6.

If we add an additional sample to the training set, we obtain the bi-stable dynamics illustrated in the second row of Tab. 7.1. The concept of bi-stable attractor dynamics is illustrated in Fig. 2.8 (middle). I successively increase the number of training samples placed along a non-linear manifold to show that multi-stable output feedback dynamics can be imprinted into the network by Algorithm 7.6 (see remaining rows in Tab. 7.1). I use a constant parameter setting throughout the experiments, in particular $l = 0.1$, which shows the robustness of programming multi-stable output feedback dynamics with the proposed approach. A more detailed analysis of the effect of $K$ and $l$ on the performance of Algorithm 7.6 follows in Sec. 7.3.

With decreasing distance between desired attractor states, it becomes more and more difficult to separate attractor basins. In particular for initial conditions in between the basins of attraction, i.e. starting in the saddle nodes, convergence to a desired attractor can fail (see arrows that end in between training samples in the third and fourth row of Tab. 7.1). The saddle node between attractors can be seen in the fourth column of Tab. 7.1: The rate of change is low in between two attractor states and starting close to saddle nodes may result in very slow convergence. However, the outputs "get stuck" at the saddle nodes in the worst case which is far more desirable than divergence of the outputs to some undefined state. Saddle nodes and their effects are an intrinsic property of the dynamical approach to multi-valued function approximation and not specific to the presented model.

In the limit of many training samples along a continuous manifold, the attractor basins merge to a continuous attractor manifold. These continuous attractor dynamics emerge first for $N = 7$ and generalize to the manifold hidden in the training data for $N = 10$ and $N = 30$ (compare last three rows in Tab. 7.1). Continuous attractor dynamics model the limit case of infinitely many solution to a problem, i.e. a multi-valued function whose set of solutions is a continuum, and correspond to the illustration in Fig. 2.8 (right). A continuous valley of attraction with small state changes can also be observed in the fourth column in Tab. 7.1 for $N \geq 7$. To understand the shaping of continuous attractor dynamics by Algorithm 7.6, consider overlapping sequences produced by Algorithm 7.6 for nearby outputs $\mathbf{y}_n \approx \mathbf{y}_m$ and the same input $\mathbf{x}$. We then have similar network states $\mathbf{h}(\mathbf{x}, \tilde{\mathbf{y}}_n(k)) \approx \mathbf{h}(\mathbf{x}, \tilde{\mathbf{y}}_m(k))$ for the overlapping sequences $\tilde{\mathbf{y}}_n(k)$ and $\tilde{\mathbf{y}}_m(k)$ that are mapped to conflicting outputs $\tilde{\mathbf{y}}_n(k+1) \neq \tilde{\mathbf{y}}_m(k+1)$. Along anti-parallel directions, learning will effectively average out the desired changes of the outputs. Similar directions of the sequences still shape the output feedback dynamics. We obtain effective target sequences for the learning which approach a line orthogonal to the connecting line between two

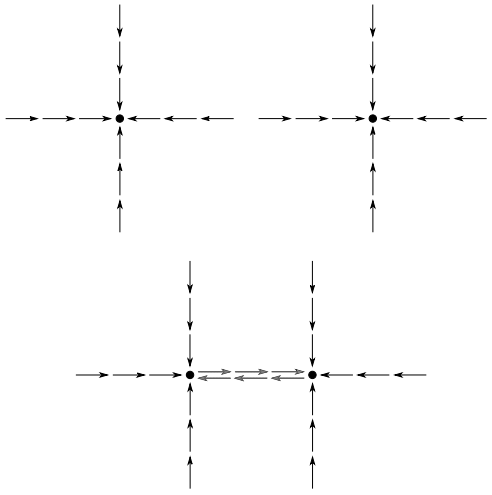
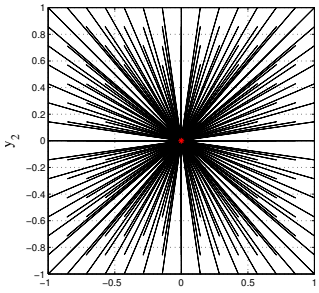
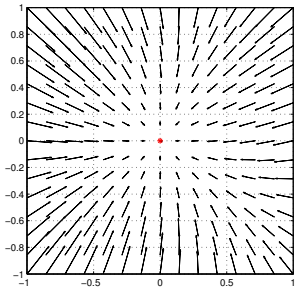
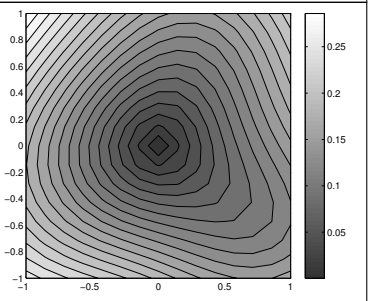
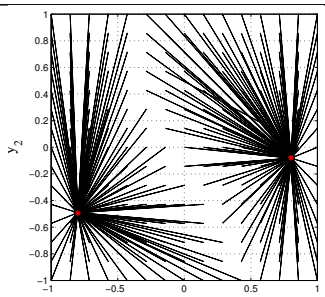
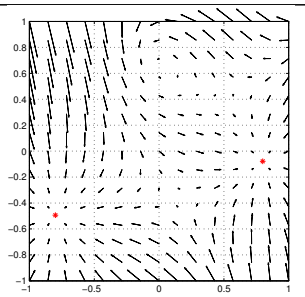
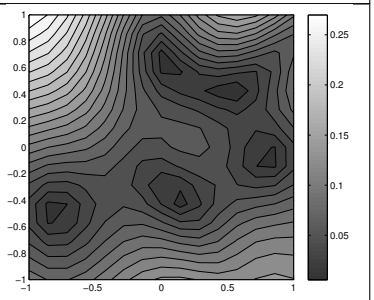
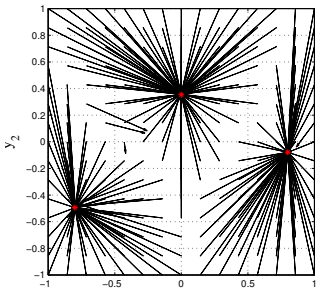
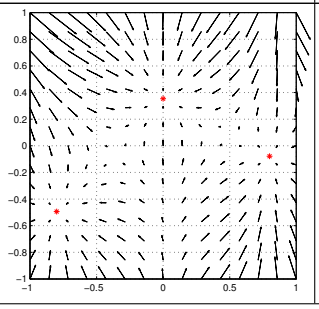
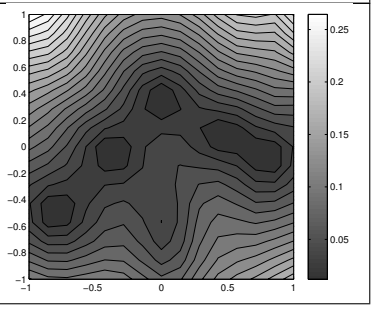
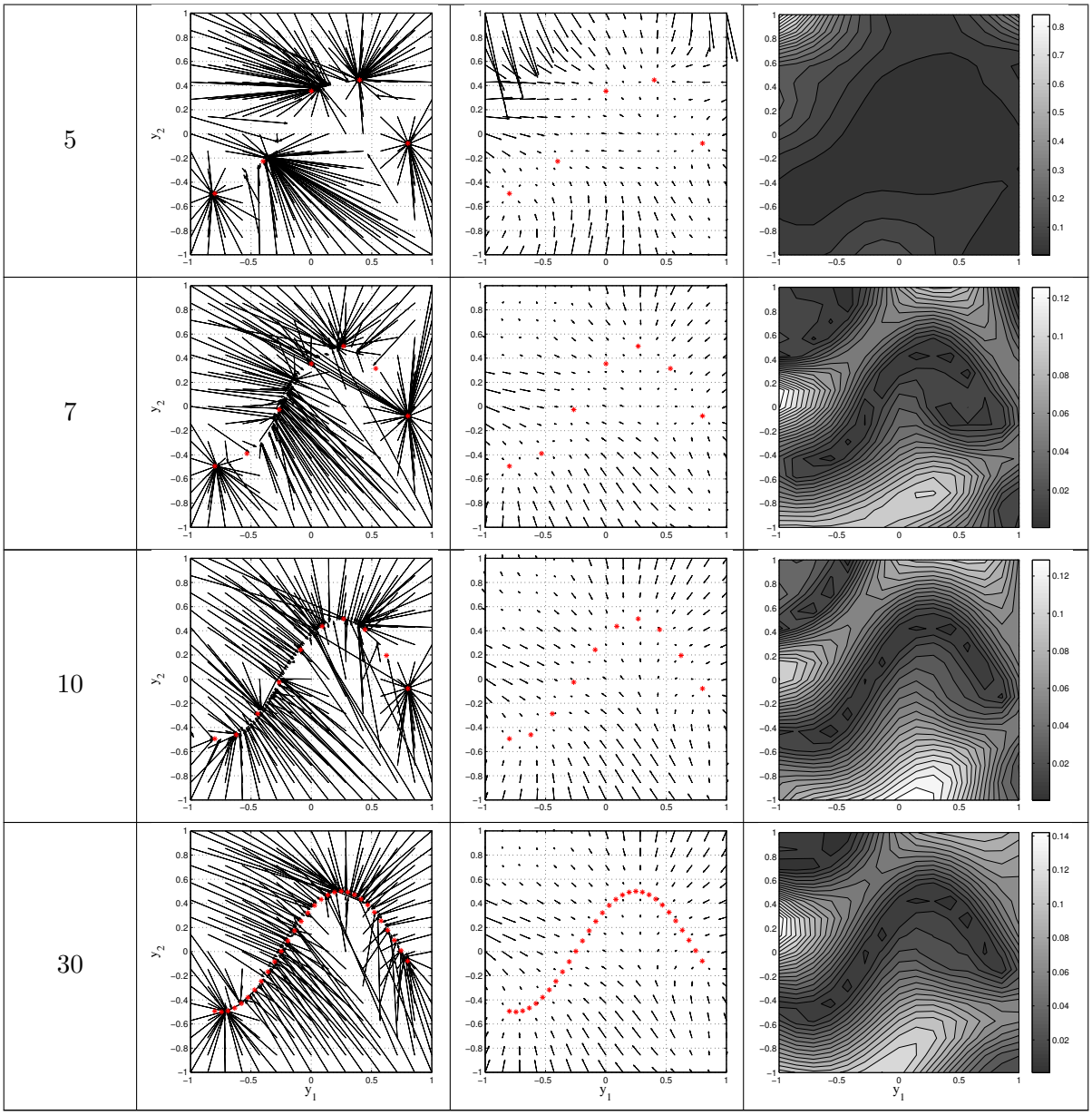


Fig. 7.4: Two data points with synthesized sequences in the output space (top). If data points are closer to each other, synthesized sequences may overlap (bottom).

output samples $\mathbf{y}_n$ and $\mathbf{y}_m$ of the original data set. Fig. 7.4 illustrates the overlap of synthesized sequences $\tilde{\mathbf{y}}_n^s(k)$. Overlapping sequences can occur if the sequence length $l$ is increased, or data points are close to each other. Note that the choice of a particular solution on the non-linear manifold in output space is not determined by an input but by initial conditions of the output feedback dynamics. That is, the network models infinitely many solutions to a single (in this case constant) input.

Tab. 7.1: Multi-stable output feedback dynamics. For the same input, say $\mathbf{x} = 0$, the number of ambiguous outputs $N$ (left column) is increased (see red markers in columns two and three). In the second column, the arrows show the point of attraction for initial conditions in the output space: The network is first teacher-forced to $\mathbf{h}(\mathbf{y})$. Then, the output feedback dynamics are iterated until convergence according to Algorithm 6.5 which yields the estimated output $\hat{\mathbf{y}}$. Arrows in the second column are drawn from $\mathbf{y}$ to $\hat{\mathbf{y}}$. In the third column, the respective phase portrait of the system is shown by means of the vector field set up by arrows $\mathbf{y} \to \hat{\mathbf{y}}(2)$. That is, the network is teacher-forced to the initial condition $\mathbf{h}(k = 1, \mathbf{y})$ and then iterated for a single step which yields the estimated output $\hat{\mathbf{y}}(2)$. Finally, the fourth column displays the absolute state change $||\hat{\mathbf{y}}(2) - \mathbf{y}||$, the length of the arrows in the third column.

| $N$ | **Attractor Dynamics** | **Phase Portrait** | **Absolute State Change** |
|---|---|---|---|
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |




In this section, I use the adopted convergence criterion in Algorithm 6.5 to stop iteration of the dynamics early: Learned continuous attractor basins are never perfectly level (see discussions in [57] and [150]) and dynamics typically drift slowly along the attractor valley. Algorithm 6.5 stops iteration of the output feedback dynamics approximately when the outputs reach the attractor valley, i.e. the closest (in terms of the number of network iterations) output in the attractor valley is returned by Algorithm 6.5. Note that Algorithm 6.5 is applied throughout the experiments displayed in Tab. 7.1 and performs well also in case of fixed-point attractor dynamics. It is not necessary to select a specific convergence criterion for each network iteration which would be cumbersome because one then needs to know the kind of attractor in advance.

## 7.2 Properties of the dynamical approach to ambiguity

The previous section demonstrated the basic capabilities of output feedback dynamics to represent multi-valued functions trained from ambiguous data samples. To proceed in a more principled way, I define desired properties of the output feedback dynamics and thus the approximated multi-valued functions in this section.

### 7.2.1 Robustness of solution branches to input perturbations

An important property of output feedback dynamics in the context of multiple, continuous solution branches is that for a small change of the input, the output resides on the same solution branch. That is, the output feedback dynamics shall be robust against small perturbations of the input and generalize along a solution manifold rather than switching the ambiguity resolution scheme.

More formally, if $\hat{\mathbf{y}}(\mathbf{x})$ belongs to a particular solution manifold defined by the set $Y_1$, then the output $\hat{\mathbf{y}}(\mathbf{x}+\nu)$ for a small perturbation $\nu$ of the input shall reside in the same solution branch, i.e. $\hat{\mathbf{y}}(\mathbf{x}+\nu) \in Y_1$. This property can be understood as robustness of the output feedback dynamics to small perturbations of the input and is illustrated for the first exemplary input in Fig. 7.5 (left).

### 7.2.2 Transition properties

The previous property demands the output feedback dynamics to stay in the same basin of attraction if the input is varied. The following properties focus on the ways of switching the ambiguity resolution scheme. That is, how can output feedback dynamics switch the solution branch if the output feedback dynamics are robust to perturbations of the input according to Sec. 7.2.1. There are two ways of switching the output feedback dynamics' basin of attraction: Either by forcing the outputs to a different basin of attraction, i.e. solution branch, or by driving the network inputs in a certain manner.

#### Output feedback-driven transition

If $\hat{\mathbf{y}}(\mathbf{x}) \in Y_1$, then perturbation of the actually fed back output, i.e. $\hat{\mathbf{y}} := \hat{\mathbf{y}}(\mathbf{x})+\nu$, shall result in $\hat{\mathbf{y}}(\mathbf{x}) \in Y_1$ for small perturbations $\nu$. This robustness of the output feedback dynamics to small perturbations is illustrated in the middle of Fig. 7.5 (left). For larger perturbations $\nu$ that drive the network output into another basin of attraction, e.g. corresponding to the solution branch $Y_2$, the output feedback dynamics will switch the ambiguity resolution scheme, i.e. $\hat{\mathbf{y}}(\mathbf{x}) \in Y_2$. The transition of the solution branch by strong perturbation of the output is shown in the right example in Fig. 7.5 (left).

#### Input-driven transition

The transition from one solution branch to another one can also be initiated in an input-driven manner under certain conditions. Consider two solution branches $\mathbf{f}_1(\mathbf{x})$ and $\mathbf{f}_2(\mathbf{x})$ defined on the same domain of inputs $X_1$ and $X_2$ with $X_1 = X_2 = X$. Then, according to the robustness to perturbation property stated above, the output feedback dynamics will reside in one of the branches for all inputs from the domain $X$. That is, the ambiguity resolution scheme is cyclic: Driving the network with a cyclic trajectory $\mathbf{x}(k)$ in input space for $k = 1, \ldots, K$ and with $\mathbf{x}(1) = \mathbf{x}(K)$, the same ambiguity resolution scheme is applied at time step $k = 1$ and $k = K$. In other words, if $\hat{\mathbf{y}}(\mathbf{x}(1)) \in Y_b$ then also $\hat{\mathbf{y}}(\mathbf{x}(K)) \in Y_b$. A transition between solution branches can only be initiated by forcing the outputs to the other basin of attraction.

However, if $X_2$ is a subset of $X_1$, i.e. $X_2 \subset X_1$, then there exist paths $\mathbf{x}(k)$ in input space for $k = 1, \ldots, K$ and with $\mathbf{x}(1) = \mathbf{x}(K)$ such that $\hat{\mathbf{y}}(\mathbf{x}(1)) \in Y_2$ and $\hat{\mathbf{y}}(\mathbf{x}(K)) \in Y_1$. That means the ambiguity resolution scheme is not cyclic and the transition between solutions can be initiated in an input-driven manner. Even necessary criteria for the input sequence $\mathbf{x}(k)$ to result in a switching of the ambiguity resolution scheme can be defined: If robustness to inputs and outputs according to Sec. 7.2.1 and Sec. 7.2.2 apply, then $\hat{\mathbf{y}}(\mathbf{x}(K)) \in Y_1$ can only be achieved if the input sequence $\mathbf{x}(k)$ leaves the domain $X_2$ for some $\mathbf{x}(k)$. The input-driven transition from solution branch $Y_2$ to $Y_1$ is illustrated in Fig. 7.5 (right).

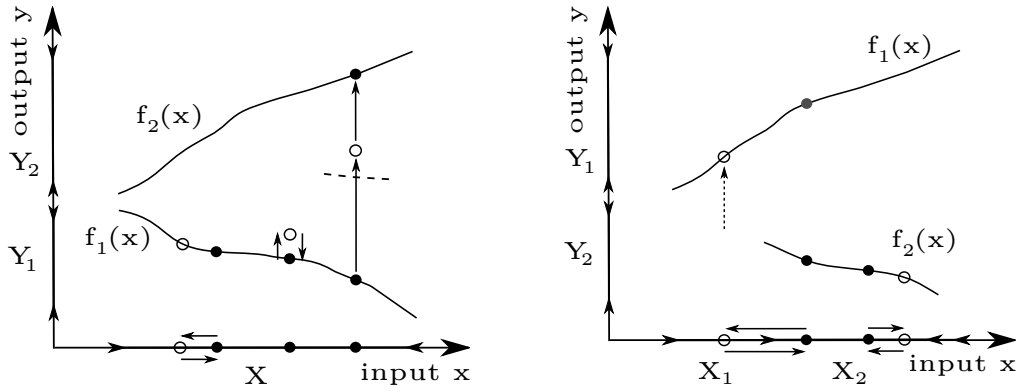



Fig. 7.5: Robustness properties of input-driven multi-stable output feedback dynamics (left): For small changes of the input, the output feedback dynamics shall stay in the same solution branch. Small perturbations of the output shall not result in switching the solution branch. Output feedback-driven transition of solution branch shall occur only for larger perturbations. Input-driven transition between solution branches (right): Driving the inputs out of the domain of a solution branch and back again results in a non-cyclic output sequence. The output remains in the same solution branch if the input sequence stays in the domain of this branch and the output feedback dynamics are robust against changes of the input.

Otherwise, the output feedback dynamics remain in the same solution attractor: The ambiguity resolution scheme is cyclic, i.e. $\hat{\mathbf{y}}(\mathbf{x}(K)) \in Y_2$, if the input sequence $\mathbf{x}(k)$ does not leave the domain $X_2$ and the criterion from Sec. 7.2.1 and Sec. 7.2.2 apply. This is illustrated by the second example input in Fig. 7.5 (right).

The input-driven transition property applies more generally if $X_1 \cap X_2 \neq \emptyset$ and $(X_1 \cap X_2) \subset X_1$ as well as $(X_1 \cap X_2) \subset X_2$. This scheme can be analogously generalized to multiple solution branches with respective input domains. The selected solution depends then on two factors: The boundaries of overlapping domains and the closeness of solution branches in output space. That is, if the input drives the network out of the domain of the current solution branch into a region of two solution branches for instance, the selected solution branch depends on the distances of the current outputs to both solution branches.

Moreover, the kind of switching between solution branches can be classified. Following the terminology introduced in Sec. 2.3.6, we have a flip node transition from one solution branch to the other if the solution branches in the output space do not merge smoothly into each other. Then, the output dynamics quickly approach the other solution branch as will be shown in Sec. 7.3. For smoothly merging solution branches, there is no particularly observable transition between both solution branches when driving inputs from a region with a single solution branch into a region with smoothly forking solution branches, the current estimated output and also external perturbations of the outputs decide which branch is followed after the fork.

### 7.2.3 Attractor-based short-term memory

I combine the properties described in Sec. 7.2.1 and Sec. 7.2.2 into a formulation of *attractor-based short-term memory*. Output feedback dynamically binds inputs to one of possibly many outputs. If the output feedback dynamics reside in a particular input-parameterized attractor, i.e. solution branch, for perturbations of inputs and outputs according to the criteria described in Sec. 7.2.1 and Sec. 7.2.2, this binding of outputs to inputs can be understood as an attractor-based memory: The ambiguity resolution scheme is memorized over time as long as the inputs do not leave the domain of a solution branch or output perturbations force the dynamics to select another solution branch. The memory is *attractor-based* because solution branches are modeled by multi-stable attractor dynamics. The memory is *short-term* because association

of inputs to solution branches is based on the network activity, not the network connectivity. The latter refers to knowledge that is invariant to the current system state. See [160, 161] for an analogous notion of short-term and long-term memory. A similar notion of attractor-based short-term memory has been described in [150], [31] (p. 25) and [162] (p. 1219).

There are also *transient-based* short-term memories implemented by transient system states. This is the typical case in transient-based computation with reservoir networks [144, 87]. Transient- and attractor-based short-term memories have different properties: While transients are temporally elusive, i.e. appear only in a confined time span, attractor states are invariant over time (as long as they are not perturbed too much). Sec. 7.5 compares the transient- and attractor-based approaches to short-term memory in reservoir and AELM networks with each other.

#### Multi-stable output feedback dynamics and output feedback stability

We finally discuss how multi-stable output feedback dynamics relate to the definition of output feedback stability in Sec. 3.4.2. Due to the input-output specific formulation of output feedback stability, the definition in Sec. 3.4.2 also covers the case of multi-stable output feedback dynamics to some extent:

Consider a teacher-forced network state sequence $\mathbf{h}(\mathbf{x}(k), \mathbf{y}(k))$ for $k = -\infty, \ldots, 0$, where $\mathbf{y}(k)$ belongs to a particular solution branch $Y_1$. Then, releasing outputs from teacher-forcing yields an output feedback-driven state sequence $\mathbf{h}(\mathbf{x}(k), \hat{\mathbf{y}}(k))$. The system is output feedback stable if $||\mathbf{h}(\mathbf{x}(k), \mathbf{y}(k)) - \mathbf{h}(\mathbf{x}(k), \hat{\mathbf{y}}(k))|| < \epsilon$ for $k = 1, \ldots, K$ and a small $\epsilon > 0$. This is fulfilled if the network was trained to have at least a single stable attractor of the output feedback dynamics (conditions from Sec. 7.2.1 and Sec. 7.2.2 are fulfilled) that matches the solution branch in the target sequence, i.e. $\hat{\mathbf{y}}(k) \in Y_1$, and $\mathbf{x}(k)$ does not leave the domain of solution branch $Y_1$.

However, the definition of output feedback stability has to be restricted for the case of switching solutions in the target sequence $\mathbf{y}(k)$. It depends on many factors which solution is selected by the output feedback-driven network (see discussion in Sec. 7.2). A match between the ambiguity resolution in the target sequence $\mathbf{y}(k)$ and the estimated output sequence $\hat{\mathbf{y}}(k)$ can generally not be expected for arbitrarily switch solutions of the target sequence $\mathbf{y}(k)$. Different ambiguity resolution schemes imply large differences $||\mathbf{h}(\mathbf{x}(k), \mathbf{y}(k)) - \mathbf{h}(\mathbf{x}(k), \hat{\mathbf{y}}(k))|| > \epsilon$ if $\mathbf{y}(k) \in Y_1$ and $\mathbf{y}(k) \notin Y_1$. Then, the network is not output feedback stable, although it might fulfill the more general properties outlined in Sec. 7.2.1 and Sec. 7.2.2.

## 7.3 Forward and inverse kinematics of a planar arm revisited

The previous examples in Sec. 7.1 and Sec. 7.1.4 showed the principles of representing multiple solutions in a network using output feedback connections and Algorithm 7.6. In this section, the kinematics of the planar robot arm are revisited to evaluate the impact of Algorithm 7.6 and its parameters systematically. Moreover, the desired properties of the output feedback dynamics stated in Sec. 7.2 are shown to be implemented by the proposed method. In particular, it is shown that output feedback dynamics allow smooth generalization along a solution branch, i.e. the network is robust to perturbations of the input also in areas where no training samples are presented.

### 7.3.1 Experiment setup

Although the planar robot arm has only two degrees of freedom and also two degrees of freedom are controlled, i.e. the end effector position in the plane, the inverse kinematics have two distinct solutions: a “lefty” elbow and a “righty” elbow solution (compare Fig. 7.6). Data is generated

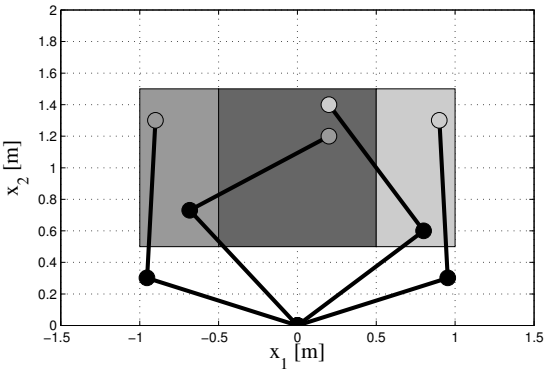



Fig. 7.6: Inverse kinematics with two solutions in overlapping rectangles: Lefty solution in the left rectangle and righty solution in the right rectangle. In the overlapping region (dark gray), both solutions are present in the training data.

with both solutions in overlapping rectangles (see Fig. 7.6) with

$$\begin{aligned}\mathbf{x} \in & X_1 = \{(x_1, x_2) | x_1 \geq -1 \wedge x_1 \leq 0.5 \text{ and } x_2 \geq 0.5 \wedge x_2 \leq 1.5\} \quad \text{and} \\ \mathbf{x} \in & X_2 = \{(x_1, x_2) | x_1 \geq 0.5 \wedge x_1 \leq 1 \text{ and } x_2 \geq 0.5 \wedge x_2 \leq 1.5\}\end{aligned}$$

on equally spaced grids with $15 \times 10$ vertices using the inverse kinematics described in Appendix A.4. The lefty solution is applied throughout $X_1$, and the righty solution in $X_2$.

Properties of trained networks are evaluated for 20 independent trials. The data is randomly split into training and test set per trial with 75% of the data for training and 25% for testing. Each trial is conducted with a freshly initialized network that comprises 100 hidden neurons with *tanh* activation functions (3.5). The weights $\mathbf{W}^{inp}$ and $\mathbf{W}^{fdb}$ are initialized uniformly in $[-0.75, 0.75]$ and $[-0.5, 0.5]$, respectively. Biases $\mathbf{b}$ of the hidden neurons are initialized in $[-1, 1]$. In each trial, performance measures are evaluated depending on the parameters of Algorithm 7.6, i.e. the network is retrained with different numbers of sequence steps $K = 1, \ldots, 5$ and noise scales $l = 0.1, \ldots, 0.5$. Prior to read-out learning according to Algorithm 7.6, contributions of the inputs and outputs to the hidden layer are balanced and regularized on the training data (without synthesized sequences) using $g^{inp} = g^{fdb} = 0.5$ and $\beta^{inp} = \beta^{fdb} = 0.01$ in (5.9) and (5.11), respectively. Read-out learning is conducted according to Algorithm 7.6 and Algorithm 5.4, i.e. the network states and outputs are harvested for the synthesized sequences and then the optimal regularization parameter $\alpha^{out}$ is determined by a line search. Training of the backward model proceeds analogously: The network is driven by the synthesized input sequences and the network states are harvested. Then, the optimal regularization parameter $\alpha^{rec}$ is determined by a line search. Note that optimization of the read-out regularization parameters $\alpha^{out}$ and $\alpha^{rec}$ is accomplished on the training set which is meaningful due to the output feedback stability problem (see discussion in Sec. 5.3).

### 7.3.2 Role of parameters $K$ and $l$ in Algorithm 7.6

In the following, we evaluate the task-specific performance, number of steps required for convergence and the robustness of solution branches depending on the parameters $K$ and $l$ in Algorithm 7.6. The role of these parameters and their effect on the properties stated in Sec. 7.2.1 and Sec. 7.2.2 are discussed.

#### Robustness of the task-specific performance

First, the task specific errors are considered depending on parameters $K$ and $l$ in Algorithm 7.6. Errors of the forward kinematics are calculated by (6.1) and (6.2).

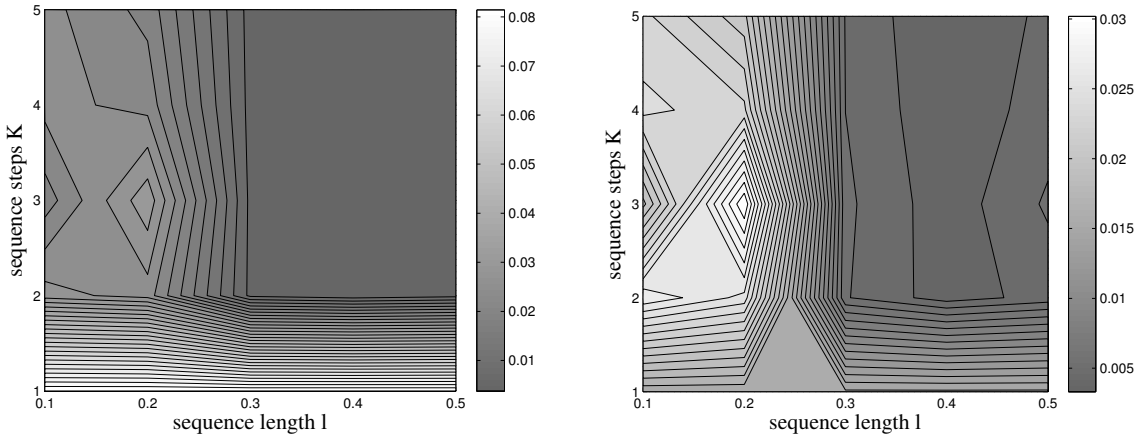



Fig. 7.7: Mean error in task space (meters) depending on $l$ and $K$. Forward kinematics (left) and inverse kinematics (right).

Fig. 7.7 shows test errors in meters for the forward (left) and inverse kinematics (right). Note that the inverse model comprises two different solutions to the inverse kinematics, which is not considered by this error measure. The actual solution selection mechanism and the robustness of both solutions with respect to perturbations of the input is discussed in more detail later on. For sequences with a single step ($K = 1$), errors for the forward and inverse model are rather high. Similarly, rather small sequence lengths $l < 0.3$ yield to inferior performance. Single step sequences imprint only fixed-point conditions into the output feedback dynamics and are not sufficient to program the spatially extended attractor basins. However, choosing $l \geq 0.3$ and $K \geq 2$ yields robust results. Sufficiently long and spatially extended sequences are necessary to shape multi-stable output feedback dynamics.

### Speed of output feedback convergence

Sequence length $l$ and the number of steps per sequence $K$ affect the speed of network convergence in a coupled manner. Intuitively, increasing the sequence length $l$ and keeping the number of steps constant yields to an increased speed or step width. Note that this step width directly shapes the output feedback dynamics according to Algorithm 7.6. Thus, one can expect increased speed of convergence in this case which depends on both sequence length and

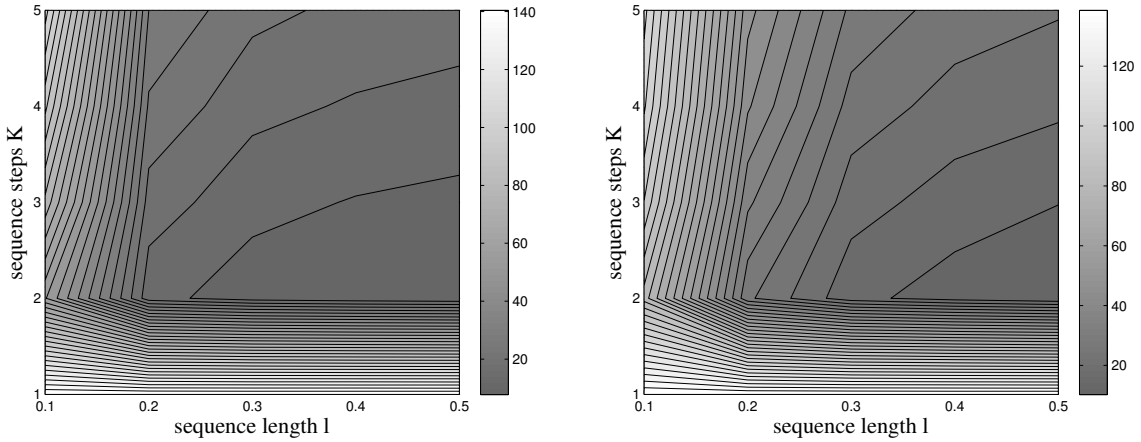



Fig. 7.8: Average number of iterations until the output feedback dynamics converge as function of $l$ and $K$. Forward kinematics (left) and inverse kinematics (right).

number of steps. Fig. 7.8 illustrates this relation by showing the average number of iterations until the network state converges depending on $l$ and $K$. Note that in case of a single sequence step ($K = 1$), increasing $l$ does not have the expected effect because then only the fixed-point condition is programmed, i.e. there is no sequence and hence the parameter $l$ has no effect. For $K \geq 2$, the expected dependency of convergence speed and sequence length $l$ can be observed. Note that less iterations until convergence decrease the computational load for network exploitation. The iteration speed smoothly varies for many pairs of $l$ and $K$ which shows a robust behavior of Algorithm 7.6.

**Robustness of solution branches**

In the context of learning and resolving multiple solutions of a mapping, it is particularly important to fulfill the robustness criterion stated in Sec. 7.2.1. The output feedback dynamics shall not switch solutions arbitrarily for small changes of the input. A solution should be switched only by forcing the network at the output to change the redundancy solution scheme or by feeding the network with a sequence of inputs that leaves the domain of the current solution branch.

Starting from a teacher-forced solution, e.g. $\mathbf{y} \in Y_1$, I query joint angles for all target positions $\mathbf{x}$ randomly chosen from a grid in the respective input domain, e.g. $\mathbf{x} \in X_1$. The estimated outputs $\hat{\mathbf{y}}(\mathbf{x})$ are then classified into lefty or righty solution based on the value of the second joint angle $\hat{y}_2$ (see Appendix A.4 for more details). More formally, let

$$id_{Y_1}(\hat{\mathbf{y}}(\mathbf{x})) = \begin{cases} 1 & \text{if} \quad \hat{\mathbf{y}}(\mathbf{x}) \in Y_1 \\ 0 & else \end{cases}$$

be an indicator function that returns unity if the estimated output is element of the respective solution branch $Y_1$. I calculate the average of the correct solution branch over all input targets $\mathbf{x}_n$ with $n = 1, \ldots, N$ depending on the training parameters $K$ and $l$ by

$$Y_1(l, K) = \frac{1}{N} \sum_{n=1}^{N} id_{Y_1}(\hat{\mathbf{y}}(\mathbf{x}_n)) \qquad \forall\, \mathbf{x}_n \in X_1.$$

The results for all queried targets and both solution branches are combined in a single measure that reflects the robustness of the trained solutions to perturbations of the input depending on the parameters of Algorithm 7.6:

$$R(l, K) = \frac{1}{2} \left( Y_1(l, K) + Y_2(l, K) \right) \tag{7.4}$$

where for $Y_1(l, K)$ the network was initially forced to the lefty and for $Y_2(l, K)$ to the righty solution.

Fig. 7.9 (left) shows the results for the measure (7.4). For $K = 1$ or small sequence lengths $l$, solutions are switched rather frequently. Only ten to twenty percent of the estimated outputs have the same redundancy resolution scheme as in the initial teacher-forced condition for these parameter configurations. This confirms the observation of flipping solution branches in Sec. 7.1.1: For training fixed-point conditions only ($K = 1$ or $l = 0$), training samples may be fixed-points of the output feedback dynamics but their basin of attraction is rather small. Then, driving the network with several inputs easily causes a switch of the solution branch. For $K \geq 2$ and $l \geq 0.2$, however, more than ninety percent of the queried targets stay within the initially teacher-forced redundancy resolution scheme. In this range of parameters of Algorithm 7.6, Fig. 7.9 (left) indicates a stable redundancy resolution scheme which allows to query outputs smoothly along a solution branch.

The robustness of solution branches to perturbations of the input is shown spatially for $K = 3$ and $l = 0.4$ in Fig. 7.9 (right). There is only very small variance of the upper solution

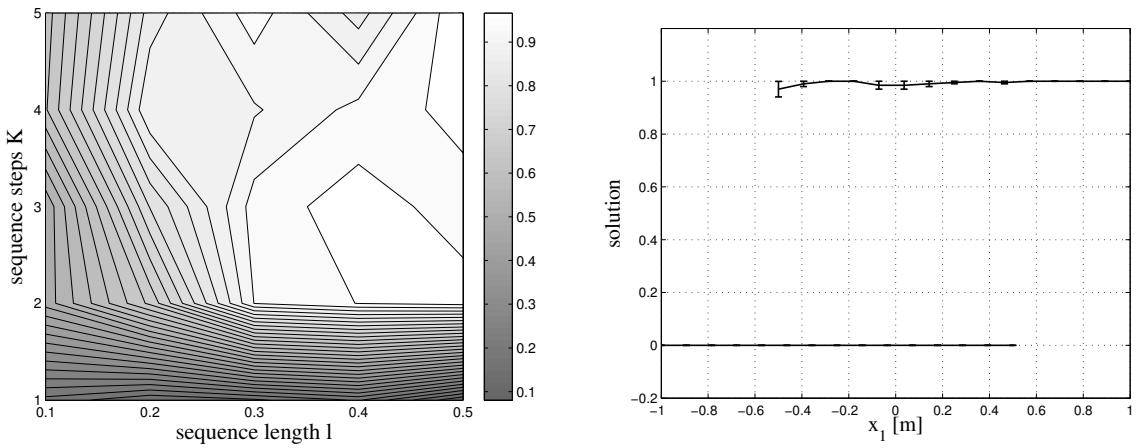



Fig. 7.9: Robustness of solution branches against perturbations of the input as function of the parameters $l$ and $K$ (left). Robustness of each solution branch shown spatially for $K = 3$ and $l = 0.4$ (right), where results are averaged over the second task dimension $^{-1}_{2}$ and all networks.

branch in Fig. 7.9 (right) which confirms that most of the trained networks stay in the solution branch they were initially forced to. It is worth mentioning that solution branches are "holey" and unbalanced in training data due to the random selection of training samples. The fact that solution branches are nevertheless robustly represented in the networks shows that generalization along solution branches is possible also if parts of the manifolds are missing in the training data.

### 7.3.3 Hysteresis of bi-stable output feedback dynamics

Finally, I show that the attractor-based solution selection can result in hysteresis effects during network exploitation. Due to the only partially overlapping regions for each solution in task space (compare Fig. 7.6), the hysteresis effect can be easily shown by feeding the network with target trajectories that cross the border of the three areas in Fig. 7.6 from different directions. The following results were obtained using one of the trained networks above with $K = 3$ and $l = 0.4$. The network is fed with input trajectories $\mathbf{x}(k)$ and the outputs $\hat{\mathbf{y}}(k)$ are read out immediately. That is, I do not let the output feedback dynamics converge.

Fig. 7.10 shows the hysteresis of the output feedback dynamics in joint (left) and task space (right). Target movements from $\mathbf{x} = (-1, 1)^T$ to $\mathbf{x} = (1, 1)^T$ and backward result in different redundancy resolution schemes for the same network input $\mathbf{x}$: Starting with the lefty solution from the left corner in Fig. 7.6, the network remains in the lefty solution until the target position reaches the end of the rectangle $X_1$ with the lefty training data at $x_1 = 0.5$. For $x_1 > 0.5$, only the righty solution was trained and is dominant such that the dynamics switch to the righty solution.

Going on with the reverse movement from right-to-left in Fig. 7.6, the network remains in the righty solution until the target position reaches the left end of the rectangle with the lefty training data only for $x_1 < -0.5$. Note that the previous solution is kept slightly behind the end of the respective rectangles (see Fig. 7.10 (left)) which shows that the network extrapolates the current solution to a certain extent also into the area of the other solution. The hysteresis of the output feedback dynamics in this particular configuration of input and output domains illustrates also the memory of the system: The history of previous states is retained in the attractor dynamics of the output feedback loop during the movement.

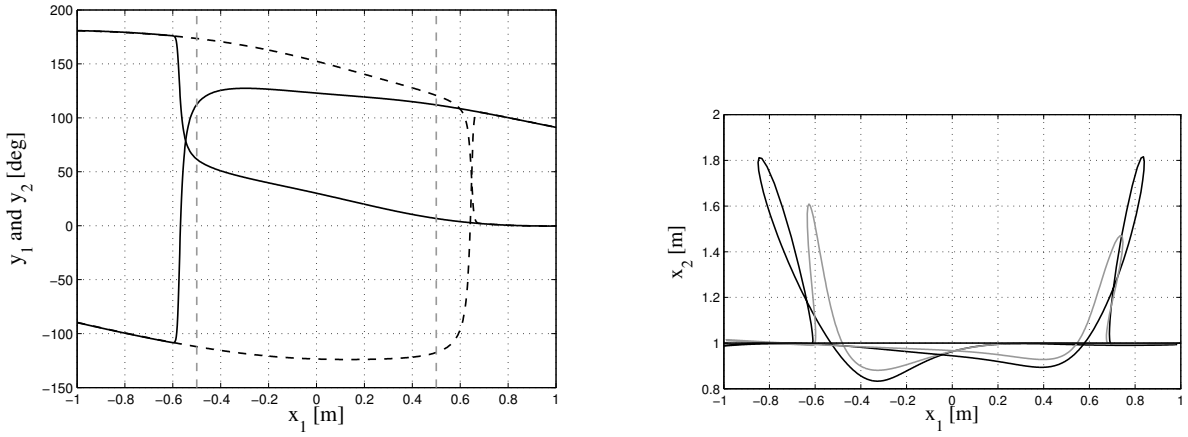



Fig. 7.10: Hysteresis of output feedback dynamics in joint space (left) and task space (right). Linear target movements from $\mathbf{x}(1) = (-1, 1)^T$ to $\mathbf{x}(K) = (1, 1)^T$ and backward result in different redundancy resolution schemes for the same network input $\mathbf{x}$. Left: Joint angle trajectories $(y_1(k), y_2(k))$ are shown for both, forward direction ($k = 1, \ldots, K$, solid lines), and backward direction ($k = K, \ldots, 1$, dashed lines). Right: Black and gray lines are the respective end effector trajectories of the forward and backward movement in task space.

## 7.4 Forward and inverse kinematics of the humanoid robot iCub

In this section, an AELM is trained to learn the forward and inverse kinematics of the right arm of the humanoid robot iCub (shown in Fig. 7.11). I restrict the model to four degrees of freedom of the upper arm and control the end effector position only. First, the data collection and network training procedure is outlined. Then, I statistically evaluate the performance of the trained networks on the forward and inverse kinematics, the combined application of task and joint space constraints, and analyze the selection of solutions to the inverse kinematic mapping depending on the initial condition.

### 7.4.1 Sampling of kinematic data

Data is generated by sampling the four-dimensional joint space according to an equally spaced grid with seven steps per dimension yielding 2401 data points. I calculate the forward kinematics for each sample and remove samples with self-collision joint configurations. I finally obtain $N = 2202$ samples, which are randomly split into training and test set per network initialization. Hereby, I assure that samples with ambiguous solutions to the inverse kinematic mapping are included in the training set for consistent evaluation.

To obtain normalized inputs for the network, I preprocess joint angles and task space coordinates by linearly transforming them to the range $[-1, 1]$. Joint angles are transformed according to the robot's joint limits and task space coordinates according to their minimal and maximal values.

### 7.4.2 Network initialization and training

The network is set up with three neurons $\mathbf{x}$ for the end effector position and four neurons $\mathbf{y}$ which encode the shoulder and upper arm joint angles. I use networks with $R = 300$ neurons with Fermi-activation functions (3.4) in the hidden layer. The input weights $\mathbf{W}^{inp}$ and $\mathbf{W}^{fdb}$ are initialized uniformly in $[-2, 2]$ and $[-1, 1]$, respectively. The biases $\mathbf{b}$ of the activation functions are drawn uniformly from $[-1, 1]$.

To balance contributions of task space and joint angle inputs to the hidden representation and to reduce the potential gain of the output feedback loop, I recompute the weights $\mathbf{W}^{inp}$

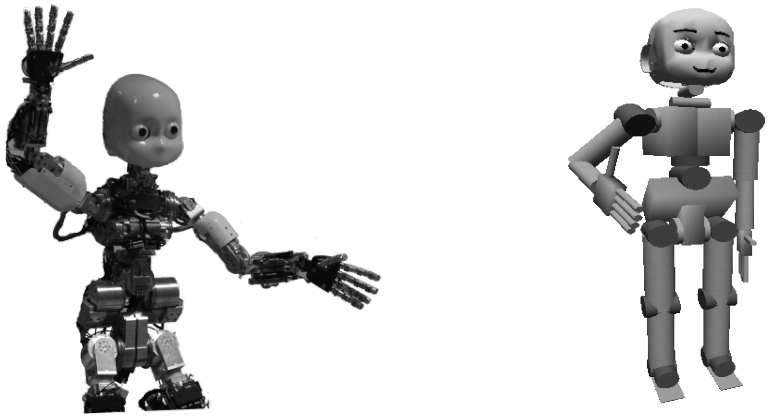


Fig. 7.11: The iCub robot [163]: Real robot (left) and robot simulation [164] (right).

and $\mathbf{W}^{fdb}$ with smaller values according to (5.9) and (5.11) on the training data. I set $\beta^{inp} = \beta^{fdb} = 0.1$ and $g^{inp} = g^{fdb} = 0.5$ in (5.9) and (5.11) throughout the experiments.

Learning of the read-out weights $\mathbf{W}^{out}$ and $\mathbf{W}^{rec}$ is accomplished according to Algorithm 7.6 with regularization parameters $\alpha^{out} = \alpha^{rec} = 10^{-3}$ and perturbations $\nu$ of length $l = 0.6$ and $K = 3$. I train 100 networks in a cross-validation manner on randomly selected subsets of the data comprising 75% of its samples.

### 7.4.3 Forward and inverse kinematics

I evaluate forward and inverse kinematics in task space according to (6.1) and (6.2), respectively, and present errors in meters. To account for the different solutions to the inverse kinematics in the data, the network is teacher-forced to the input-output pair from the data first, i.e. driving the network to state $\mathbf{h}(\mathbf{x}, \mathbf{y})$. Then, the output feedback loop is iterated until convergence.

Tab. 7.2 displays errors with standard deviations of the forward and inverse kinematic models averaged over 100 network initializations. Note that the training data covers a cube of approximately $58 \times 56 \times 61 cm$ in task space. Tab. 7.2 confirms previous results for learning one solution only [98, 24] and underlines the generalization capabilities of the distributed representation in neural networks. Typically, the forward kinematics are approximated more accurately than the inverse mapping which reflects the different difficulties of both mappings.

### 7.4.4 Dynamical selection of solutions to the inverse kinematics

I evaluate the dynamic solution selection mechanism of the trained networks for a bi-stable attractor corresponding to an elbow up and elbow down configuration of the arm that is contained in the training set. Network-produced solutions for both joint configurations are shown in Fig. 7.12 (a) and (b). For a systematic evaluation of the dynamical solution selection mechanism by the output feedback dynamics, I generate initial conditions $\mathbf{y}(\alpha)$ on the connecting line between both solutions $\mathbf{y}_1$ and $\mathbf{y}_2$ in joint space according to $\mathbf{y}(\alpha) = \mathbf{y}_1 + \alpha(\mathbf{y}_2 - \mathbf{y}_1)$. Then, the networks run output feedback-driven until convergence where only the desired end effector position $\mathbf{x}$ is kept constant at the inputs.

Tab. 7.2: Mean error of the forward and inverse kinematic mapping in meters averaged over 100 network initializations.

| | **Fwd. Kin.** *FK* $[m]$ | **Inv. Kin.** *IK* $[m]$ |
|---|---|---|
| **Train** | $0.0062 \pm 2.38 \cdot 10^{-4}$ | $0.0068 \pm 3.37 \cdot 10^{-4}$ |
| **Test** | $0.0067 \pm 2.93 \cdot 10^{-4}$ | $0.0072 \pm 4.08 \cdot 10^{-4}$ |

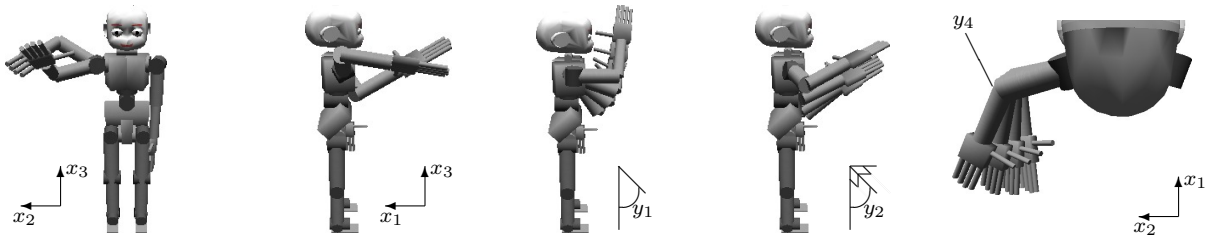



Fig. 7.12: Elbow up and elbow down solutions to the inverse kinematics reproduced by a trained network (frontal view (a) and lateral view (b)). Mixed task and joint space control ((c) – (e)). Exemplary postures when constraining $x_1$, $x_2$ and $y_1$ (c), $x_1$, $x_2$ and $y_2$ (d), and $x_1$, $x_3$ as well as $y_4$ (e).

Fig. 7.13 shows the selection of solution $\mathbf{y}_1$ or $\mathbf{y}_2$ depending on $\alpha$ averaged over all trained networks. Both solutions have a basin of attraction and are fixed-points of the network dynamics. The border of attractor basins is rather clearcut on average for $\alpha \approx 0.5$. The steep slope in Fig. 7.13 confirms a statistically consistent behavior of the networks: The geometrically closest solution in joint space is approached which indicates evenly spread basins of attraction implemented by Algorithm 7.6.

### 7.4.5 Mixed task and joint space control

Finally, I demonstrate the combination of task and joint space constraints through associative completion according to (3.11). The remaining, unconstrained components run in the output feedback loop. I evaluate the mixed constraint satisfaction performance for three scenarios: Fig. 7.12 (c) shows exemplary postures obtained for constraining $x_1$, $x_2$ and $y_1$, which means that the end effector should reside on the vertical line through $(x_1, x_2)$ while $y_1$ controls the anteversion of the upper arm. In Fig. 7.12 (d) $x_1$, $x_2$ and $y_2$ are constrained, i.e. the end effector shall be positioned on the vertical line through $(x_1, x_2)$ while $y_4$ controls the abduction of the elbow. Finally, in Fig. 7.12 (e) $x_1$, $x_3$ and $y_4$ are constrained, i.e. the end effector shall be positioned on the horizontal line through $(x_1, x_3)$ while $y_4$ controls the flexion of the elbow.

I evaluate the accuracy of the constraint satisfaction for all 100 networks systematically. The rows in Fig. 7.14 display the error in task (left column) and joint space (right column) for the three scenarios illustrated in Fig. 7.12 (c) – (e). Errors are evaluated with respect to the constrained variables, i.e. $E^{task}_{x_1x_2} = ||(x_1, x_2) - FK(\hat{IK}(x_1, x_2, y_1))_{x_1,x_2}||$ is the error in task space evaluated for the constrained dimensions $x_1$ and $x_2$. The networks comply with the constraints in the range of the overall accuracy of the trained models. Application of mixed task and joint space constraints emphasizes the flexibility of the associative network.

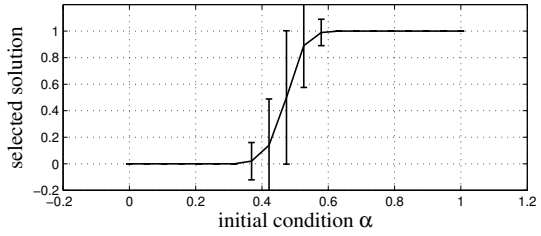



Fig. 7.13: Dynamical selection of a solution depending on the initial condition parameterized by $\alpha$ (see text). Results averaged over 100 networks.

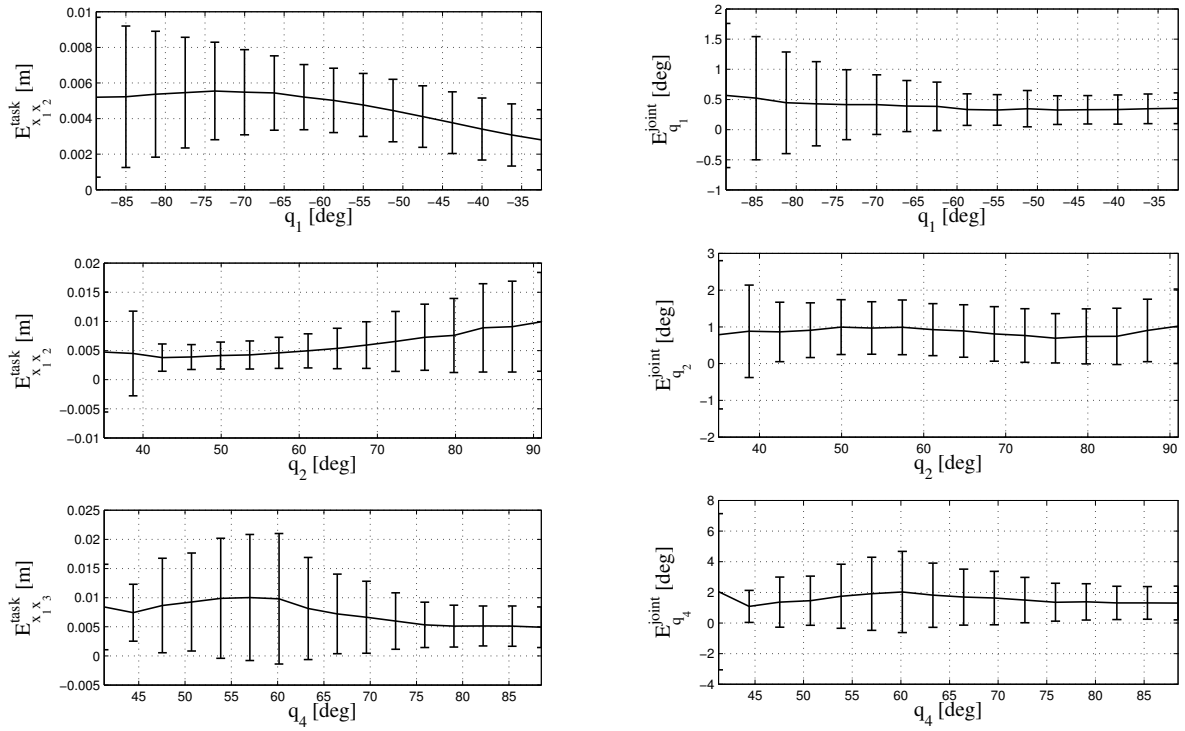



Fig. 7.14: Accuracy of the mixed constraint satisfaction in task (left column) and joint space (right column) averaged over all network initializations. Constrained variables in rows: $x_1$, $x_2$ and $y_1$; $x_1$, $x_2$ and $y_2$; $x_1$, $x_3$ and $y_4$.

## 7.5 Transient- and attractor-based short-term memory

Originally, reservoir networks were introduced for temporal pattern processing like time-series transduction, classification and generation where the explicit focus resides on the transient-based short-term memory implemented by the dynamical reservoir [144, 30]. Due to the Echo State Property (see Sec. 3.2.3), which implies global stability of the reservoir, these transients are rather restricted in their temporal length and thus is memory. Local stability of the output feedback dynamics provides a more sustained and resistant type of short-term memory based on attractor states. In this section, the concept of attractor-based short-term memory presented in Sec. 7.2.3 is demonstrated in an inverse graphics example and compared to the traditional transient-based short-term memory approach in Echo State Networks (ESNs).

### 7.5.1 Lissajous figures that virtually rotate

A sequence of Lissajous figures provides an illustrative example for resolution of ambiguity based on the input history. Lissajous figures [165, 166] have been discovered by Nathaniel Bowditch in 1815 who studied movements of coupled pendulums. The curves such a pendulum draws are better known as Lissajous figures due to the French physicist Jules Antoine Lissajous (1822–1880). The (simplified) parametric form of Lissajous figures is

$$x = acos(\varphi t + \theta) \tag{7.5}$$

$$y = bcos(\omega t), \tag{7.6}$$

where $a$ and $b$ are amplitudes, $\varphi$ and $\omega$ frequencies and $\theta$ the phase of the respective oscillations. I use $\varphi = 2$ and $\omega = 1$ to obtain a figure eight-like pattern with $a = b = 1$. Drawing all points produced by these equations in a plane for sampling over $t$, a single "image" $\mathbf{x}$ as shown in Fig. 7.15 is obtained. I project such Lissajous figures onto $13 \times 13$ pixel images, i.e. $\mathbf{x} \in \mathbb{R}^{169}$.

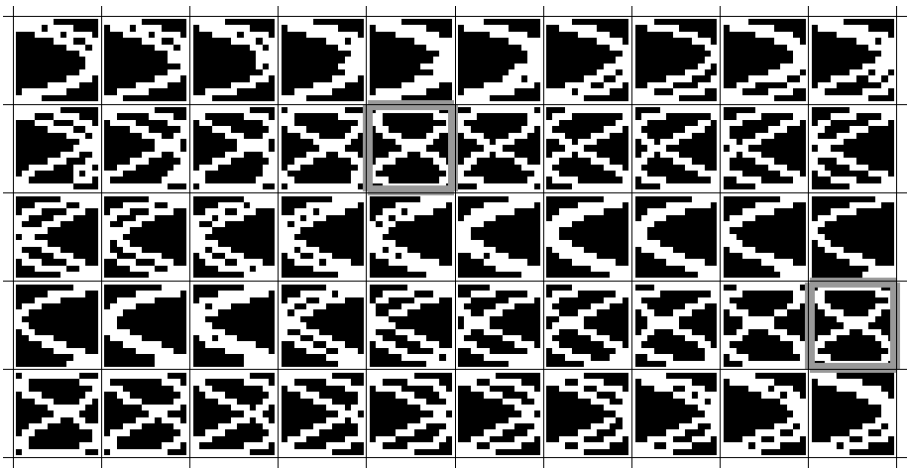


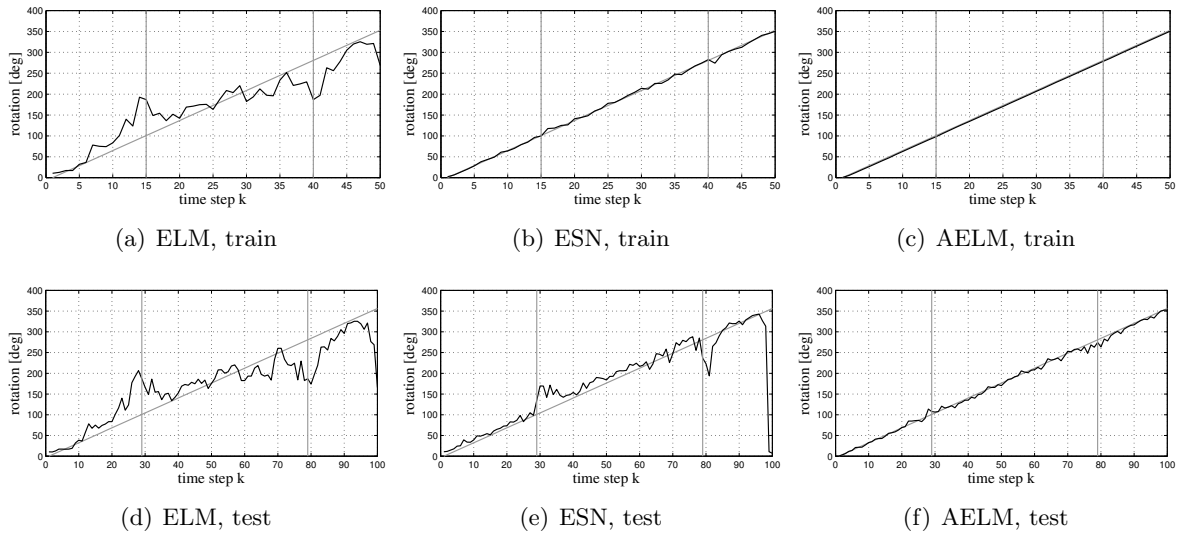
Fig. 7.15: Lissajous figures. The generating variable $\theta$ results in equal stimuli at $k = 15$ and $k = 40$ (marked images). The "inverse graphic" mapping that estimates $\theta$ from the images above is therefore ambiguous.

Following the images along the lines in Fig. 7.15 provokes the impression of a rotation of the Lissajous figure. Virtual rotation is induced by shifting the phase $\theta$ in (7.5) at successive time steps $k$. I sample $\theta$ in small steps, i.e. $\mathbf{x}(k) = \mathbf{x}(\varphi, \omega(\theta(k)), \theta(k))$ with $\theta(k) = \frac{2\pi}{50}k$ and $k = 0, \ldots, 49$. The length of the input sequence $\mathbf{x}(k)$ is denoted by $K$. Note that there are two identical images in the sequence at $k = 15$ and $k = 40$ (see marked images in Fig. 7.15). This means that there is the same stimulus which is caused by different angles $\theta(15) = 100$ and $\theta(40) = 280$ degrees, i.e. the inverse model from Lissajous figures $\mathbf{x}$ to rotation angles $\theta$ is ambiguous. All other images in the sequence can be uniquely mapped to the angular variable $\theta$ which I explicitly enforce by introducing an additional oscillation in $\omega(\theta) = \omega + \frac{1}{10}(sin(4\theta) + 1)$.

### 7.5.2 Disentangling ambiguous rotation angles

The task is to estimate the rotation angle $\theta(k)$ from input images $\mathbf{x}(k)$, i.e. to learn the inverse model of the data generation process given by (7.5)–(7.6). The ambiguity at $k = 15$ and $k = 40$ can be resolved by utilizing the temporal context: At previous and succeeding time steps, inputs $\mathbf{x}(k)$ uniquely map to angular variables $\theta(k)$. Integration of the temporal context also uniquely determines the correct angle at the time steps $k = 15$ and $k = 40$ with ambiguous stimuli. This temporal context can be represented by transients in the network state trajectory or by dynamical attractor selection depending on initial conditions, i.e. multi-stable output feedback dynamics. Depending on the mechanism that implements the memory functionality, I expect different performances of the models in particular when considering variations of the input sequence. How is the memory influenced by temporal elongation and contraction of the input sequence, i.e. increasing or decreasing the number of samples per period?

Before we start with this comparison, I present the memoryless baseline performance on this task achieved with an Extreme Learning Machine. More details about the initialization and training of the models can be found in Appendix A.3.4. Training a pure feed-forward model yields pure approximation of the inverse problem. In particular for $k = 15$ and $k = 40$, the trained ELM outputs the same rotation angle averaged over both target angles, i.e. $\theta(15) = 100$ and $\theta(40) = 280$ yields $\hat{\theta}(15) = \hat{\theta}(40) = 190$. Ambiguous samples therefore result in an error of 90 degrees (see Fig. 7.16 (a) and (d)). This is due to the same network state that is calculate in a feed-forward manner from the same input (compare discussion in Sec. 7.1.1). Fig. 7.17 shows the pairwise differences $||\mathbf{h}(i) - \mathbf{h}(j)||$ of the hidden representation for time steps $i \neq j$ and confirms that the network states $\mathbf{h}(15)$ and $\mathbf{h}(40)$ (and thus the input patterns $\mathbf{x}(15)$ and $\mathbf{x}(40)$) are identical (see black dot in upper triangle in Fig. 7.17 (a) and (d)).




(a) ELM, train (b) ESN, train (c) AELM, train

(d) ELM, test (e) ESN, test (f) AELM, test

Fig. 7.16: Target rotation angle $\theta(k)$ (gray lines) and estimated rotation angle $\hat{\theta}(k)$ (black lines) for the training sequence (top row) and test sequence with $K = 100$ (bottom row): Pure feed-forward Extreme Learning Machine (left), Echo State Network with transient-based short-term memory (middle), and Associative Extreme Learning Machine with attractor-based short-term memory (right).

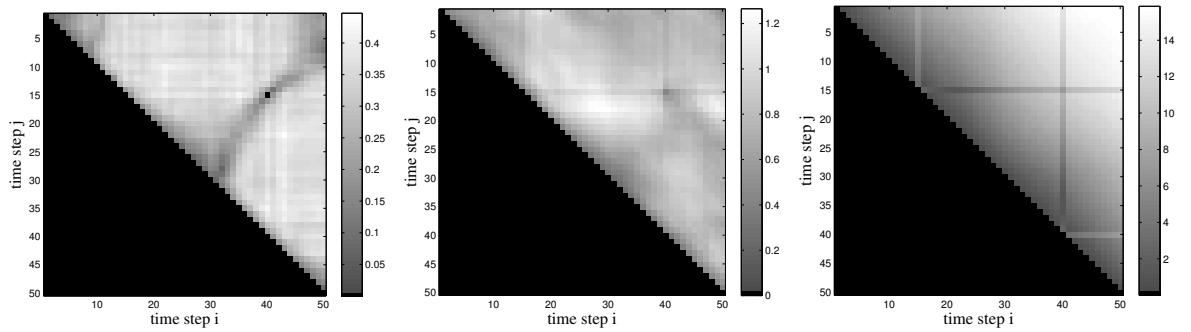
### 7.5.3 Transient- versus attractor-based memory

To resolve the ambiguity in the sequence of Lissajous figures, two different implementations of a short-term memory are considered. The transient-based short-term memory is implemented by a standard ESN without output feedback connections. The network is teacher-forced to wash out initial transients using an entire period of the pattern before training or testing is applied. The ESN operates using transient-based computation.

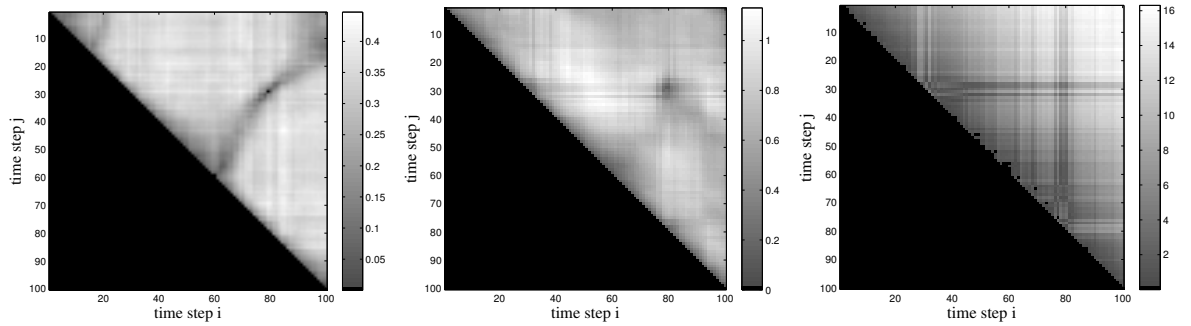
The attractor-based short-term memory is implemented by an AELM trained with Algorithm 7.6 which exhibits no transient dynamics in the hidden layer since $\mathbf{W}^{res} = \mathbf{0}$. However, dynamics are introduced by the output feedback loop in the AELM. Learning shapes bi-stable attractor dynamics of the output feedback loop for the ambiguous training samples. Initialization of the attractor-based memory is simply achieved by driving the network with an input that uniquely maps to an rotation angle. This means that there is no supervised information required to initialize the memory. The AELM operates in an attractor-based computation mode, i.e. the output feedback dynamics are iterated until convergence for each input sample $\mathbf{x}(k)$.

Both networks with transient- and attractor-based memory are capable of estimating the ambiguous rotation angle from the input stimuli in the training sequence accurately (see Fig. 7.16 (b) and (c)). The pairwise distances of the network states on the training sequence are greater than zero at time steps $k = 15$ and $k = 40$ (see Fig. 7.17 (b) and (c)). Therefore, the read-out layer can disentangle the ambiguity and estimate the angle $\theta(k)$ correctly throughout the training sequence. Note, however, that there are distinct reasons for the different states of the ESN and AELM in Fig. 7.17: In the Echo State Network, transient dynamics induced by recurrent connections in the reservoir unfold over time and render states distinguishable at $k = 15$ and $k = 40$. In the AELM, there is no recurrent reservoir, but output feedback of estimated rotation angles at the previous time steps drive the output feedback dynamics into distinct attractors which cause also different hidden states at $k = 15$ and $k = 40$. That is, ambiguous rotation angles for the same input stimulus are disentangled by multi-stable attractor dynamics.

Both approaches to memory result in distinct behavior of the models in test scenarios: If the speed of the rotation is varied, i.e. $K$ in $\theta(k) = \frac{2\pi}{K}k$ is increased, the transient-based memory




(a) ELM, train (b) ESN, train (c) AELM, train




(d) ELM, test (e) ESN, test (f) AELM, test

Fig. 7.17: Pairwise distances $||\mathbf{h}(i) - \mathbf{h}(j)||$ between the hidden states at time steps $i$ and $j$ on the training sequence (top row) and test sequence (bottom row): Pure feed-forward Extreme Learning Machine (left), Echo State Network with transient-based short-term memory (middle), and Associative Extreme Learning Machine with attractor-based short-term memory (right).

implementation fails, as long as not a considerable amount of sequences with different $K$ is used for training (compare Fig. 7.16 (e)). The transients do not generalize to the slower changing inputs and the hidden states become very similar for the ambiguous samples (see enlarged dark area in the upper triangle in Fig. 7.17 (e)). The attractor-based short-term memory, in contrast, retains the ambiguity resolution scheme also over several iteration steps and can therefore resolve the ambiguity robustly on the test sequence (compare Fig. 7.16 (f)). This is also reflected in the large distances between network states in Fig. 7.17 (f).

I evaluate the generalization performance of the networks systematically for a range of sequence lengths $K \in \{10, 20, 30, \ldots, 100, 150, 200, 300, 400, 500\}$. I calculate the mean deviation from the target angle in degrees for each sequence by

$$E = \frac{1}{K} \sum_{k=1}^{K} ||\theta(k) - \hat{\theta}(k)||,$$

where $\hat{\theta}(k)$ is the estimated rotation angle decoded from the coordinate representation in the two output neurons (see Appendix A.3.4 for details).

The generalization performance averaged over 200 independent network initializations is displayed in Fig. 7.18 (left). Echo State Networks and Associative ELMs achieve similar estimation errors on the training set. But for temporally prolonged or shortened image sequences, the Echo State Networks rapidly drop in performance. In the limit of fast input sequences, Echo State Networks perform even worse than the pure feed-forward network (compare dashed lines in Fig. 7.18). The attractor-based short-term memory implemented in the AELM, in contrast, achieves a rather stable performance irrespective of the sequence length (see solid lines in

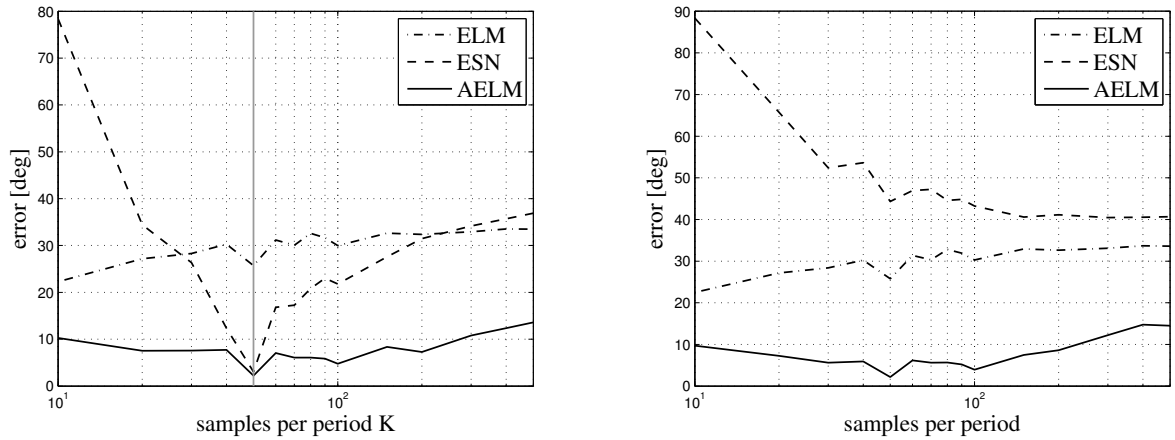



Fig. 7.18: Generalization performance of networks in degree for several sequence lengths (samples per period $K$). Direction of rotation same as in the training set (left) and reverse rotation (right). Training sequence indicated by vertical line (left).

Fig. 7.18). This confirms the observations in [31] that multi-stable output feedback dynamics implement an attractor-based short-term memory that maintains information over longer time spans than pure transient-based short-term memories.

In addition, the attractor-based memory is not sensitive to the order of inputs. That is, the output feedback dynamics can be driven into one of the two attractors, which represent the ambiguous rotation angle, from both forward and reverse rotation directions. The generalization errors for reverse presentation of the input sequences are shown in Fig. 7.18 (right). The transients of the Echo State Network are sensitive to the temporal order of the inputs and thus can only resolve ambiguity in the direction of the training data (see large errors in Fig. 7.18 (right)). The AELM generalizes to reverse presentation of inputs (see Fig. 7.18 (right)) which emphasizes the flexibility of the attractor-based short-term memory.

## 7.6 Concluding remarks

In this chapter, I showed the capability of output feedback dynamics to disentangle ambiguous outputs dynamically. The associative setup resolves the non-convexity problem of learning multi-valued mappings. Solution branches are represented by multi-stable output feedback dynamics where the points of attraction are additionally parameterized by the network input. For robust imprinting of multi-stable dynamics, I introduced an algorithm based on the State Prediction approach presented in Chapter 4. Solution branches are dynamically selected by the network dynamics depending on the current network state. I demonstrated that the associative network model enables the combined satisfaction of constraints in both, the input and output space. Several properties that a dynamical approach to ambiguity resolution shall fulfill were identified. The presented scenarios emphasize the potential of trainable dynamical systems to represent multiple solutions of ambiguous mappings. In particular, there is no supervised information about the number of solution branches necessary: Uni-, multi- and continuous attractor dynamics are covered within a single framework.

Chapter 8

# Movement generation with output feedback dynamics

In Echo State Networks, the standard approach for movement generation is to program cyclic attractors into the output feedback dynamics [34, 86, 167]. This autonomous pattern generation is equivalent to recursive time series prediction (see Sec. 3.3.3). The network dynamics can also be parametrized by additional inputs, e.g. to switch between different gaits [119] or antennae movements for a simulated stick insect in [120], which is conceptually similar to the recurrent neural network with parametric bias (RNNPB, [28, 121, 168, 169]). It has recently been shown that even initially unstable output feedback dynamics can be trained for autonomous pattern generation using the FORCE learning [101] or an unsupervised, reward-modulated Hebbian learning rule [102]. All these approaches target non-stable network configurations, where the output nodes display a trained and desired non-stationary behavior in closed loop operation. In contrast to this work, I make use of transient dynamics for movement generation while the overall network state approaches a fixed-point attractor. Attractor-based movement generation with reservoir networks has been introduced by the author in [67, 24, 55].

## 8.1 Forward and inverse models for movement generation

Dexterous and flexible movement generation is a prerequisite for sophisticated and intelligent robots. Learning a representation of movement primitives that enables generalization and adaptation has been proposed as key concept to cope with new situations [13, 170]. Several computational models that integrate movement primitives as building blocks for movement generation have been proposed, in part relating to findings on the motor system of higher vertebrates. One of the most influential ones has been proposed in [13] assuming that forward and inverse models are needed and provide the building blocks for more complex behavior. The models themselves were realized as feed-forward neural networks in [13] and assumed to be trained by error correction. The general idea has been picked up also in the computational HAMMER model [171], which comprises feed-forward and feedback modules in a control architecture.

A particularly interesting approach for movement generation based on dynamical systems has been proposed under the notion of Dynamic Movement Primitives (DMPs) [170, 172]. Attractor-based dynamical systems are an appealing approach to movement generation, because they intrinsically generalize to new situations, are robust against perturbations, and can be trained to produce a desired behavior [170, 173, 174, 172]. Basically, the DMP approach focuses on movement representation by means of movement primitives in task or joint space that can be combined and modulated to generate new movements. Representing movements in task space makes task-oriented adaptation simple. However, it is not straight forward to translate

movements represented in task space to joint angle trajectories for execution as this requires to solve the highly non-linear and typically redundant inverse kinematics of the robot [175]. A task-based representation lacks the ability to exploit redundancies of a manipulator for task-specific movement optimization. Representation of movements in joint space circumvents this difficulty, but planning and generalization is hardly solvable in a task-related manner without projecting trajectories into task space. Combinations of attractor-based movement generation following the vector integration to endpoint model (VITE, [176]) with Jacobian-based analytic inverse kinematic solvers have been proposed [177, 178], but there is also no deeper connection between solving the inverse kinematics and movement generation.

A recent probabilistic approach for joint optimization of movements in task and posture space was introduced by Toussaint [179]. Planning on a trajectory level is however computationally demanding, requires updates in case of unforeseen perturbations and natural movements are rather achieved by tuning the contribution of constraints than by providing a computational model which exhibits these properties generically. Although the approach in [179] considers task and joint space simultaneously, it does not learn this transformation and assumes that the inverse kinematic problem is solved independently. The SURE_Reach model is capable of integrating mixed constraints in task and joint space by using a hierarchy of representations [180]. In this approach, connections between representation layers are plastic and trained by Hebbian learning. However, the SURE_Reach model suffers from the curse of dimensionality and requires demanding computations between layers.

### 8.1.1 Neural Dynamic Movement Primitives (NDMP)

When DMPs were introduced, it was already pointed out by Ijspeert that recurrent neural networks may provide an alternative implementation for DMPs but "the complexity of training these networks to obtain stable attractor landscapes, however, has prevented a widespread application so far" [181]. In my opinion, this account is still correct with respect to standard recurrent networks [32]. However, it can be revised in view of the efficiently trainable reservoir networks, in particular the echo state approach, which restricts learning to adaptation of output weights by linear methods. I use a particular flavor of reservoir networks, which combine reservoir computing with ideas from associative learning to implement Neural Dynamic Movement Primitives (NDMPs).

NDMPs couple task and joint space by means of a paired forward and inverse model in an associative reservoir network (see Fig. 8.1, [24, 55] and Sec. 3.2 in this thesis). The associative nature of the neural model allows for learning of forward and inverse kinematics in parallel and in a single network. The mapping is bidirectional: The network takes task coordinates as input and joint space coordinates as output or vice versa. These two mappings functionally implement a pair of internal forward and inverse model as described in [13, 182]. The NDMP framework further integrates afferent motor copies for sensory prediction of the input and feedback control. The presented approach avoids explicit model selection during learning and execution, which is typically the main difficulty in architectures of expert modules as proposed in [13]. The selection of a particular inverse model is accomplished dynamically by the attractor-based short-term memory mechanism implemented in the network. While the kinematic mapping between end effector coordinates and joint angles is static in nature, I show that representing this relation within a recurrent neural network enables movement generation. The basic idea is to exploit the network's transient dynamics when approaching an attractor state for smooth and robust movement generation. I start by training the dynamic network with trajectories, e.g. demonstrated by a teacher, or by sampling joint angle configurations and their corresponding end effector position, in order to learn the kinematic mapping. This training makes no explicit reference to the form of the trajectory, which consequently is not stored directly in the network. However, the representation of kinematics in a dynamical system provides a generic movement

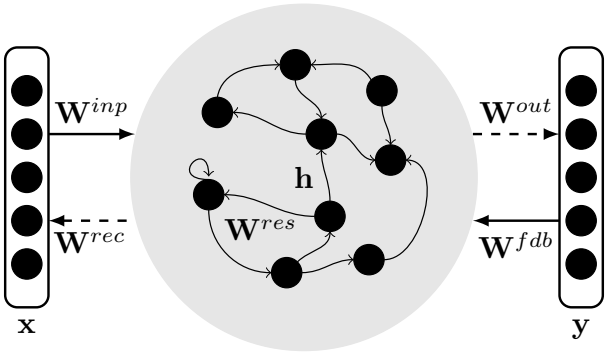



Fig. 8.1: The Associative Reservoir Computing architecture connects task and joint space variables bidirectionally. Thereby the inverse and forward kinematic models can be be queried continuously for generalization.

primitive that is robust to perturbations and exploits ambiguities of the inverse kinematics for flexible movement generation.

Similar to the DMP approach, generalization in NDMPs is achieved by modification of the targeted attractor and using the transient behavior of the network to generate a respective movement. A conceptually similar approach using non-linear spring-damper systems is also proposed to model data of human straight point-to-point reaching movements in the VITE system [176]. Even in a simple version, VITE models kinematic properties of human reaching movements such as bell-shaped velocity profiles. A recent extension in [183] shows that by addition of a non-linear goal-dependent term, which acts as an additional force field, more complex self-touching human movements can be modeled. The strength of the VITE models is their ability to explain the characteristic properties of human movements which is not possible in the DMP framework. However, the VITE models lack a mechanism to learn more complex movements. I demonstrate that the trajectories generated with the NDMP framework have to some degree the kinematic features of human movements despite being learned and lacking explicit control of their geometrical shape. In this respect, NDMP has features of both the DMP and VITE-like approaches. NDMPs differs from those approaches in that DMPs are trained to reproduce movement shape and velocities from sample trajectories in a single space, whereas NDMPs learn static coordinate transformations, i.e. the kinematics, only. The VITE model lacks a mechanism to learn particular movement shapes but models properties of human movements like velocity profiles in task space. NDMPs learn the static kinematic mapping and generate movements with properties of human reaching movements generically. Even though NDMPs are not explicitly trained to reach a target, I show that the generation of straight reaching movement in task space is possible without explicit trajectory training or planning. Contrary to target adaptation in the DMP framework, the presented approach generates additionally the appropriate trajectories in joint space. Flexible movement generation exploiting the manipulator's redundant degrees of freedom without explicit trajectory planning is the main benefit of a coupled representation of task and joint space within a dynamical system.

NDMPs implemented by associative reservoir networks have a strong connection to biological networks: Yamazaki and Tanaka point out strong similarities of the reservoir computing approach to the cerebellum [184], and Mass *et al.* model cortical microcolumns with reservoir networks [185]. I do not aim at precise modeling of brain structures, but the principle structure and functional similarities remain. The cerebellum is involved in motor learning, and it was shown that the cerebellum enables smooth and accurate movements of humans even without feedback (see [186] for a review). I apply the reservoir model in a similar manner to motor learning in robotics and is has been shown that whole body motions can be learned in similar reservoir models with impressive precision [98, 152, 153]. In contrast to [98], where the reservoir

is driven by a smooth target trajectory in task space, I show that movements can be generated by providing a target position only while the network dynamics solve the trajectory formation problem automatically.

### 8.1.2 Attractor-based movement generation

In this section, attractor-based movement generation with associative reservoir networks is introduced. We start with a notational convention to facilitate the further discussion. In the outset of associative methods, there is no dedicated input or output: With respect to the network architecture and the learning algorithm, task space variables $\mathbf{x}$ and joint space variables $\mathbf{y}$ are functionally equivalent (compare Fig. 8.1). For kinematics learning, it is nevertheless convenient to name task space coordinates $\mathbf{x}$ as input and joint angles $\mathbf{y}$ as output, because generally it is required to drive the robot towards targets (inputs) in task space coordinates. With respect to this convention, the inverse kinematic ($IK$) mapping $\mathbf{y} = IK(\mathbf{x})$ is from left to right in Fig. 8.1 (input-to-output) and the forward kinematic ($FK$) mapping $\mathbf{x} = FK(\mathbf{y})$ from right to left (output-to-input).

I exploit the attractor-based representation of the kinematics in the reservoir network for reaching movement generation. Assume that the network has settled to an attractor state associated with an arbitrary position, i.e. $\mathbf{h}(\mathbf{x}, \mathbf{y})$. Assume furthermore that a new target $\mathbf{x}^*$ is selected by a random (or ultimately higher-level) process, e.g. for grasping. To query the corresponding joint angles $\mathbf{y}$, the network input $\mathbf{x}$ is clamped to the new target position $\mathbf{x}^*$. Due to the reservoir and output feedback dynamics, the network needs some iterations until the new attractor state is reached. The idea is to generate end effector movements of the robot by retracing these network transients during convergence to the attractor state. Note that movement generation has not explicitly been trained and all properties of the trajectories are generic to the dynamic network model.

I use a quasi-continuous variant of the network dynamics that slow down the network's convergence to the target attractor state in order to generate smooth trajectories at the output neurons for movement generation. The network update equations (3.3)–(3.7) change to

$$\mathbf{h}(k{+}1) = \sigma((1 - \Delta t)\mathbf{a}(k) + \Delta t\mathbf{a}(k{+}1)) \tag{8.1}$$

$$\mathbf{y}(k{+}1) = (1 - \Delta t)\mathbf{y}(k) + \Delta t\mathbf{W}^{out}\mathbf{h}(k) \tag{8.2}$$

$$\mathbf{x}(k{+}1) = (1 - \Delta t)\mathbf{x}(k) + \Delta t\mathbf{W}^{rec}\mathbf{x}(k) \tag{8.3}$$

where $\Delta t$ parameterizes the network step width per iteration. Parametrization of movement speed by $\Delta t$ during network exploitation raises the question whether a change of $\Delta t$ corrupts the input-output relation learned by the network.

It is straightforward to show that this quasi-continuous variant of the network dynamics does not affect the fixed-point conditions. We obtain from the quasi-continuous version of the network dynamics using the compact notation from (3.8) with

$$\tilde{\mathbf{a}}(k{+}1) = (1 - \Delta t)\tilde{\mathbf{a}}(k) + \Delta t\mathbf{W}^{net}\mathbf{z}(k)$$

that the dependency on $\Delta t$ vanishes for the fixed-point conditions $\tilde{\mathbf{a}}(k{+}1) = \tilde{\mathbf{a}}(k)$:

$$\begin{aligned}
\tilde{\mathbf{a}}(k{+}1) &= (1 - \Delta t)\tilde{\mathbf{a}}(k) + \Delta t\mathbf{W}^{net}\mathbf{z}(k) \\
&= (1 - \Delta t)\tilde{\mathbf{a}}(k{+}1) + \Delta t\mathbf{W}^{net}\mathbf{z}(k) \\
&= \tilde{\mathbf{a}}(k{+}1) - \Delta t\tilde{\mathbf{a}}(k{+}1) + \Delta t\mathbf{W}^{net}\mathbf{z}(k{-}1) \\
\Leftrightarrow \tilde{\mathbf{a}}(k{+}1) - \tilde{\mathbf{a}}(k{+}1) &= -\Delta t\tilde{\mathbf{a}}(k{+}1) + \Delta t\mathbf{W}^{net}\mathbf{z}(k) \\
\Leftrightarrow \Delta t\tilde{\mathbf{a}}(k{+}1) &= \Delta t\mathbf{W}^{net}\mathbf{z}(k) = 0 \\
\Leftrightarrow \tilde{\mathbf{a}}(k{+}1) &= \mathbf{W}^{net}\mathbf{z}(k)
\end{aligned}$$

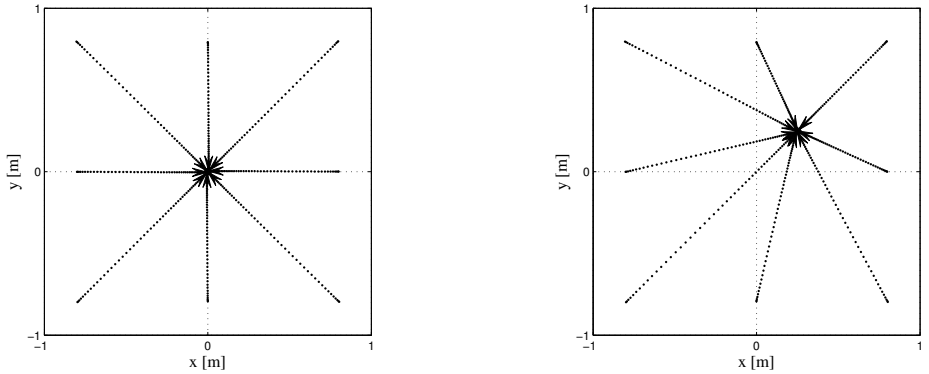



Fig. 8.2: Exemplary movements to different targets in a two-dimensional task space, i.e. $\mathbf{x} = (x, y)$: Target $\mathbf{x}^* = (0, 0)$ (left) and $\mathbf{x}^* = (\frac{1}{4}, \frac{1}{4})$ (right).

where $\tilde{\mathbf{a}}(k) = (\mathbf{x}(k)^T, \mathbf{a}(k)^T, \mathbf{y}(k)^T)^T$ is the combined activity vector before application of the non-linearity. Though this argument applies strictly only if the system is already in a fixed-point, the empiric results in this section show that the network also approaches the same attractor states independent of the chosen $0 < \Delta t \leq 1$.

Fig. 8.2 illustrates the movement generation capabilities of the the attractor-based network. When the network dynamics (3.2), (8.1)–(8.3) are iterated with a fixed target position $\mathbf{x}^*$ at the input, straight reaching trajectories as shown in Fig. 8.2 are generated. Note that the movement speed depends on the relative distance between current position and the target (compare step widths between dots in Fig. 8.2 (left) and (right)) which is a general feature of the presented movement generation scheme and is quantitatively investigated in Sec. 8.2.4.

### 8.1.3 Feedforward-feedback control framework

I integrate the attractor-based movement generation into a control scheme for autonomous reaching and introduce a convergence criterion that measures if the network has settled to an attractor state based on feedback of actual sensory values. Fig. 8.3 illustrates the concept of the proposed approach. A target position $\mathbf{x}^*$ is clamped to the network input and estimated joint angles $\hat{\mathbf{y}}(k)$ are read out. The estimated joint angles $\hat{\mathbf{y}}(k)$ serve as targets for the robot hardware controller. Upon execution, sensory feedback $\mathbf{y}$ about the true robot's joint positions is acquired and fed back to the network. Based on this proprioceptive feedback of joint angles, the network computes an internal sensory prediction $\hat{\mathbf{x}}(k)$ of the task space position utilizing the learned

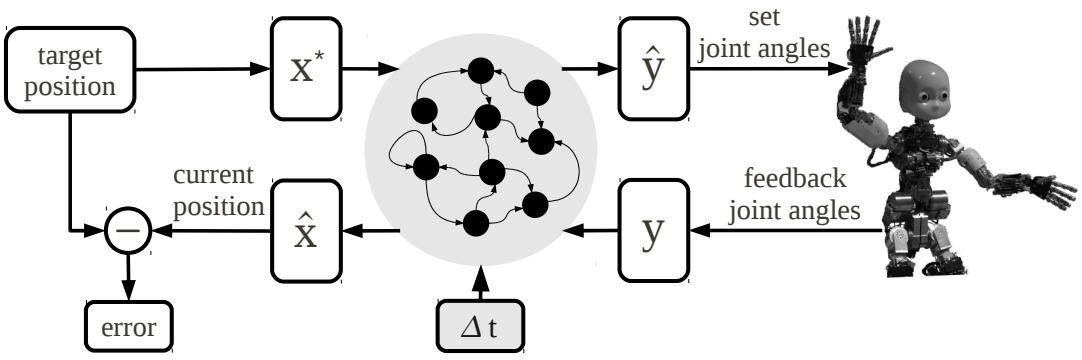



Fig. 8.3: The associative reservoir network integrated in a feedforward-feedback controller for reaching movement generation. Target positions $\mathbf{x}^*$ and current joint angles $\mathbf{y}$ are fed into the network. Estimated joint angles $\hat{\mathbf{y}}$ and end effector position $\hat{\mathbf{x}}$ are read out. Parameterizing the network dynamics by $\Delta t$ allows to control the speed of the movements.

**Algorithm 8.7** Generation of reaching movements

**Require:** get target position $\mathbf{x}^*$
**Require:** set $\Delta e = \infty$, $\delta_{target} = 10^{-6}$ and $k = 0$
1: **while** $\Delta e > \delta_{target}$ **do**
2: inject target $\mathbf{x}^*$ into network
3: execute network iteration (3.2), (8.1)–(8.3)
4: compute error change $\Delta e = |e(k) - e(k-1)|$,
where $e(k) = \|\mathbf{x}^* - \hat{\mathbf{x}}(k)\|$
5: execute (non-blocking) movement to joint angles $\hat{\mathbf{y}}(k)$
6: feedback actual joint angles into network (optional)
7: $k = k+1$
8: **end while**

forward model. The notion of feedforward-feedback control is due to the fact that the network can estimate control variables directly from the command variables in a feed-forward manner. In combination with the slowed down network dynamics and integration of proprioceptive signals from the control variables, the control scheme is also iterative and incorporates feedback.

In order to detect whether the target is reached, I calculate the task space error $e(k) = ||\mathbf{x}^* - \hat{\mathbf{x}}(k)||$ based on the learned forward model. The absolute error change $\Delta e(k) = |e(k) - e(k-1)|$ serves as stopping criterion: If the error change $\Delta e(k)$ becomes smaller than a constant $\delta_{target}$, the controller assumes that the target is reached. Though explicit stopping of the network iteration is generally not necessary, this mechanism can provide feedback of the overall reaching progress to higher level processes and defines a clear segmentation of generated reaching movements. The control framework for reaching movement generation is condensed in Algorithm 8.7.

## 8.2 Online learning of kinematics for movement generation

In this section, I outline how NDMP networks are trained online from trajectory data in order to generate reaching movements with the humanoid robot iCub (see Fig. 8.4 (left)).

### 8.2.1 Network Setup and Training

All experiments in this section are conducted using a reservoir network with 200 hidden neurons. The network is set up with three inputs corresponding to the end effector position of iCub's right

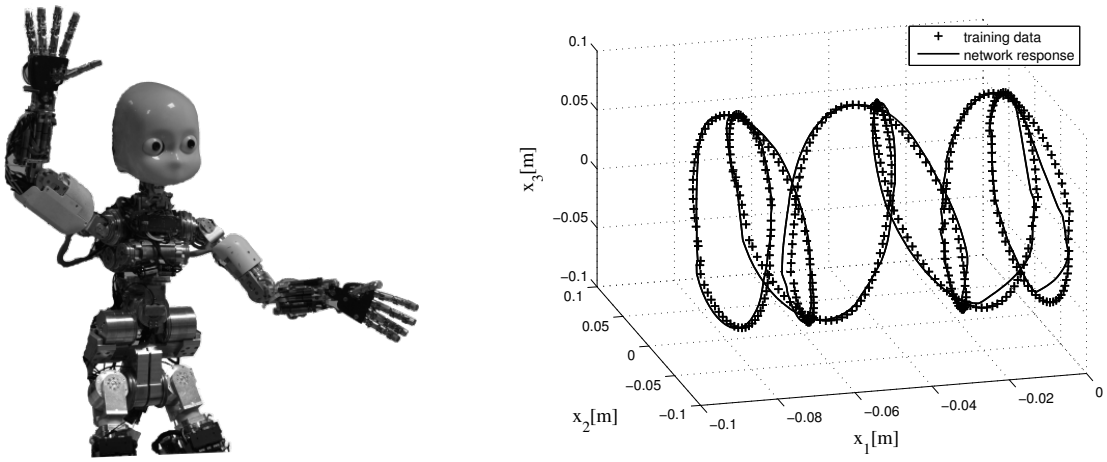



Fig. 8.4: The iCub robot [163] (left). Training data for the right arm of iCub in task space with network response (right).

arm in task space, and seven output neurons encoding the respective joint angles. The weight matrices $\mathbf{W}^{inp}$, $\mathbf{W}^{res}$ and $\mathbf{W}^{fdb}$, which excite the reservoir, are sparse and randomly initialized. I initialize only 40% randomly chosen entries of $\mathbf{W}^{inp}$ and $\mathbf{W}^{fdb}$ with values according to uniform distributions in $[-0.4, 0.4]$ and $[-0.1, 0.1]$, respectively. 20% of the entries in $\mathbf{W}^{res}$ are initialized in range $[-0.05, 0.05]$. Read-out weights $\mathbf{W}^{out}$ and $\mathbf{W}^{rec}$ are fully connected and set to zero initially. The reservoir comprises parameterized Fermi-neurons (3.4) that I train by intrinsic plasticity (IP, see Sec. 3.2.5). IP provides a parameter to adjust the desired mean activity level $\mu$ of the reservoir neurons. I set $\mu = 0.2$, corresponding to a sparse network activity, and use the learning rate $\eta_{IP} = 0.00002$. Supervised learning of $\mathbf{W}^{out}$ and $\mathbf{W}^{rec}$ is conducted online according to (3.17) with $\Delta t = 1$ and learning rate $\eta = 0.01$. In the training phase, network inputs $\mathbf{x}(k)$ and outputs $\mathbf{y}(k)$ are teacher forced: Input and output neurons are clamped to the target values before iterating the network dynamics. Initial transients are washed out prior to learning by a teacher-forced convergence phase with the first data sample $(\mathbf{x}(1), \mathbf{y}(1))$. Note that $\Delta t = 1$ is used during learning because then the network can follow the input sequence more rapidly and thus training programs the static kinematic mappings into the attractor states even though it is driven by a time-series (see [83] and compare discussion on transient and attractor-based computation in Sec. 3.2.4).

A spiral-like motion is used as training pattern in this scenario. In task space, end effector positions are defined by

$$x_1(k) = -0.08 \ (0.5 \, sin(\bar{\omega} \, k) + 0.5) \tag{8.4}$$

$$x_2(k) = 0.08 \, cos(6 \, \bar{\omega} \, k) \tag{8.5}$$

$$x_3(k) = 0.08 \, sin(6 \, \bar{\omega} \, k). \tag{8.6}$$

This motion is constrained to a cylinder of $8cm$ in length and with a diameter of $16cm$ (see Fig. 8.4 (right)). The Cartesian end effector coordinates $(x_1, x_2, x_3)$ are presented in meters and have their point of origin at the end effector position of iCub's right arm in the home position. To excite the network with values in a reasonable range, the network gets task space inputs in decimeters and joint angles in radians. I set $\bar{\omega} = 2\pi/400$ and record two pattern periods ($K = 800$ samples) as training data $(\mathbf{x}(k), \mathbf{y}(k))$ using a Jacobian-based inverse kinematics solver: I compute for each task space input $(x_1(k), x_2(k), x_3(k))$ the seven corresponding joint angles $(y_1(k), \ldots, y_7(k))$ of iCub's right arm using a variable gain controller in combination with a steepest descent method[1]. Basically, the joint angle velocities $\dot{\mathbf{q}}$ are calculated according to

$$\dot{\mathbf{q}} = -\mathbf{J}^{+}(\mathbf{x}^{*} - \mathbf{x}), \tag{8.7}$$

where $\mathbf{J}^{+}$ is the pseudo-inverse of the Jacobian $\mathbf{J}$ of the forward kinematics. $\mathbf{x}^{*}$ and $\mathbf{x}$ are the desired and actual end effector position. Equation (8.7) is iterated until convergence to obtain the final pair of end effector position $\mathbf{x}(k)$ and corresponding joint angles $\mathbf{y}(k) = \bar{\mathbf{q}}$.

I train the network for 1000 epochs on the two periods of the training pattern shown in Fig. 8.5 (left) with the corresponding end effector positions defined by (8.4), (8.5) and (8.6). Note that I apply online learning from the training sequence without permutation of examples such that learning has to cope with temporally correlated training data.

Fig. 8.5 (left) shows the joint angle trajectories obtained by iterating (8.7) for each target of two pattern periods (periodicity shown by vertical line in Fig. 8.5). The controller does not use constraints to resolve the redundancy which results in a drifting joint angle controlling the rotation of the forearm (bold line in Fig. 8.5 (left)). This method yields ambiguous data for the learner, i.e. same inputs are mapped to different outputs. Due to the output feedback, the

[1]The implementation is part of the iCub software repository. The particular module was written by Ugo Pattacini and first released in June 2008. See `http://eris.liralab.it/iCub/main/dox/html/group__iKinInv.html` and `http://eris.liralab.it/iCub/main/dox/html/classiCub_1_1iKin_1_1VarKpSteepCtrl.html` for more details.

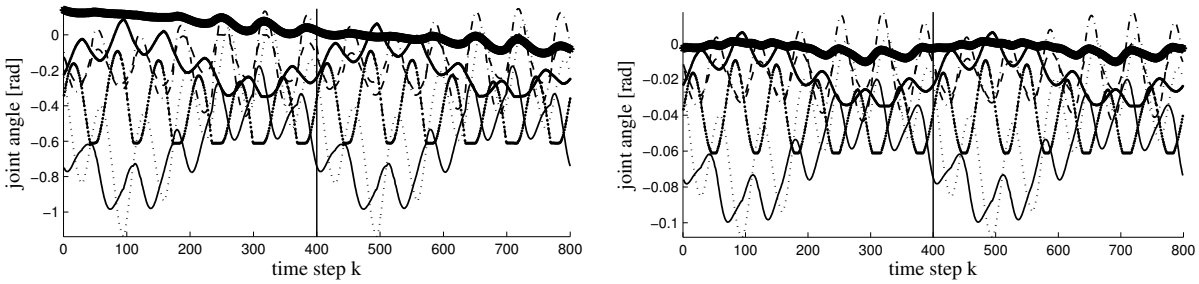



Fig. 8.5: Joint angle trajectories corresponding to the pattern shown in Fig. 8.4 (right): analytic inverse kinematic controller (left) and network outputs (right). Note that online learning resolves the drift present in the training data by adopting the samples of the second period to a stable solution.

reservoir network copes with ambiguous training data (compare discussion in Chapter 7). Although online learning does not form bi-stable output feedback dynamics due to its regularizing nature (compare discussion in Sec. 5.3), the network approximates the solution present in the second period of the training data very well. That is, the network does not form a compromise (average) between both solutions. I found a consistent behavior over several network initialization that the networks decide for the solution of the second period in the data. This seems to be due to the combination of online learning and sequential data presentation.

### 8.2.2 Movement generation

Movement generation proceeds by applying reaching targets in task space to the input of the network. Exemplary movements are shown in Fig. 8.6 (a)–(b) which have been generated by the control framework presented in Sec. 8.1.3 using $\Delta t = 0.1$ and $\delta_{target} = 10^{-9}$ (movement direction from left to right). Target positions are always reached up to small residual errors due to the approximation of the kinematic model. Fig. 8.6 (a)–(b) also confirm that the stopping criterion based on the estimated error in task space performs as demanded. Remarkable is the smooth trajectory generation in areas where no training data was presented to the network. Fig. 8.6 (c)–(d) shows a similar reaching movement on the real robot iCub.

In the following, I systematically investigate the properties of the proposed approach. For association, I measure the generalization performance of the static inverse kinematics. For movement generation, I investigate dynamic properties of the generated trajectories. I address the following key questions: How well does the network generalize to untrained target positions? What is the typical velocity profile of a generated movement? What is the impact of different values of $\Delta t$ on the convergence behavior, and is the network robust to perturbations?

### 8.2.3 Static properties

I first test the generalization performance to unlearned targets systematically in simulation. I calculate the positioning error for the iCub arm movements in task space by

$$E(x_1, x_2, x_3) = ||(x_1, x_2, x_3) - FK(\hat{IK}(x_1, x_2, x_3)))||,$$

where $FK$ denotes the known analytic forward kinematics and $\hat{IK}$ the learned inverse kinematics. I compute errors in task space since errors in joint space can be misleading: the network could find a different solution of the redundant kinematics than the analytic model in order to approach the same target position.

The iCub arm kinematics are trained on the spiral data shown in Fig. 8.4 (right), where the positions corresponding to the estimated outputs of the network for the training set are

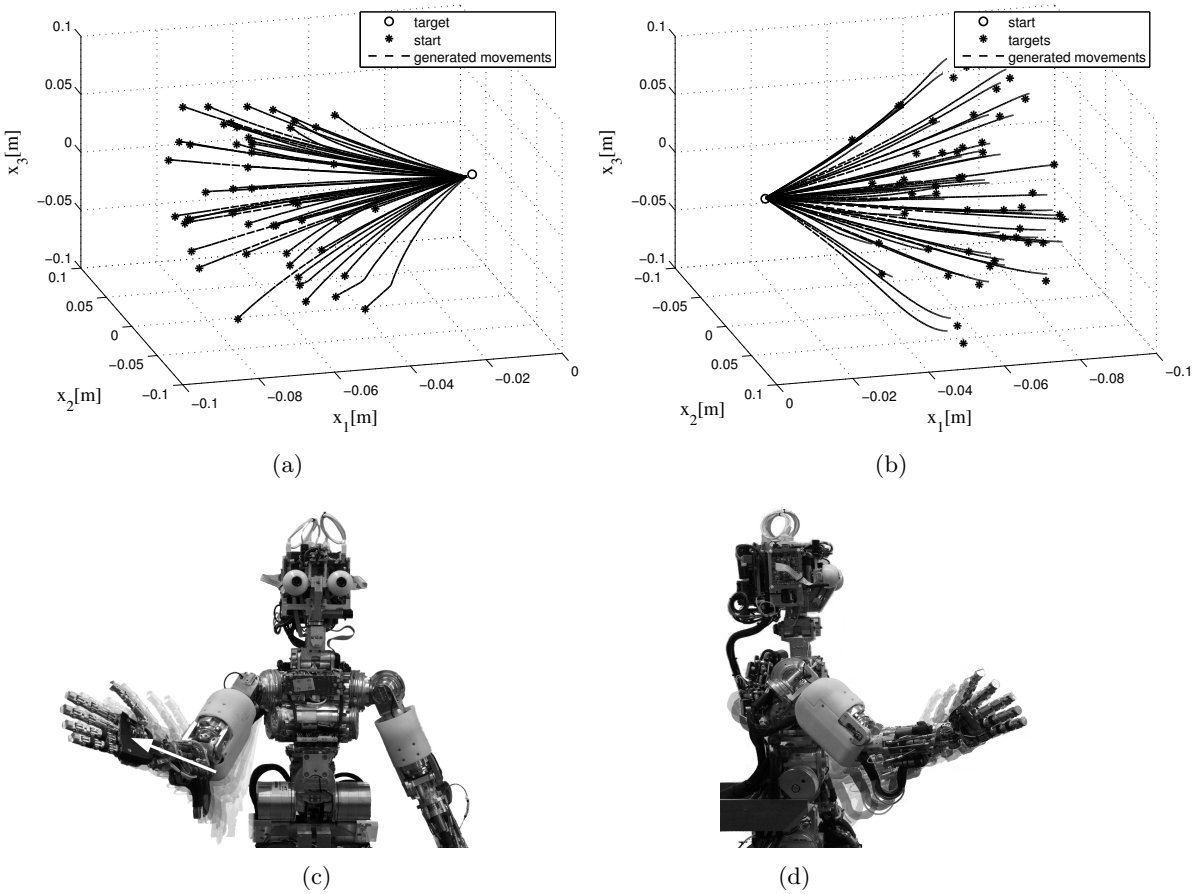



Fig. 8.6: Generalized reaching movements on iCub. The network state contracts from 50 exemplary start positions to a target attractor (a). 50 exemplary reaching movements to different targets (b). Network generated reaching movement performed on the real iCub: frontal view (c), lateral view (d).

also shown. To test the network's generalization of the inverse kinematic mapping, I query network outputs for targets sampled from a three dimensional and equally spaced grid with 50x50x50 vertices that spans a cuboid of 20x20x10$cm$ in task space. The error histogram and cumulative distribution for all 125.000 targets in the cuboid are shown in Fig. 8.7 (left). 75% of the 125.000 target positions are reached with an residual error less than 1$cm$. The maximal error is below 5$cm$.

In order to visualize the error distribution spatially in Fig. 8.7 (right), I project errors in the cuboid onto the frontal plane by marginalization with respect to the $x_1$ axis. The error surface is smooth, which means that similar errors can be expected for nearby targets. Additionally, the network displays a graceful degradation of performance away from the training data. The reservoir network generalizes well in the cuboid and achieves a mean reaching precision of 0.8$cm$, although the training samples cover only a small subset of the three dimensional volume (compare Fig. 8.4 (right)).

### 8.2.4 Gain control

In terms of control theory, the jump of the target at the network input when a new movement is initiated is equivalent to a step of the command variable. Fig. 8.8 (a) shows the responses of the network to a step of the target position from $x_2 = 0.0$ to $x_2 = 0.08$ for $\Delta t$ ranging from 0.02 to 1.0 with step width 0.02. The step response gets continuously steeper with increasing $\Delta t$ which

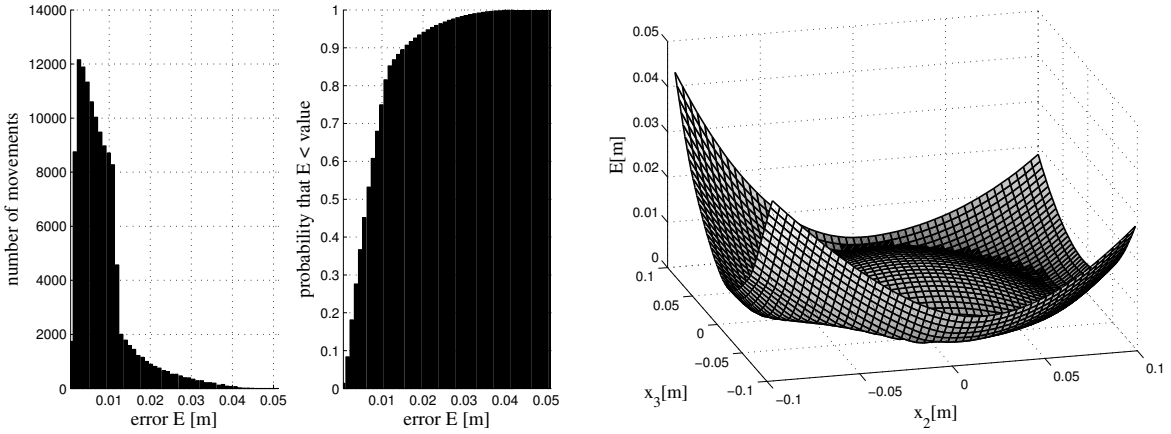


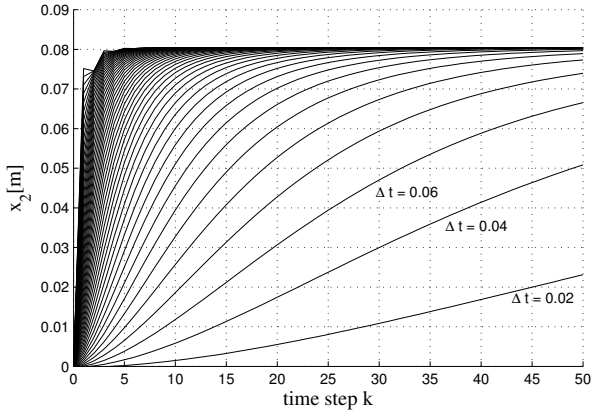

Fig. 8.7: Generalization errors of the learned inverse kinematics. Error histogram and cumulative distribution in the cuboid (left). Reaching errors in the cuboid projected onto the frontal plane (right).

shows that $\Delta t$ enables a smooth parametrization of the controller's gain. Note that for this evaluation it is crucial that the attractor states are not modified by $\Delta t$, which was shown for the fixed-points in Sec. 8.1.2. The empiric results in Fig. 8.8 (a) show that the network actually approaches the same attractor state independent of the chosen $0 < \Delta t \leq 1$: The same end effector position in response to the step at the input is finally reached in Fig. 8.8 (a) irrespective of the applied $\Delta t$.

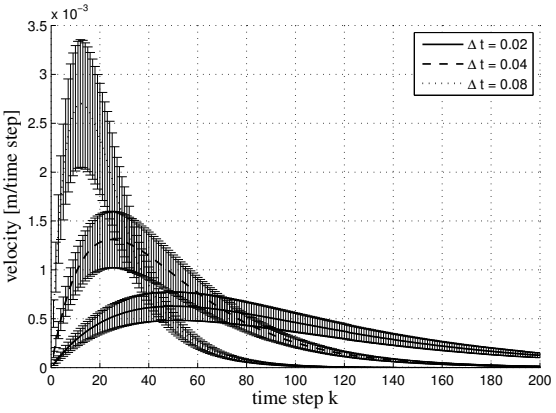
I also compute the speed profile produced by the controller when a reaching movement is performed. I record network responses during reaching movements, i.e. estimated joint angles and resulting end effector trajectories, and calculate the corresponding velocities. Fig. 8.8 (b) shows the speed profiles for different values of $\Delta t$ averaged over fifty reaching movements similar to those shown in Fig. 8.6. The velocities first reach a peak and then decay exponentially. This is a generic feature of the first order dynamics (8.1)–(8.2) that govern the neural model. The typical decay behavior shows that NDMPs are robust to strong jumps of the command variable. Decreasing $\Delta t$ leads to a damped peak of the velocities at the onset of the movements (see Fig. 8.8 (b)).

Interestingly, a linear dependency between the distance of a target and the peak velocity can be observed which is very similar to data of monkey arm movements [187]. Fig. 8.8 (c) illustrates this relation for different values of $\Delta t$: The more distant a target is, the higher is the maximal velocity of the reaching movement. The linear regression lines fitted to the data for each $\Delta t$ show a highly significant correlation. Note that this is a generic property of the network dynamics and not the result of any optimization criterion. Steeper slopes and offsets of the regression lines in Fig. 8.8 (c) also reflect the increased gain of the controller when $\Delta t$ is increased.

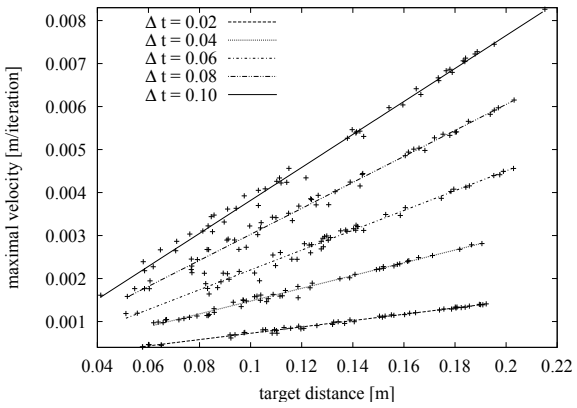
Conceptually, $\Delta t$ remains a hyper-parameter of NDMPs and is therefore depicted as external input to the reservoir network in Fig. 8.3. Parametrization of the reservoir dynamics by a temporal scaling factor has previously been performed to speed up or slow down the autonomous generation of an oscillatory "figure eight" pattern [118]. Because of error amplification, this can easily cause instability and a significant deviation of the output feedback dynamics from the target pattern. In contrast, NDMPs utilize fixed-point attractor dynamics to new targets for movement generation that are robust for a wide range of values of $\Delta t$ and, in practice, variation of $\Delta t$ only speeds up or slows down convergence. Consequently, $\Delta t$ can be used to robustly regulate the sensitivity of the system ranging from slow movements ($\Delta t \ll 1$) up to high sensitivity with fast response times ($\Delta t \approx 1$) without affecting the basic functionality.



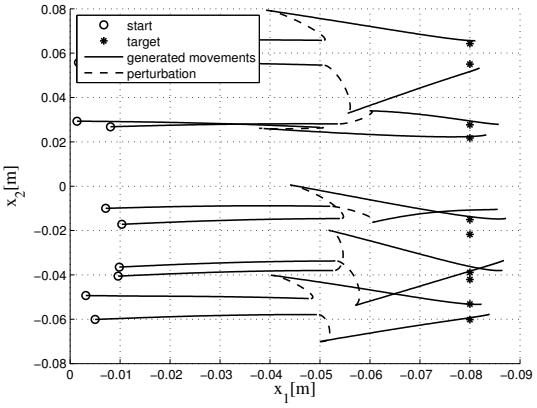

(a) Step response.




(b) Speed profiles.




(c) Speed versus distance.




(d) Perturbed reaching movements.

Fig. 8.8: Dynamic properties of generated reaching movements for different gains $\Delta t$ (a)–(c). Step response of the network (a). Averaged speed profiles of reaching movements with standard deviations (b). Linear correlation between target distance and peak velocity (c). Feedback of joint angles allows to continue with a smooth movement after perturbation of reaching movements (d).

### 8.2.5 Robustness to perturbation

To illustrate the robustness against perturbations of attractor-based movement generation with NDMPs, I disturb the system during reaching. A random deviation of up to five degrees per joint is applied for 50 iteration steps to simulate a strong hit on the arm. Fig. 8.8 (d) shows the effect of the perturbations for ten exemplary reaching movements (movement direction from left to right): The robot arm deviates strongly from the original position. Then, the neural controller picks up the reaching trajectory again when the external force is released. In this scenario, feedback of the actual joint angles into the network (see Fig. 8.3) is essential for the reaching process to respond to perturbations. Otherwise, Algorithm 8.7 would terminate if the network has converged to the attractor state even though the target position was not reached. Closing the loop enables to estimate the current end effector position and to evaluate the error change $\Delta e(k)$ such that Algorithm 8.7 can iterate the network further until convergence. Due to the attractor-based computation of the network, perturbations do not harm convergence because the attractors can be approached from arbitrary directions (compare Fig. 8.6 (left)).

## 8.3 Movement generation with multiple inverse solutions

In the previous section, forward and inverse kinematics have been trained online for movement generation in the NDMP framework. Thereby, only a single solution was learned by the network from trajectory data. In this section, movement generation by the NDMP framework is demonstrated also for the case of multiple solutions and offline training of forward and inverse kinematics from temporally non-contiguous data. We continue with the trained Associative Extreme Learning Machines from Sec. 7.4.

### 8.3.1 Movement generation with multi-stable NDMPs

I demonstrate the movement generation capabilities of NDMP when multiple solutions of the inverse kinematics have been stored in the network. The networks are trained as described in Sec. 7.4 and already represent multiple solutions to the inverse kinematics by multi-stable output feedback dynamics (see Fig. 7.12 (a)). The networks in Sec. 7.4 were trained on the first four joint angles of iCub's right arm. Note that the AELM utilizes an asynchronous update of the network (compare (3.20) in Sec. 3.3.4). AELMs can nevertheless be utilized in the NDMP framework analogously to the online trained network with synchronous update equations above. In contrast to [24] and the NDMP model discussed in the previous section, I use a decreased network speed $\Delta t = 0.05$ only for the hidden layer and $\hat{\mathbf{y}}$ (see (8.1) and (8.2)) whereas the estimated end effector position $\hat{\mathbf{x}}$ is not low-pass filtered. In doing so, I obtain an instantaneous estimation $\hat{\mathbf{x}}(k)$ of the current end effector position (equation (3.7) remains unchanged).

Exemplary reaching movements with estimated end effector positions are shown in Fig. 8.9 (left). The trajectories are smooth and straight. The end effector position is accurately estimated during reaching (compare black and gray lines in Fig. 8.9 (left)).

Additionally, I show movement generation utilizing different redundancy resolution schemes. I generate movements starting from an elbow up and down solution like shown in Fig. 7.12 (a) and (b). Fig. 8.9 shows the generated movements in task (middle) and joint space (right) and confirms that two distinct solutions to the inverse kinematics are applied throughout the movement (compare black and gray lines for each joint angle in Fig. 8.9 (right)). That is, the output feedback dynamics memorize the current redundancy resolution scheme (see discussion on attractor-based short-term memory in Sec. 7.2.3). Feedback of actual joint angles into the network directly acts as constraint in joint space and can cause the network dynamics to switch

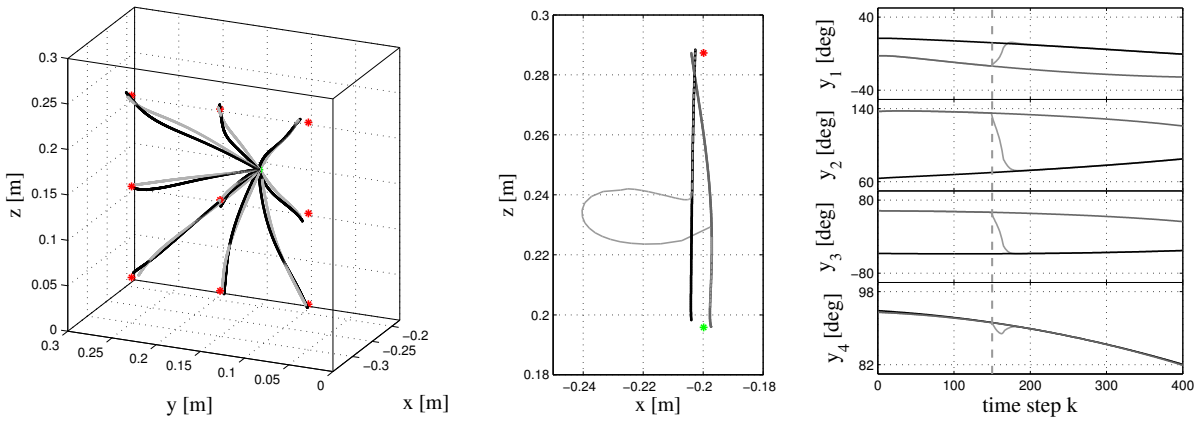



Fig. 8.9: Movement generation. Generated movements in black with estimated end effector positions in gray (left). Movement generation in task (middle) and joint space (right) utilizing two different solutions of the inverse kinematics (black and gray line). Perturbation of the robot arm can result in switching the redundancy resolution scheme (light gray line). Start of perturbation is indicated by the dashed horizontal line (right).

the redundancy resolution scheme.

To illustrate the switching of solutions by external perturbations, I generate a movement starting from the elbow up solution and then apply a perturbation in direction of the elbow down solution. The network switches smoothly to the elbow down solution (see light gray lines in Fig. 8.9 (middle) and Fig. 8.9 (right)). This output feedback-driven transition of the redundancy resolution scheme is formally described in Sec. 7.2.2.

In NDMP, movements are solely parameterized by the desired target position in task space and the speed parameter $\Delta t$. Attractor-based movement generation in combination with flexible redundancy resolution, which is responsive to external perturbations, is the main conceptual strength of the NDMP model.

### 8.3.2 Properties of the controller

In this section, I show that the trajectories generated by the AELM network display properties of human reaching movements. I therefore calculate the velocity profile of each movement. Fig. 8.10 (left) and (middle) show the raw and normalized speed profiles with movement onset at $t = 0$, where I normalized the velocity profiles according to [188] in order to account for different target distances and maximal speeds.

The profiles in Fig. 8.10 show an asymmetric, bell-shaped form with a more steep acceleration phase and a slightly prolonged deceleration. This is qualitatively similar to the results reported in [188, 176] for human reaching movements. Note, however, that there are also symmetrically velocity profiles reported in the literature [189, 187]. Symmetric velocity profiles are typically observed for tasks with low spatio-temporal accuracy constraints, whereas asymmetrical velocity profiles are observed for movements with spatial accuracy constraints (see [188] for a more detailed discussion). Adaptive control of the network step width $\Delta t$ during reaching could in principal be implemented in the future to interpolate between symmetric and asymmetric velocity profiles. However, the proposed NDMP model generically exhibits speed profiles that are structurally similar to those observed for human reaching movements with spatial accuracy constraints.

Another strong similarity to human reaching movements can be observed with respect to the relation between target distance and peak velocity. Fig. 8.10 (right) displays this relation for 100 reaching movements and confirms the results in Sec. 8.2.4 also for offline trained NDMPs with multi-stable output feedback dynamics. The linear correlation between target distance and peak velocity of the generated movements is highly significant (Spearman rank

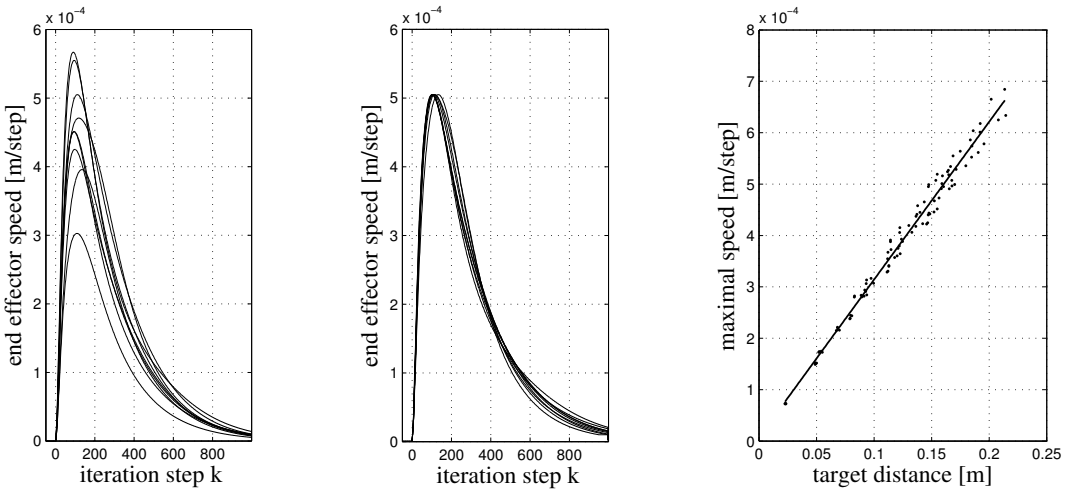



Fig. 8.10: Speed profiles with movement onset at $t = 0$: Raw profiles (left) and normalized profiles (middle). Linear relation between target distance and peak velocity (right).

correlation test with significance level 0.01). This relation is also known to rule human and monkey reaching movements [188, 187], and is closely related to Fitts' law [190] which predicts movement duration from target distance and size. The attractor-based movement generation with generic properties of human motion in combination with flexible incorporation of feedback and redundancy resolution are key features of NDMPs.

## 8.4 Concluding remarks

I conclude that NDMPs based on the associative reservoir computing paradigm provide a flexible model for movement generation with dynamical systems. Attractor-based computation in a feedforward-feedback controller is utilized for smooth movement generation where the speed of motion is controlled by a single parameter. The associative network underlying the NDMPs solves the inverse kinematics and provides an estimate of the current end effector position based on proprioceptive feedback of joint angles. NDMPs can exploit the redundancy of the actuator by utilizing multi-stable output feedback dynamics. The attractor-based short-term memory implemented by the output feedback dynamics thereby resolves redundancies consistently during movement generation. Proprioceptive feedback may result in switching between solution branches, i.e. switching the basin of attraction. Perturbations therefore "write" a particular redundancy resolution scheme to the memory: The output feedback dynamics respond to the external forces in a compliant fashion by adopting their dynamical regime. Finally, the NDMP framework generates rather straight movements which share properties of human reaching movements. An more detailed evaluation of this relation is subject to future research. NDMPs are dynamical primitives for robust movement generation based on coupled forward and inverse kinematic models.

Chapter 9

# Conclusion

In this thesis, I proposed a dynamical system approach to bidirectional association based on the concept of output feedback in reservoir networks. Reservoir networks comprise a hidden representation based on non-linear random projections and thereby render learning efficient. Association in these networks is input-driven which resolves the restriction of traditional associative memories to piecewise constant functions and enables the learning of and generalization to smooth forward and inverse mappings.

Success of learning in these reservoir networks with output feedback crucially depends on the ability to robustly shape the output feedback loop. Even though the general network configuration with output feedback has been proposed in previous work, the problem of error amplification in trained output feedback dynamics has not been tackled sufficiently. I formalized the problem of error amplification in these networks under the notion of output feedback stability. I addressed the issue of output feedback stability by a rigorous regularization concept of the read-out learning and the reservoir. Regularization of the reservoir is accomplished efficiently and in one shot based on the novel state prediction learning scheme. I showed that this two-fold regularization of the read-out layer and the reservoir indeed enables robust offline training of networks with output feedback. Moreover, the proposed state prediction approach applies more generally to input-driven recurrent neural networks and enables the efficient shaping of dynamics by solving a linear regression problem.

In the context of bidirectional association, it is essential to balance contributions of inputs and outputs to the hidden representation in order to assure that the model can be exploited in an forward and backward modus. This is particularly important in cases with very different input and output dimensionalities. I addressed this issue by integrating the constraint of balanced contributions into the reservoir regularization process. Then both, the regularization of the hidden representation and the balancing of contributions to this representation, are accomplished in one shot utilizing the efficient state prediction method.

In a next step, I demonstrated the modeling power of output feedback dynamics by imprinting multiple solution branches into the network. I showed that output feedback in principle copes with multiple outputs for the same input but suffers from narrow basins of attraction which typically results in poor generalization along a solution branch. Stabilization of solution branches is achieved by explicitly shaping attractor dynamics of the output feedback loop. The learning of ambiguous inverse problems is demonstrated on a series of experiments which show that single, multiple or even infinitely many solutions to a problem can be represented by output feedback dynamics. In particular, the number of solutions can change over the input domain of the ambiguous model which is achieved by the learning in an unsupervised manner, i.e. the number of solution branches must not be known in advance. Moreover, multi-stable output feedback dynamics implement an attractor-based short-term memory functionality which is in some respects superior to the traditional transient-based short-term memory approach in reser-

voir networks. The efficient and robust modeling of ambiguous mappings by output feedback dynamics is a key contribution of this thesis and demonstrates the capabilities of dynamical systems in comparison to pure feed-forward models.

I integrated the presented methodology into a control framework for movement generation under the notion of neural dynamic movement primitives. The coupled representation of forward and inverse kinematics in the associative network is thereby exploited as feedforward-feedback controller. Besides solving the coordinate transformation problem, the dynamical system setup provides a generic primitive for human-like reaching movement generation which is responsive to perturbations. Multi-stable output feedback dynamics, moreover, add flexibility to the movement generation process by exploiting manipulator redundancies.

The NDMP framework is a first step towards integration of the proposed methodology into a larger system context. One of the the main future challenges is to build larger architectures of the rather compact model presented in this thesis. For example, the extension of the presented method to shallow, localized modules can be considered to cover larger areas of a robot's workspace. In such systems, the robustness of the basic modules is a necessary prerequisite. The methods and formalizations presented in this thesis provide a promising starting point for this purpose. But also the refinement of aspects already mentioned in this thesis imply future research directions. In particular, the parametrization of contributions from input modalities to the hidden representation in order to flexibly weight their influence by means of a parameterized constraint satisfaction mechanism, and the refinement of the algorithm to imprint multi-stable output feedback dynamics for improved capturing of specific properties of solution branches are particularly interesting. In the context of movement generation, a next step is the incorporation of trajectory information to shape the specific convergence behavior to solution branches such that additional characteristics of demonstrated movements can be integrated into the neural dynamic movement primitives.

It is widely accepted that dynamical systems provide a decent methodology for computation but practical issues, in particular the combination of adaptability and stability, impeded their application to a great extent. The presented examples show that it is feasible to model complex relationships with dynamical systems if the problems of stability and learning are tackled. This thesis provides a coherent framework to address these issues in a connectionist model and the results are promising for future applications of adaptive dynamical systems to problems that can no be solved in a pure feed-forward manner.

Appendix A

# Appendix

## A.1 Solving the dual problem

Insertion of (4.11) into (4.8) yields the dual form of the primary optimization problem $L(\mathbf{W}^{net}, \lambda)$:

$$\begin{aligned} L_{dual}(\lambda) &= R(\mathbf{W}^{net}_{can}) - C(\mathbf{W}^{net}_{can}, \lambda) \\ &= \frac{1}{2}\sum_{i=1}^{R}\sum_{j=1}^{N}\left(\sum_{k=1}^{K}\lambda_i(k)s_j(k)\right)^2 \\ -\sum_{k=1}^{K}\sum_{i=1}^{R}\lambda_i(k)&\left[\left(\sum_{j=1}^{N}s_j(k)\sum_{n=1}^{K}\lambda_i(n)s_j(n)\right) - a_i(k+1)\right]. \end{aligned}$$

Optimization requires partial derivation of $L_{dual}(\lambda)$ with respect to all $\lambda_m(l)$ for $m = 1, \dots R$ and $l = 1, \dots, K$. We have

$$\frac{\partial}{\partial\lambda_m(l)}L_{dual}(\lambda) = \frac{\partial}{\partial\lambda_m(l)}R(\mathbf{W}^{net}_{can}) - \frac{\partial}{\partial\lambda_m(l)}C(\mathbf{W}^{net}_{can}, \lambda). \tag{A.1}$$

In the following, I calculate the partial derivatives for the regularizer $R(\mathbf{W}^{net}_{can})$ and constraints $C(\mathbf{W}^{net}_{can}, \lambda)$ separately.

The partial derivative of the regularizer $R(\mathbf{W}^{net}_{can}, \lambda)$ is

$$\frac{\partial}{\partial\lambda_m(l)}R(\mathbf{W}^{net}_{can}, \lambda) = \sum_{j=1}^{N}s_j(l)\sum_{k=1}^{K}\lambda_m(k)s_j(k). \tag{A.2}$$

For partial derivation of $C(\mathbf{W}^{net}_{can}, \lambda)$, I rewrite the constraints

$$\begin{aligned} C(\mathbf{W}^{net}_{can}, \lambda) &= -\sum_{k=1}^{K}\sum_{i=1}^{R}\lambda_i(k)a_i(k+1) \\ &+\sum_{k=1}^{K}\sum_{i=1}^{R}\lambda_i(k)\sum_{j=1}^{N}s_j(k)\sum_{n=1}^{K}\lambda_i(n)s_j(n). \end{aligned}$$

The partial derivative of the constraints is

$$\begin{aligned} &\frac{\partial}{\partial\lambda_m(l)}C(\mathbf{W}^{net}_{can}, \lambda) = \\ &-a_m(l+1) + \frac{\partial}{\partial\lambda_m(l)}\sum_{k=1}^{K}\sum_{i=1}^{R}\lambda_i(k)\sum_{j=1}^{N}s_j(k)\sum_{n=1}^{K}\lambda_i(n)s_j(n). \end{aligned} \tag{A.3}$$

We concentrate on the last term:

$$\begin{aligned}
&\sum_{k=1}^{K}\sum_{i=1}^{R}\lambda_i(k)\sum_{j=1}^{N}s_j(k)\sum_{n=1}^{K}\lambda_i(n)s_j(n)\\
&=\sum_{k\neq l}\sum_{i=1}^{R}\lambda_i(k)\sum_{j=1}^{N}s_j(k)\sum_{n=1}^{K}\lambda_i(n)s_j(n)\\
&\quad+\sum_{i=1}^{R}\lambda_i(l)\sum_{j=1}^{N}s_j(l)\sum_{n=1}^{K}\lambda_i(n)s_j(n).
\end{aligned}$$

Partial derivation yields:

$$\begin{aligned}
&\frac{\partial}{\partial\lambda_m(l)}\sum_{k\neq l}\sum_{i=1}^{R}\lambda_i(k)\sum_{j=1}^{N}s_j(k)\sum_{n=1}^{K}\lambda_i(n)s_j(n)\\
&\quad+\frac{\partial}{\partial\lambda_m(l)}\sum_{i=1}^{R}\lambda_i(l)\sum_{j=1}^{N}s_j(l)\sum_{n=1}^{K}\lambda_i(n)s_j(n)\\
&\qquad=\sum_{k\neq l}\lambda_m(k)\sum_{j=1}^{N}s_j(k)s_j(l)\\
&\quad+\sum_{j=1}^{N}s_j(l)\sum_{n=1}^{K}\lambda_m(n)s_j(n)+\lambda_m(l)\sum_{j=1}^{N}s_j(l)s_j(l)\\
&=\sum_{k=1}^{K}\lambda_m(k)\sum_{j=1}^{N}s_j(k)s_j(l)+\sum_{j=1}^{N}s_j(l)\sum_{n}\lambda_m(n)s_j(n).
\end{aligned}\tag{A.4}$$

Inserting the partial derivatives (A.2) and (A.3), where (A.4) is the partial derivative of the last term in (A.3), into (A.1) yields the solution

$$\begin{aligned}
&\frac{\partial L_{dual}}{\partial\lambda_m(l)}=a_m(l+1)-\sum_{j=1}^{N}s_j(l)\sum_{n=1}^{K}\lambda_m(n)s_j(n)=0\\
&\qquad\Leftrightarrow\sum_{n=1}^{K}\lambda_m(n)\sum_{j=1}^{N}s_j(l)s_j(n)=a_m(l+1).
\end{aligned}$$

## A.2 Reservoir regularization for initially Gaussian distributed weights

In this section, results for the same inverse kinematics learning example as described in Sec. 5.5 are presented where initially Gaussian distributed network weights are used. That is, the weights $\mathbf{W}^{inp}$, $\mathbf{W}^{res}$ and $\mathbf{W}^{fdb}$ are initialized according to a Gaussian distribution with standard deviation 0.5. Then, the spectral radius of $\mathbf{W}^{res}$ is scaled to 0.9.

Fig. A.1 (left) shows the results for the output feedback stability-related measure (5.6). Increasing the read-out regularization parameter $\alpha$ stabilizes the output feedback dynamics. Reservoir regularization mitigates this parameter dependence: For stronger reservoir regularization by increasing $\beta$, the error (5.6) is much smaller for a wide range of read-out regularization parameters $\alpha$. Fig. A.1 (right) confirms that reservoir regularization reduces the weight norm with increasing regularization parameter $\beta$ also in case of initially Gaussian distributed weights.

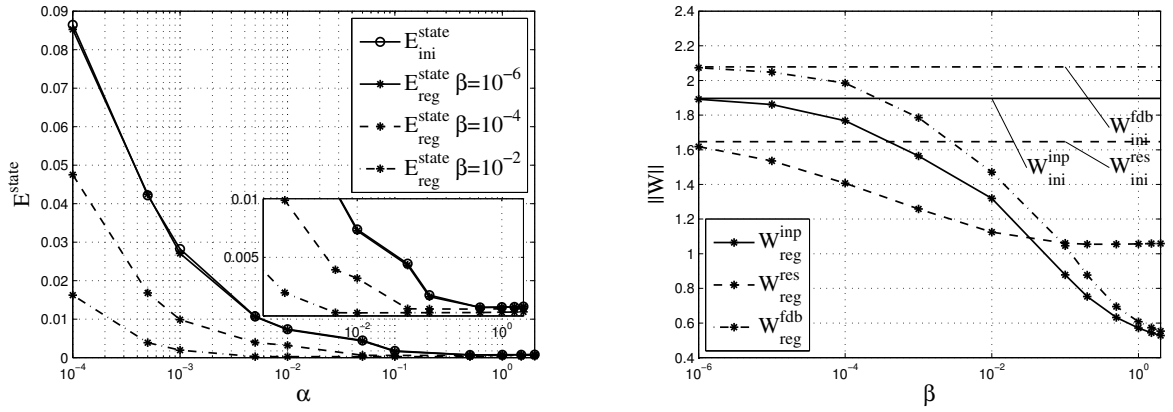



Fig. A.1: Deviation $E^{state}$ of the output feedback-driven state sequence from the teacher-forced state sequence for the training scenario as function of the read-out regularization parameter $\alpha$ (left). Matrix norm of weights $\mathbf{W}^{inp}$, $\mathbf{W}^{res}$ and $\mathbf{W}^{fdb}$ as function of the reservoir regularization parameter $\beta$ (right). Subscripts indicate whether the networks where regularized ($reg$) before read-out training, or not ($ini$).

## A.3 Network initialization and learning parameters

In this section, details of the initialization and training procedures omitted in the main text are given. The weight matrices are densely initialized if not stated otherwise.

### A.3.1 A tiny example

An AELM (see Sec. 3.3.4) with $R = 50$ Fermi-neurons (3.4) is initialized using $a^{inp} = a^{fdb} = 1$ and $\rho^{inp} = \rho^{fdb} = 1$. Learning proceeds offline according to Sec. 3.2.4. The regularization parameters are set to $\alpha^{out} = \alpha^{rec} = 10^{-3}$ and $\beta^{inp} = \beta^{fdb} = 0.1$, where I use the activity distribution method described in Sec. 5.6.2 with $g^{inp} = g^{fdb} = 0.5$.

### A.3.2 Introductory example to resolving ambiguity with output feedback

The data is produced by normalizing a parabola with 50 noisy samples per branch. Networks have $R = 100$ hidden neurons with hyperbolic tangent activation functions (3.5) and different connectivity patterns: First, an Extreme Learning Machine is considered with input weights $\mathbf{W}^{inp}$ initialized uniformly in $[-1, 1]$. Then, an Associative Extreme Learning Machine is applied

which comprises additional output feedback weights $\mathbf{W}^{fdb}$ that are also initialized uniformly in $[-1, 1]$. The biases $\mathbf{b}$ and slopes $\mathbf{s}$ of the activation functions are initialized uniformly in $[-1, 1]$.

Reservoir regularization with balancing of contributions according to Sec. 5.6.2 is applied to all networks with the following parameters depending on the network configuration: (a) In the ELM, only inputs excite the hidden layer. I nevertheless use the reservoir regularization technique according to (5.9) using $g^{inp} = 1$ and $\beta^{inp} = 0.01$. (b) In the AELM, inputs and outputs excite the hidden layer. I use $g^{inp} = g^{fdb} = 0.5$ as well as $\beta^{inp} = \beta^{fdb} = 0.01$ in (5.9) and (5.11) in order to balance their contribution to the hidden state. Read-out learning is conducted in all models using $\alpha^{out} = 10^{-3}$ in (3.14). The explicit imprinting of multiple solutions by Algorithm 7.6 is conducted with $K = 3$ and $l = 0.1$, where $\alpha^{out} = 10^{-3}$ is used in (7.2).

### A.3.3 Multi-stable and continuous attractor dynamics

An AELM (see Sec. 3.3.4) with $R = 300$ hidden Fermi-neurons (3.4), no input (consider $\mathbf{x}(k) \equiv \mathbf{0}$) and two output neurons is used. The feedback weights $\mathbf{W}^{fdb}$ as well as the biases $\mathbf{b}$ are initialized uniformly in $[-1, 1]$.

Training of multi-stable output feedback dynamics proceeds offline according to Algorithm 7.6 with $K = 10, l = 0.1$ and read-out regularization parameter $\alpha^{out} = 10^{-4}$. Note that in this example the reservoir regularization technique is not applied due to the few training samples. The output feedback dynamics are nevertheless (locally) stable because of the rather many sequence steps ($K = 10$ in Algorithm 7.6) which explicitly enforce the output feedback dynamics to have attractor states.

### A.3.4 Inverse graphics: Disentangling Lissajous figure rotations

All networks considered in this section, i.e. Extreme Learning Machines, Echo State Networks and Associative Extreme Learning Machines, comprise a hidden layer with $R = 200$ neurons and hyperbolic tangent activation functions (3.5). The input images have $13 \times 13$ pixels which are concatenated row-wise into vectors $\mathbf{x}(k) \in \mathbb{R}^{169}$. Outputs encode the virtual rotation angle $\theta$ of the Lissajous figures using a coordinate representation, i.e. $\mathbf{y}(k) = (sin(\theta), cos(\theta))^T \in \mathbb{R}^2$, to prevent discontinuities or diverging output values for the angular variable.

The network weights are initialized in the same intervals for all network models. However, each model may comprise only some of the weights: (a) The input weights $\mathbf{W}^{inp}$ are initialized uniformly in $[-\frac{1}{169}, \frac{1}{169}]$ (ELM, ESN, AELM). (b) The Echo State Networks have in addition recurrent reservoir connections $\mathbf{W}^{res}$ which are initialized in $[-1, 1]$ and then scaled such that a spectral radius of $\lambda_{max}(\mathbf{W}^{res}) = 0.95$ is achieved. (c) The AELM has, in addition to the input weights $\mathbf{W}^{inp}$, output feedback connections $\mathbf{W}^{fdb}$ which are initialized uniformly in $[-2, 2]$. Biases are set to zero ($\mathbf{b} = \mathbf{0}$) and slopes to unity ($\mathbf{s} = \mathbf{1}$) for all models.

Read-out training is preceded by reservoir regularization using balancing of contributions according to Sec. 5.6.2. Different weightings of the contributions from input modalities to the hidden layer are applied depending on the network model: In the ELM, only inputs contribute to the hidden layer and I thus set $g^{inp} = 1$. In the ESN, the reservoir receives inflow from the input neurons and from the reservoir layer itself. I set $g^{inp} = g^{res} = 0.5$ which expresses an equal contribution from inputs and recent reservoir states to the hidden representation. In the AELM, inputs and outputs excite the hidden representation and I set $g^{inp} = g^{fdb} = 0.5$. The regularization of the weights according to (5.9)–(5.11) is conducted with $\beta^{inp} = \beta^{res} = \beta^{fdb} = 10^{-3}$.

Read-out training is conducted offline according to (3.14) using $\alpha^{out} = 0.01$ for the ELM and ESN. The AELM is trained according to Algorithm 7.6 using $K = 3$ and $l = 0.3$. The regularization parameter $\alpha^{out}$ of the read-out learning is also set to 0.01 for the AELM.

## A.4 Kinematics of a planar robot arm with two degrees of freedom

In this section, the forward and inverse kinematics of a planar robot arm with two degrees of freedom are described in detail. The setup of the arm annotated with the variables used in the following is shown in Fig. A.2. The length of the first link $l_1 = 1m$ and length of the second link $l_2 = 1m$ are constant throughout this thesis. The joint angles $q_1$ and $q_2$ are specified in radians and the end effector position $(x_1, x_2)$ in meters (red dot in Fig. A.2).

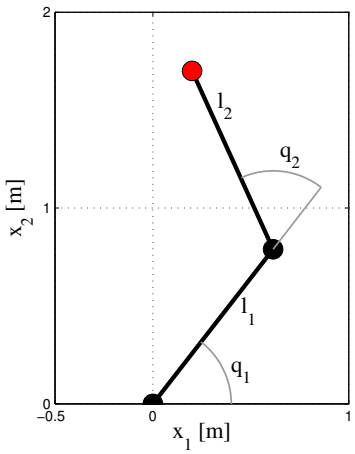



Fig. A.2: Planar robot arm with two degrees of freedom.

### A.4.1 Forward kinematics

The forward kinematics $FK : (q_1, q_2) \mapsto (x_1, x_2)$ map joint angles to Cartesian end effector coordinates as follows

$$x_1 = l_1 cos(q_1) + l_2 cos(q_1 + q_2)$$
$$x_2 = l_1 sin(q_1) + l_2 sin(q_1 + q_2).$$

### A.4.2 Inverse kinematics:

The inverse kinematics $IK : (x_1, x_2) \mapsto (q_1, q_2)$ map desired end effector positions to joint angles. Although the number of controlled variables equals the number of degrees of freedom, the inverse kinematics of the planar arm are ambiguous: There exist two solutions per position which can be described as a lefty or righty elbow configuration according to the angle of the second joint. If $q_2 \geq 0$, then the elbow is right of the end effector, i.e. the arm is in a righty configuration (this case is shown in Fig. A.2). If $q_2 < 0$, the elbow is left of the end effector in a lefty configuration. A flag $s \in \{0, 1\}$ determines which solution is applied, i.e. $s = 0$ is the lefty and $s = 1$ the righty configuration.

The inverse solution is calculated according to

$$q_1 = \arctan2(x_2, x_1) - \arctan2(k_2, k_1)$$
$$q_2 = \arctan2(a_2, a_1),$$

where

$$a_1 = \frac{\|x\|^2 - l_1^2 - l_2^2}{2 l_1 l_2}$$

$$a_2 = \begin{cases} \sqrt{1 - a_1^2} & \text{if } s = 1 \\ -\sqrt{1 - a_1^2} & \text{else} \end{cases}$$

and

$$k_1 = l_1 + l_2 a_1$$
$$k_2 = l_2 a_2.$$

The modified inverse tangent function $\arctan2(y, x)$ uses the sign of the inputs $x$ and $y$ to determine the specific quadrant in contrast to using $arctan(y/x)$.

# References


[1] Michael S. Fanselow and Andrew M. Poulos. The neuroscience of mammalian associative learning. *Annual Review of Psychology*, 56(1):207–234, 2005.

[2] Andrew Mayes, Daniela Montaldi, and Ellen Migo. Associative memory and the medial temporal lobes. *Trends in Cognitive Sciences*, 11(3):126–135, 2007.

[3] Masahiko Haruno and Mitsuo Kawato. Different neural correlates of reward expectation and reward expectation error in the putamen and caudate nucleus during stimulus-action-reward association learning. *Journal of Neurophysiology*, 95(2):948–959, 2006.

[4] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, 1982.

[5] Bart Kosko. Bidirectional associative memories. *IEEE Trans. Syst. Man Cybern.*, 18:49–60, 1988.

[6] T. Kohonen, P. Lehtiö, J. Rovamo, J. Hyvärinen, K. Bry, and L. Vainio. A principle of neural associative memory. *Neuroscience*, 2(6):1065–1076, 1977.

[7] G. Palm. On associative memory. *Biological Cybernetics*, 36:19–31, 1980. 10.1007/BF00337019.

[8] Daniel Kersten. Inverse 3-D graphics: A metaphor for visual perception. *Behavior Research Methods*, 29:37–46, 1997. 10.3758/BF03200564.

[9] Tomaso Poggio, Vincent Torre, and Christof Koch. Computational vision and regularization theory. *Nature*, 317(6035):314–319, 1985.

[10] Geoffrey Hinton. To recognize shapes, first learn to generate images. Technical Report UTML TR 2006-004, Department of Computer Science, University of Toronto, 2006.

[11] A.M. Liberman, F.S. Cooper, D.P. Shankweiler, and M. Studdert-Kennedy. Perception of the speech code. *Psychological Review*, 74(6):431–461, 1967.

[12] Alessandro D'Ausilio, Friedemann Pulvermüller, Paola Salmas, Ilaria Bufalari, Chiara Begliomini, and Luciano Fadiga. The motor somatotopy of speech perception. *Current Biology*, 19(5):381–385, 2009.

[13] D.M. Wolpert and M. Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7-8):1317–1329, 1998.

[14] Andy Clark and Rick Grush. Towards a cognitive robotics. *Adaptive Behavior*, 7(1):5–16, 1999.

[15] Michael I. Jordan and David E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16(3):307–354, 1992.

[16] Jörg A. Walter, Claudia Nölker, and Helge J. Ritter. The PSOM algorithm and applications. In *Proc. ICSC Symposion on Neural Computation (Berlin)*, pages 758–764, 2000.

[17] M. Rolf, J.J. Steil, and M. Gienger. Goal babbling permits direct learning of inverse kinematics. *IEEE Transactions on Autonomous Mental Development*, 2(3):216–229, 2010.

[18] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[19] M. Shizawa. Regularization networks for approximating multi-valued functions: learning ambiguous input-output mappings from examples. In *IEEE International Conference on Neural Networks and IEEE World Congress on Computational Intelligence*, volume 1, pages 137–142, 1994.

[20] Y. Tomikawa and K. Nakayama. Approximating many valued mappings using a recurrent neural network. In *IEEE International Joint Conference on Neural Networks*, volume 2, pages 1494–1497, 1998.

[21] Roelof K. Brouwer and Witold Pedrycz. One-to-many mappings represented on feed-forward networks. In *Proc. ESANN*, pages 365–370. d-side publi., 2009.

[22] Eric. Antonelo, Benjamin Schrauwen, and Jan Van Campenhout. Generative Modeling of Autonomous Robots and their Environments using Reservoir Computing. *Neural Processing Letters*, 26:233–249, 2007. 10.1007/s11063-007-9054-9.

[23] R. Felix Reinhart and Jochen J. Steil. Recurrent neural associative learning of forward and inverse kinematics for movement generation of the redundant PA-10 robot. In *Learning and Adaptive Behaviors for Robotic Systems (LAB-RS)*, pages 35–40, 2008.

[24] R. Felix Reinhart and Jochen J. Steil. Reaching movement generation with a recurrent neural network based on learning inverse kinematics for the humanoid robot icub. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 323–330, 2009.

[25] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[26] Michael I. Jordan. Constrained supervised learning. *Journal of Mathematical Psychology*, 36(3):396–425, 1992.

[27] H. Sebastian Seung. Learning continuous attractors in recurrent networks. In *Advances in Neural Information Processing Systems*, pages 654–660. MIT Press, 1998.

[28] Jun Tani, Masato Ito, and Yuuya Sugita. Self-organization of distributedly represented multiple behavior schemata in a mirror system: reviews of robot experiments using RNNPB. *Neural Networks*, 17(8-9):1273–1289, 2004.

[29] Jacob Barhen, Sandeep Gulati, and Michail Zak. Neural learning of constrained nonlinear transformations. *Computer*, 22:67–76, 1989.

[30] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks. Technical Report 148, German National Research Center for Information Technology, 2001.

[31] Wolfgang Maass, Prashant Joshi, and Eduardo D Sontag. Computational aspects of feedback in neural circuits. *PLoS Comput Biol*, 3(1):e165, 2007.

[32] Barbara Hammer, Benjamin Schrauwen, and Jochen J. Steil. Recent advances in efficient learning of recurrent networks. In *Proc. ESANN*, pages 213–226. d-side publi., 2009.

[33] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *IEEE International Joint Conference on Neural Networks*, volume 2, pages 985–990, 2004.

[34] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004.

[35] Lizhong Wu and John Moody. A smoothing regularizer for feedforward and recurrent neural networks. *Neural Computation*, 8:461–489, 1996.

[36] Francis wyffels, Benjamin Schrauwen, and Dirk Stroobandt. Stable Output Feedback in Reservoir Computing Using Ridge Regression. In Véra Kurková, Roman Neruda, and Jan Koutník, editors, *Artificial Neural Networks (ICANN)*, volume 5163 of *Lecture Notes in Computer Science*, pages 808–817. Springer Berlin / Heidelberg, 2008.

[37] T. Kohonen, E. Oja, and P. Lehtiö. Storage and processing of information in distributed associative memory systems. In G.E. Hinton and J.A. Anderson, editors, *Parallel Models of Associative Memory*, pages 105–143. Lawrence Erlbaum Associates, Hillsdale, NJ, 1981.

[38] Teuvo Kohonen. Correlation matrix memories. *IEEE Transactions on Computers*, C-21(4):353–359, 1972.

[39] Guilherme de A. Barreto, Aluizio F.R. Araújo, and Helge J. Ritter. Self-organizing feature maps for modeling and control of robotic manipulators. *Journal of Intelligent & Robotic Systems*, 36:407–450, 2003. 10.1023/A:1023641801514.

[40] M. Hagiwara. Multidirectional associative memory. In *International Joint Conference on Neural Networks*, volume 1, pages 3–6, 1990.

[41] Jiongtao Huang and M. Hagiwara. A new multidirectional associative memory based on distributed representation and its applications. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 1, pages 194–199, 1999.

[42] Kaoru Nakano. Associatron - A model of associative memory. *Systems, Man and Cybernetics, IEEE Transactions on*, 2(3):380 –388, 1972.

[43] Helge J. Ritter, Thomas M. Martinetz, and Klaus J. Schulten. Topology-conserving maps for learning visuo-motor-coordination. *Neural Networks*, 2(3):159–168, 1989.

[44] Thomas Martinetz, Helge J. Ritter, and Klaus Schulten. Learning of visuomotor coordination of a robot arm with redundant degrees of freedom. In *Proc. Parallel Processing in Neural Systems and Computers (ICNC)*, pages 431–434, 1989.

[45] Yoji Uno, Naohiro Fukumura, Ryoji Suzuki, and Mitsuo Kawato. A computational model for recognizing objects and planning hand shapes in grasping movements. *Neural Networks*, 8(6):839–851, 1995.

[46] Daniel M. Wolpert, R. Chris Miall, and Mitsuo Kawato. Internal models in the cerebellum. *Trends in Cognitive Sciences*, 2(9):338–347, 1998.

[47] M. Kawato, Kazunori Furukawa, and R. Suzuki. A hierarchical neural-network model for control and learning of voluntary movement. *Biological Cybernetics*, 57:169–185, 1987. 10.1007/BF00364149.

[48] M.I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. *Cognitive Science*, pages 531–546, 1986.

[49] Holk Cruse and U. Steinkühler. Solution of the direct and inverse kinematic problems by a common algorithm based on the mean of multiple computations. *Biological cybernetics*, 69(4):345–351, 1993.

[50] U. Steinkühler, W. Beyn, and H. Cruse. A simplified MMC model for the control of an arm with redundant degrees of freedom. *Neural Processing Letters*, 2:11–15, 1995. 10.1007/BF02279932.

[51] Holk Cruse, Ulrich Steinkühler, and Christan Burkamp. MMC - a recurrent neural network which can be used as manipulable body model. *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, pages 381–389, 1998.

[52] D. Verstraeten, B. Schrauwen, and D. Stroobandt. Reservoir-based techniques for speech recognition. In *International Joint Conference on Neural Networks*, pages 1050–1053, 2006.

[53] Zoubin Ghahramani. Solving inverse problems using an EM approach to density estimation. In *Proceedings of the 1993 Connectionist Models Summer School*, pages 316–323, 1993.

[54] Christopher M. Bishop. Mixture Density Networks. Technical Report NCRG/94/004, 1994.

[55] Rene Felix Reinhart and Jochen Jakob Steil. Neural learning and dynamical selection of redundant solutions for inverse kinematic control. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 564–569, 2011.

[56] H.S. Seung. How the brain keeps the eyes still. *Proceedings of the National Academy of Sciences of the United States of America*, 93(23):13339–13344, 1996.

[57] H. Sebastian Seung. Continuous attractors and oculomotor control. *Neural Networks*, 11(7-8):1253–1258, 1998.

[58] Jacob Barhen and Sandeep Gulati. *Self-organizing neuromorphic architecture for manipulator inverse kinematics*, pages 179–202. Springer-Verlag New York Inc., 1991.

[59] Jörg Walter and Helge Ritter. Associative completion and investment learning using PSOMs. In Christoph von der Malsburg, Werner von Seelen, Jan Vorbrüggen, and Bernhard Sendhoff, editors, *Artificial Neural Networks (ICANN)*, volume 1112 of *Lecture Notes in Computer Science*, pages 157–164. Springer Berlin / Heidelberg, 1996.

[60] Jörg Walter. *Rapid Learning in Robotics*. Cuvillier, 1996. PhD thesis, Bielefeld University.

[61] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[62] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982. 10.1007/BF00337288.

[63] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[64] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning (ICML)*, 2008.

[65] Hugo Larochelle, Dumitru Erhan, and Pascal Vincent. Deep learning using robust interdependent codes. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 312–319, 2009.

[66] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, 2010.

[67] R. Felix Reinhart and Jochen J. Steil. Goal-directed movement generation with a transient-based recurrent neural network controller. In *Learning and Adaptive Behaviors for Robotic Systems (LAB-RS 2009)*, pages 112–117, 2009.

[68] B.A. Pearlmutter. Gradient calculations for dynamic recurrent neural networks: a survey. *IEEE Transactions on Neural Networks*, 6(5):1212–1228, 1995.

[69] Amir F. Atiya and Alexander G. Parlos. New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Trans. Neural Networks*, 11:697–709, 2000.

[70] Wolfgang Maass, Thomas Natschläger, 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.

[71] J.J. Steil. Backpropagation-decorrelation: online recurrent learning with O(N) complexity. In *IEEE International Joint Conference on Neural Networks*, volume 2, pages 843–848, 2004.

[72] Ulf D. Schiller and Jochen J. Steil. Analyzing the weight dynamics of recurrent learning algorithms. *Neurocomputing*, 63:5 – 23, 2005.

[73] F. wyffels, B. Schrauwen, D. Verstraeten, and D. Stroobandt. Band-pass Reservoir Computing. In *IEEE International Joint Conference on Neural Networks*, pages 3204–3209, 2008.

[74] Georg Holzmann. Reservoir Computing: A powerful black-box framework for nonlinear audio processing. In *Proc. of the 12th Int. Conference on Digital Audio Effects (DAFx)*, pages 1–8, 2009.

[75] Georg Holzmann and Helmut Hauser. Echo state networks with filter neurons and a delay&sum readout. *Neural Networks*, 23(2):244–256, 2010.

[76] Kristof Vandoorne, Wouter Dierckx, Benjamin Schrauwen, David Verstraeten, Roel Baets, Peter Bienstman, and Jan Van Campenhout. Toward optical signal processing using Photonic Reservoir Computing. *Optics Express*, 16(15):11182–11192, 2008.

[77] C. Fernando and S. Sojakka. Pattern recognition in a bucket. In *European Conference on Artificial Life (ECAL)*, pages 588–597, 2003.

[78] Peter Tiňo and Georg Dorffner. Recurrent neural networks with iterated function systems dynamics. In *International ICSC/IFAC Symposium on Neural Computation*, pages 526–532, 1998.

[79] David Verstraeten and Benjamin Schrauwen. On the Quantification of Dynamics in Reservoir Computing. In Cesare Alippi, Marios Polycarpou, Christos Panayiotou, and Georgios Ellinas, editors, *Artificial Neural Networks (ICANN)*, volume 5768 of *Lecture Notes in Computer Science*, pages 985–994. Springer Berlin / Heidelberg, 2009.

[80] Peter Tiňo and Barbara Hammer. Architectural Bias in Recurrent Neural Networks: Fractal Analysis. *Neural Computation*, 15(8):1931–1957, 2003.

[81] Christian Emmerich, R. Felix Reinhart, and Jochen J. Steil. Recurrence enhances the spatial encoding of static inputs in reservoir networks. In *International Conference on Artificial Neural Networks (ICANN)*, volume 6353 of *Lecture Notes in Computer Science*, pages 148–153, 2010.

[82] Mark J. Embrechts, Luís A. Alexandre, and Jonathan D. Linton. Reservoir computing for static pattern recognition. In *Proc. ESANN*, pages 245–250. d-side publi., 2009.

[83] R. Felix Reinhart and Jochen J. Steil. Attractor-based computation with reservoirs for online learning of inverse kinematics. In *Proc. ESANN*, pages 257–262. d-side publi., 2009.

[84] L.A. Alexandre, M.J. Embrechts, and J. Linton. Benchmarking reservoir computing on time-independent classification tasks. In *International Joint Conference on Neural Networks (IJCNN)*, pages 89–93, 2009.

[85] Luís Alexandre and Mark Embrechts. Reservoir size, spectral radius and connectivity in static classification problems. In Cesare Alippi, Marios Polycarpou, Christos Panayiotou, and Georgios Ellinas, editors, *Artificial Neural Networks (ICANN)*, volume 5768 of *Lecture Notes in Computer Science*, pages 1015–1024. Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-04274-4_104.

[86] Jochen J. Steil. Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning. *Neural Networks*, 20(3):353–364, 2007.

[87] Misha Rabinovich, Ramon Huerta, and Gilles Laurent. Transient dynamics for neural processing. *Science*, 321(5885):48–50, 2008.

[88] D. Verstraeten, J. Dambre, X. Dutoit, and B. Schrauwen. Memory versus non-linearity in reservoirs. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2010.

[89] Klaus Neumann, Christian Emmerich, and Jochen Steil. Synergies between intrinsic plasticity and recurrence in reservoir networks. 2011. In preparation.

[90] Lars Büsing, Benjamin Schrauwen, and Robert Legenstein. Connectivity, Dynamics, and Memory in Reservoir Computing with Binary and Analog Neurons. *Neural Computation*, 22(5):1272–1311, 2009.

[91] Joschka Boedecker, Oliver Obst, N. Michael Mayer, and Minoru Asada. Studies on reservoir initialization and dynamics shaping in echo state networks. In *Proc. ESANN*, pages 227–232. d-side publi., 2009.

[92] Ali Ajdari Rad, Martin Hasler, and Mahdi Jalili. Reservoir optimization in recurrent neural networks using properties of kronecker product. *Logic Journal of IGPL*, 18(5):670–685, 2010.

[93] M. Buehner and P. Young. A tighter bound for the echo state property. *IEEE Transactions onNeural Networks*, 17(3):820–824, 2006.

[94] Jochen Triesch. Synergies between intrinsic and synaptic plasticity in individual model neurons. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1417–1424. MIT Press, Cambridge, MA, 2004.

[95] Klaus Neumann and Jochen J. Steil. Batch Intrinsic Plasticity for Extreme Learning Machines. In Timo Honkela, Włodzisław Duch, Mark Girolami, and Samuel Kaski, editors, *Artificial Neural Networks and Machine Learning (ICANN)*, volume 6791 of *Lecture Notes in Computer Science*, pages 339–346. Springer, 2011.

[96] Jochen Triesch. A gradient rule for the plasticity of a neuronâs intrinsic excitability. In Wlodzislaw Duch, Janusz Kacprzyk, Erkki Oja, and Slawomir Zadrozny, editors, *Artificial Neural Networks: Biological Inspirations (ICANN)*, volume 3696 of *Lecture Notes in Computer Science*, pages 65–70. Springer Berlin / Heidelberg, 2005. 10.1007/11550822_11.

[97] David Verstraeten, Benjamin Schrauwen, and Dirk Stroobandt. Adapting reservoir states to get gaussian distributions. In *Proc. ESANN*, pages 495–500. d-side publi., 2007.

[98] M. Rolf, J.J. Steil, and M. Gienger. Efficient exploration and learning of whole body kinematics. In *IEEE 8th International Conference on Development and Learning*, pages 1–7, 2009.

[99] Benjamin Schrauwen, Marion Wardermann, David Verstraeten, Jochen J. Steil, and Dirk Stroobandt. Improving reservoirs using intrinsic plasticity. *Neurocomputing*, 71(7-9):1159–1171, 2008.

[100] Petia Koprinkova-Hristova and Guenther Palm. Esn intrinsic plasticity versus reservoir stability. In Timo Honkela, Włodzisław Duch, Mark Girolami, and Samuel Kaski, editors, *Artificial Neural Networks and Machine Learning (ICANN)*, volume 6791 of *Lecture Notes in Computer Science*, pages 69–76. Springer, 2011.

[101] David Sussillo and L.F. Abbott. *Neuron*, volume 63, chapter Generating Coherent Patterns of Activity from Chaotic Neural Networks, pages 544–557. Cell Press, 2009.

[102] Gregor Michael Hörzer. Reward-modulated hebbian learning is able to induce coherent patterns of activity and simple memory functions in initially chaotic recurrent neural networks. In *Workshop on Cognitive and neural models for automated processing of speech and text (CONAS)*, Ghent, Belgium, 2010.

[103] Robert Legenstein, Steven M. Chase, Andrew B. Schwartz, and Wolfgang Maass. A reward-modulated hebbian learning rule can explain experimentally observed network reorganization in a brain control task. *The Journal of Neuroscience*, 30(25):8400–8410, 2010.

[104] W.B. Johnson and J. Lindenstrauss. Extensions of Lipshitz mapping into Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[105] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures  Algorithms*, 22(1):60–65, 2003.

[106] Sanjoy Dasgupta. Experiments with random projection. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 143–151, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[107] R.I. Arriaga and S. Vempala. An algorithmic theory of learning: robust concepts and random projection. In *40th Annual Symposium on Foundations of Computer Science*, pages 616–623, 1999.

[108] S. Kaski. Dimensionality reduction by random mapping: fast similarity computation for clustering. In *IEEE International Joint Conference on Neural Networks*, volume 1, pages 413–418, 1998.

[109] Chinmay Hegde, Michael Wakin, and Richard Baraniuk. Random projections for manifold learning. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 641–648. MIT Press, Cambridge, MA, 2008.

[110] Xiaoli Zhang Fern and Carla E. Brodley. Random projection for high dimensional data clustering: A cluster ensemble approach. In *International Conference on Machine Learning*, pages 186–193, 2003.

[111] Dmitriy Fradkin and David Madigan. Experiments with random projections for machine learning. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 517–522, New York, NY, USA, 2003. ACM.

[112] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, pages 245–250, New York, NY, USA, 2001. ACM.

[113] Ping Li and Trevor Hastie. A unified near-optimal estimator for dimension reduction in $l_\alpha$ $(0 < \alpha \leq 2)$ using stable random projections. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 905–912. MIT Press, Cambridge, MA, 2008.

[114] Guang-Bin Huang, Lei Chen, and Chee-Kheong Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, 17(4):879–892, 2006.

[115] R. Felix Reinhart and Jochen J. Steil. Reservoir regularization stabilizes learning of Echo State Networks with output feedback. In *Proc. ESANN*, pages 59–64. d-side publi., 2011.

[116] Herbert Jaeger. Adaptive nonlinear system identification with echo state networks. In *Advances in Neural Information Processing Systems*, pages 593–600. MIT Press, Cambridge, MA,, 2002.

[117] Jeffrey L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225, 1991. 10.1007/BF00114844.

[118] Herbert Jaeger, Mantas Lukosevicius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20(3):335–352, 2007.

[119] F. wyffels and B. Schrauwen. Design of a central pattern generator using reservoir computing for learning human motion. In *ECSIS Symp. on LAB-RS*, pages 118–122, 2009.

[120] 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. Springer, 2009.

[121] Masato Ito and Jun Tani. On-line imitative interaction with a humanoid robot using a dynamic neural network model of a mirror system. *Adaptive Behavior*, 12(2):93–115, 2004.

[122] Benjamin Schrauwen, David Verstraeten, and Jan Van Campenhout. An overview of reservoir computing: theory, applications and implementations. In *Proc. ESANN*, pages 471–482. d-side publi., 2007.

[123] Jochen J. Steil. Online stability of backpropagation-decorrelation recurrent learning. *Neurocomputing*, 69(7-9):642–650, 2006.

[124] G. Leonov. Strange attractors and classical stability theory. *Nonlinear Dynamics and Systems Theory*, 8(1):49–96, 2008.

[125] K. Doya. Bifurcations in the learning of recurrent neural networks. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 6, pages 2777–2780, 1992.

[126] R. Felix Reinhart and Jochen J. Steil. State Prediction: A Constructive Method to Program Recurrent Neural Networks. In Timo Honkela, Wlodzislaw Duch, Mark Girolami, and Samuel Kaski, editors, *Artificial Neural Networks and Machine Learning (ICANN)*, volume 6791 of *Lecture Notes in Computer Science*, pages 159–166. Springer Berlin / Heidelberg, 2011.

[127] R. Reinhart and Jochen Steil. A constrained regularization approach for input-driven recurrent neural networks. *Differential Equations and Dynamical Systems*, 19:27–46, 2011. 10.1007/s12591-010-0067-x.

[128] Herbert Jaeger. Reservoir self-control for achieving invariance against slow input distortions. Technical Report 23, Jacobs University, 2010.

[129] C. Radhakrishna Rao. A note on a generalized inverse of a matrix with applications to problems in mathematical statistics. *Journal of the Royal Statistical Society. Series B (Methodological)*, 24(1):152–158, 1962.

[130] A.N. Tikhonov and V.Y. Arsenin. *Solution of Ill-posed Problems*. Winston & Sons, 1977.

[131] B. Kosko. Unsupervised learning in noise. *IEEE Transactions on Neural Networks*, 1(1):44–57, 1990.

[132] Akio Yoshida and Yuko Osana. Chaotic complex-valued multidirectional associative memory with variable scaling factor. In Timo Honkela, Włodzisław Duch, Mark Girolami, and Samuel Kaski, editors, *Artificial Neural Networks and Machine Learning (ICANN)*, volume 6791 of *Lecture Notes in Computer Science*, pages 266–274. Springer, 2011.

[133] S. Chen, S.A. Billings, and W. Luo. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control*, 50:1873–1896, 1989.

[134] J.T. Connor, R.D. Martin, and L.E. Atlas. Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, 5(2):240–254, 1994.

[135] Randall D. Beer. Dynamical approaches to cognitive science. *Trends in Cognitive Sciences*, 4(3):91–99, 2000.

[136] Robert Haschke and Jochen J. Steil. Input space bifurcation manifolds of recurrent neural networks. *Neurocomputing*, 64:25–38, 2005. Trends in Neurocomputing: 12th European Symposium on Artificial Neural Networks 2004.

[137] Randall D. Beer. On the Dynamics of Small Continuous-Time Recurrent Neural Networks. *Adaptive Behavior*, 3(4):469–509, 1995.

[138] Peter Tiňo, Bill G. Horne, and C. Lee Giles. Attractive periodic sets in discrete-time recurrent networks (with emphasis on fixed-point stability and bifurcations in two-neuron networks). *Neural Computation*, 13(6):1379–1414, 2001.

[139] Robert Haschke. *Bifurcations in Discrete-Time Neural Networks – Controlling Complex Network Behaviour with Inputs*. 2003. PhD thesis, Bielefeld University.

[140] Yann Lecun and Corinna Cortes. The MNIST database of handwritten digits, 1998. `http://yann.lecun.com/exdb/mnist/`.

[141] D.P. Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic Press, 1982.

[142] S. Klanke, D. Lebedev, R. Haschke, J. Steil, and H. Ritter. Dynamic path planning for a 7-DOF robot arm. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3879–3884, 2006.

[143] P. Dahm and F. Joublin. Closed form solution for the inverse kinematics of a redundant robot arm. Technical Report IRINI 97-08, Ruhr-Universität Bochum, Institut für Neuroinformatik, 1997.

[144] Herbert Jaeger. Short term memory in echo state networks. GMD-Report 152, German National Research Center for Information Technology, 2002.

[145] Arthur Albert. *Regression and the Moore-Penrose pseudoinverse*, volume 94 of *Mathematics in science and engineering*. Academic Press, 1972.

[146] R. Felix Reinhart and Jochen J. Steil. A constrained regularization approach for input-driven recurrent neural networks. *Differential Equations and Dynamical Systems*, pages 27–46, 2011. Published online 30 December 2010.

[147] Norbert M. Mayer and Matthew Browne. Echo State Networks and Self-Prediction. In Auke Jan Ijspeert, Masayuki Murata, and Naoki Wakamiya, editors, *Biologically Inspired Approaches to Advanced Information Technology*, volume 3141 of *Lecture Notes in Computer Science*, pages 40–48. Springer Berlin / Heidelberg, 2004.

[148] J.J.E. Slotine and W. Li. *Applied Nonlinear Control*. Prentice Hall, 1991.

[149] D.P. Mandic, J.A. Chambers, and M.M. Bozic. On global asymptotic stability of fully connected recurrent neural networks. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 6:3406–3409, 2000.

[150] Carlos D. Brody, Ranulfo Romo, and Adam Kepecs. Basic mechanisms for graded persistent activity: discrete attractors, continuous attractors, and dynamic representations. *Current Opinion in Neurobiology*, 13(2):204–211, 2003.

[151] Don Riley, 2007. http://www.mathworks.com/matlabcentral/fileexchange/14932-3d-puma-robot-demo.

[152] Klaus Neumann, Matthias Rolf, Jochen Steil, and Michael Gienger. Learning inverse kinematics for pose-constraint bi-manual movements. In Stéphane Doncieux, Benoît Girard, Agnès Guillot, John Hallam, Jean-Arcady Meyer, and Jean-Baptiste Mouret, editors, *From Animals to Animats 11*, volume 6226 of *Lecture Notes in Computer Science*, pages 478–488. Springer Berlin / Heidelberg, 2010.

[153] M. Rolf, J.J. Steil, and M. Gienger. Learning flexible full body kinematics for humanoid tool use. In *International Conference on Emerging Security Technologies (EST)*, pages 171–176, 2010.

[154] P.I. Corke. A robotics toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 3(1):24–32, 1996.

[155] Laurens van der Maaten. Learning a parametric embedding by preserving local structure. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, pages 384–391, 2009.

[156] Kerstin Bunte, Michael Biehl, and Barbara Hammer. Supervised dimension reduction mappings. In *Proc. ESANN*, pages 281–286. d-side publi., 2011.

[157] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. `http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DeepVsShallowComparisonICML2007`.

[158] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[159] Laurens van der Maaten. t-Distributed Stochastic Neighbor Embedding, 2010. `http://homepage.tudelft.nl/19j49/t-SNE.html`.

[160] Stephen Grossberg. How does a brain build a cognitive code? *Psychological Review*, 87(1):1–51, 1980.

[161] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[162] Mikhail I. Rabinovich, Pablo Varona, Allen I. Selverston, and Henry D.I. Abarbanel. Dynamical principles in neuroscience. *Rev. Mod. Phys.*, 78:1213–1265, 2006.

[163] Giorgio Metta, Giulio Sandini, David Vernon, Lorenzo Natale, and Francesco Nori. The icub humanoid robot: an open platform for research in embodied cognition. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, PerMIS '08, pages 50–56, New York, NY, USA, 2008. ACM.

[164] V. Tikhanoff, A. Cangelosi, P. Fitzpatrick, G. Metta, L. Natale, and F. Nori. An open-source simulator for cognitive robotics research: the prototype of the iCub humanoid robot simulator. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, PerMIS '08, pages 57–61, New York, NY, USA, 2008. ACM.

[165] J.A. Lissajous. Mémoire sur l'étude optique des mouvements vibratoires. *Annales de Chimie et de Physique*, 51(3):140–231, 1849.

[166] Julio Castiñeira Merino. Lissajous Figures and Chebyshev Polynomials. *The College Mathematics Journal*, 34(2):122–127, 2003.

[167] Prashant Joshi and Wolfgang Maass. Movement generation with circuits of spiking neurons. *Neural Computation*, 17(8):1715–1738, 2005.

[168] Masato Ito and Jun Tani. Generalization in learning multiple temporal patterns using RNNPB. In *Proceedings of the 11th International Conference on Neural Information Processing*, pages 592–598, 2004.

[169] Ichiro Igari and Jun Tani. Incremental learning of sequence patterns with a modular network model. *Neurocomputing*, 72(7-9):1910–1919, 2009.

[170] A.J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1398–1403, 2002.

[171] Yiannis Demiris and Bassam Khadhouri. Hierarchical attentive multiple models for execution and recognition of actions. *Robotics and Autonomous Systems*, 54(5):361–369, 2006.

[172] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 763 –768, 2009.

[173] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Learning rhythmic movements by demonstration using nonlinear oscillators. In *In Proceedings of the IEEE/RSJ Int. Conference on Intelligent Robots and Systems (IROS)*, pages 958–963, 2002.

[174] Stefan Schaal, Auke Ijspeert, and Aude Billard. Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1431):537–547, 2003.

[175] A. D'Souza, S. Vijayakumar, and S. Schaal. Learning inverse kinematics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 298–303, 2001.

[176] Daniel Bullock and Stephen Grossberg. Neural dynamics of planned arm movements: Emergent invariants and speed-accuracy properties during trajectory formation. *Psychological Review*, 95(1):49–90, 1988.

[177] U. Pattacini, F. Nori, L. Natale, G. Metta, and G. Sandini. An experimental evaluation of a novel minimum-jerk Cartesian controller for humanoid robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1668–1674, 2010.

[178] Micha Hersch and Aude Billard. Reaching with multi-referential dynamical systems. *Autonomous Robots*, 25:71–83, 2008. 10.1007/s10514-007-9070-7.

[179] Marc Toussaint and Christian Goerick. A Bayesian view on motor control and planning. In Olivier Sigaud and Jan Peters, editors, *From Motor Learning to Interaction Learning in Robots*, volume 264 of *Studies in Computational Intelligence*, pages 227–252. Springer Berlin / Heidelberg, 2010.

[180] Oliver Herbort, Martin Butz, and Gerulf Pedersen. The SURE_REACH model for motor learning and control of a redundant arm: From modeling human behavior to applications in robotics. In Olivier Sigaud and Jan Peters, editors, *From Motor Learning to Interaction Learning in Robots*, volume 264 of *Studies in Computational Intelligence*, pages 85–106. Springer Berlin / Heidelberg, 2010.

[181] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Learning attractor landscapes for learning motor primitives. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*. MIT Press.

[182] Yiannis Demirisa and Andrew Meltzoff. The Robot in the Crib: A Developmental Analysis of Imitation Skills in Infants and Robots. *Infant and Child Development*, 17(1):43–53, 2008.

[183] Biljana Petreska and Aude Billard. Movement curvature planning through force field internal models. *Biological Cybernetics*, 100:331–350, 2009. 10.1007/s00422-009-0300-2.

[184] Tadashi Yamazaki and Shigeru Tanaka. The cerebellum as a liquid state machine. *Neural Networks*, 20(3):290–297, 2007. Echo State Networks and Liquid State Machines.

[185] Wolfgang Maass, Prashant Joshi, and Eduardo D. Sontag. Principles of real-time computing with feedback applied to cortical microcircuit models. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 835–842. MIT Press, Cambridge, MA, 2006.

[186] Masao Ito. Historical Review of the Significance of the Cerebellum and the Role of Purkinje Cells in Motor Learning. *Annals of the New York Academy of Sciences*, 978(1):273–288, 2002.

[187] Michael S.A. Graziano, Tyson N.S. Aflalo, and Dylan F. Cooke. Arm movements evoked by electrical stimulation in the motor cortex of monkeys. *Journal of Neurophysiology*, 94(6):4209–4223, 2005.

[188] C.G. Atkeson and J.M. Hollerbach. Kinematic features of unrestrained vertical arm movements. *The Journal of Neuroscience*, 5(9):2318–2330, 1985.

[189] P. Morasso. Spatial control of arm movements. *Experimental Brain Research*, 42:223–227, 1981. 10.1007/BF00236911.

[190] Paul M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, 1954.

[191] R. Felix Reinhart and Jochen J. Steil. Regularization and stability in reservoir networks with output feedback. *Neurocomputing*, 2011. In press.

# Related references by the author


[23] R. Felix Reinhart and Jochen J. Steil. Recurrent neural associative learning of forward and inverse kinematics for movement generation of the redundant PA-10 robot. In *Learning and Adaptive Behaviors for Robotic Systems (LAB-RS)*, pages 35–40, 2008.
In this paper, an early variant of associative reservoir computing with online learning rules was introduced. Short excerpts of this paper introducing the online learning rules are used in Chapter 3. However, no results of this work are reproduced in the presented thesis.

[83] R. Felix Reinhart and Jochen J. Steil. Attractor-based computation with reservoirs for online learning of inverse kinematics. In *Proc. ESANN*, pages 257–262. d-side publi., 2009.
In this paper, I first introduced attractor-based computation with non-linear reservoirs (in contrast to the linear attractor-based reservoirs introduced in [82]) and shed some light on effects of output feedback. Short excerpts of this work are used in Chapter 3.

[67] R. Felix Reinhart and Jochen J. Steil. Goal-directed movement generation with a transient-based recurrent neural network controller. In *Learning and Adaptive Behaviors for Robotic Systems (LAB-RS 2009)*, pages 112–117, 2009.
In this paper, I first presented the movement generation framework based on associative reservoir computing which is related to the content in Chapter 8.

[24] R. Felix Reinhart and Jochen J. Steil. Reaching movement generation with a recurrent neural network based on learning inverse kinematics for the humanoid robot icub. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 323–330, 2009.
This work refines the movement generation with associative reservoir networks from [67] and results are reproduced in an adopted form in Chapter 8 of this thesis.

[81] Christian Emmerich, R. Felix Reinhart, and Jochen J. Steil. Recurrence enhances the spatial encoding of static inputs in reservoir networks. In *International Conference on Artificial Neural Networks (ICANN)*, volume 6353 of *Lecture Notes in Computer Science*, pages 148–153, 2010.
A short and adopted excerpt of this collaborate work is used in Chapter 3 for the introduction of basic reservoir computing concepts.

[146] R. Felix Reinhart and Jochen J. Steil. A constrained regularization approach for input-driven recurrent neural networks. *Differential Equations and Dynamical Systems*, pages 27–46, 2011. Published online 30 December 2010.
A slightly adopted form of this paper builds the basis for Sec. 4.4.

[115] R. Felix Reinhart and Jochen J. Steil. Reservoir regularization stabilizes learning of Echo State Networks with output feedback. In *Proc. ESANN*, pages 59–64. d-side publi., 2011.
In this paper, the reservoir regularization technique is proposed for stabilizing output feedback dynamics and is partially reproduced in Chapter 5.

[191] R. Felix Reinhart and Jochen J. Steil. Regularization and stability in reservoir networks with output feedback. *Neurocomputing*, 2011. In press.
This invited paper has been accepted for publication in the *Neurocomputing* journal. It is an extension of [115] and reproduced in large part in Chapter 5. I further complemented this work in Chapter 5 of this thesis.

[126] R. Felix Reinhart and Jochen J. Steil. State Prediction: A Constructive Method to Program Recurrent Neural Networks. In Timo Honkela, Wlodzislaw Duch, Mark Girolami, and Samuel Kaski, editors, *Artificial Neural Networks and Machine Learning (ICANN)*, volume 6791 of *Lecture Notes in Computer Science*, pages 159–166. Springer Berlin / Heidelberg, 2011.
In this paper, the State Prediction approach was first introduced. The paper is reproduced, extended and more deeply related to other work in Chapter 4 of this thesis.

[55] Rene Felix Reinhart and Jochen Jakob Steil. Neural learning and dynamical selection of redundant solutions for inverse kinematic control. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 564–569, 2011.
This contribution first demonstrated the programming of multi-stable output feedback dynamics into associative reservoir networks. I reproduced the results from this paper in Chapter 7 and Chapter 8. Chapter 7 largely extends this previous work.